Ministério da
Ciência e Tecnologia

INPE-15662-TDI/1438

QSEE-TAS: EXECUÇÃO AUTOMATIZADA DE CASOS DE TESTE PARA SOFTWARE EMBARCADO EM APLICAÇÕES ESPACIAIS

Wendell Pereira da Silva

Dissertação do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Nandamudi Lankalapalli Vijaykumar e Valdivino Alexandre de Santiago Jr., aprovada em 20 de novembro de 2008.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2008/10.30.23.47>

INPE
São José dos Campos
2009

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

INPE-15662-TDI/1438

QSEE-TAS: EXECUÇÃO AUTOMATIZADA DE CASOS DE TESTE PARA SOFTWARE EMBARCADO EM APLICAÇÕES ESPACIAIS

Wendell Pereira da Silva

Dissertação do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Nandamudi Lankalapalli Vijaykumar e Valdivino Alexandre de Santiago Jr., aprovada em 20 de novembro de 2008.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m18@80/2008/10.30.23.47>

INPE
São José dos Campos
2009

Dados Internacionais de Catalogação na Publicação (CIP)

S38e Silva, Wendell Pereira da.
QSEE-TAS: execução automatizada de casos de teste para software embarcado em aplicações espaciais / Wendell Pereira da Silva. – São José dos Campos: INPE, 2009.
103p. ; (INPE-15662-TDI/1438)

Dissertação (Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 20/11/2008.

1. Teste de software. 2. Automação. 3. Aplicações espaciais. 4. Processo de teste de software. 5. Casos de teste. I.Título.

CDU 004.415.532.2

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

Computação Aplicada

Dr. Antonio Miguel Vieira Monteiro

Presidente / INPE / SJCampos - SP

Dr. Nandamudi Lankalapalli Vijaykumar

Orientador(a) / INPE / SJCampos - SP

Dr. Ricardo Varela Correa

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

Dr. Edgar Toshiro Yano

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

Dr. Márcio Eduardo Delamaro

Convidado(a) / USP-São Carlos / São Carlos - SP

Aluno (a): Wendell Pereira da Silva

São José dos Campos, 20 de novembro de 2008

"A vida é dura só para quem é mole".

Há tempos alguem me disse isso lá em Catalão, Goiás.

Á s mulheres da minha vida: Marta, Maria e Maressa.

AGRADECIMENTOS

Aos meus orientadores Valdivino Santiago e Nandamudi Vijaykumar pela dedicação e prontidão. Em especial ao Valdivino Santiago que, dentre outras coisas, me ensinou uma outra difinição sobre o que é ser "chato". Devo muito do meu amadurecimento profissional nesses últimos anos ao Valdivino.

À Financiadora de Estudos e Projetos (FINEP) pelo suporte dado ao projeto Qualidade do Software Embarcado em Aplicações Espaciais (QSEE).

À Fátima Mattiello, sempre muito zelosa na gestão do projeto QSEE, irradiando aquele astral contagiante ao mais nobre estilo *"ninguém me segura!"*.

À Ana Ambrósio, pelas valorosas observações. Certamente, não haverá vez em que não lembrarei-me da Ana quando eu ver uma seqüência de teste com zilhões de instruções.

Aos amigos Danilo Passos, Jairo Telles, Paulo Veras e Danielle Guimarães, pelas incessantes injeções de ânimo (e de falhas nos programas).

A minha esposa, Marta Regina, pela compreensão e por acreditar que um dia eu terminaria este trabalho. Se bem que eu acho que a paciência dela ou tinha acabado ou era infinita.

RESUMO

O software embarcado em satélites científicos é crítico, pois exige interações com o hardware em tempo real para, por exemplo, adquirir dados por meio de sensores, controlar atitude, controlar as cargas úteis, comunicar-se com as estações na Terra, entre outras. Uma vez que o satélite está em órbita, sua manutenção é dispendiosa dada a natureza autônoma da missão. Assim, o teste deste tipo de software demanda muito tempo e geralmente é executado em diferentes níveis (instrumento, subsistema, sistema) e em diversos modelos de hardware (engenharia, qualificação e vôo), tornando seu processo de Verificação, Validação e Teste (VV&T) um grande desafio. Portanto, automatizar a execução dos testes pode ajudar a otimizar o tempo gasto nesta atividade, possibilitando o re-uso dos casos de teste, rastrear os itens de testes e seus casos de teste, emissões de relatos de teste e formação de bases de dados com a evolução dos testes. Neste contexto, foi desenvolvida a ferramenta Qualidade do Software Embarcado em aplicações Espaciais - Teste Automatizado de Software (QSEE-TAS) cujo objetivo é automatizar a execução de testes caixa-preta (funcionais) para software embarcado em computadores de satélites e balões que usam comunicação via padrões de interface RS-232 e USB, assim como TCP/IP e portas analógicas e digitais. A QSEE-TAS também permite gerar de forma automática a documentação associada ao processo de teste. Adicionalmente, apresentam-se os resultados de uma experiência de uso da QSEE-TAS/SPAC no processo de validação de três aplicações embarcadas medindo-se a redução de custo da execução dos testes.

QSEE-TAS: AUTOMATED TEST CASE EXECUTION ON EMBEDDED SOFTWARE FOR SPACE APPLICATIONS

ABSTRACT

Software embedded in scientific satellite systems is usually critical due to, for instance, it's needed real-time behavior to interact with sensors, attitude control systems, payload control and communication systems. Once the satellite is in orbit, fixing bugs on its software is extremely expensive because of the autonomous nature of the mission. Hence, testing such software is a time consuming task and is often applied to several system levels (i.e. instruments, subsystems and systems) in several hardware models (i.e. engineering, qualification and flight models), making Verification, Validation and Testing (VV&T) a complex and challenger process. Therefore, running tests automatically may lead to optimize the efficiency of testing by creating opportunities to reuse test cases, tracking test items and their test cases, enabling the generation of the documentation related to the test process and tracking the evolution of the tests in terms of failure exposures. In this context, the QSEE-TAS/SPAC tool has been developed and it aims at automating functional test on software embedded in space platforms which uses RS-232 serial, USB, TCP/IP, and/or digital-analogic communication interfaces. Alson, the QSEE-TAS allows for automatic generation of the documentation related to the software testing process. Additionally, this work presents the results attained in terms of cost saving in the test execution of three embedded applications.

SUMÁRIO

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

1 INTRODUÇÃO 25
1.1 Motivação 26
1.2 Objetivo 26
1.3 Estrutura da dissertação 27

2 TESTE DE SOFTWARE 29
2.1 Atributos da qualidade de software sob a perspectiva do teste 29
2.2 Conceitos básicos 31
2.3 Verificação, validação e teste de software - VV&T 33
2.4 Um processo de teste e suas principais atividades 34
2.4.1 Geração de casos de teste 35
2.4.2 Execução de casos de teste 36
2.4.3 Análise dos resultados de teste 37
2.5 Teste de regressão 38
2.6 Esquemas de representação e codificação de mensagens 38
2.7 Trabalhos relacionados 39
2.8 Considerações finais 40

3 QSEE-TAS: UMA FERRAMENTA PARA EXECUÇÃO AUTOMÁTICA DE CASOS DE TESTE E GERAÇÃO AUTOMÁTICA DA DOCUMENTAÇÃO DO PROCESSO DE TESTE 43
3.1 O projeto QSEE 44
3.2 Arquitetura de teste com a QSEE-TAS 46
3.3 Fluxo de trabalho 47
3.4 Funcionalidades 48
3.4.1 Projeto de teste 51
3.4.2 Itens de teste 53
3.4.3 Casos de teste 53

3.4.4 Atribuição dos passos de teste . . . 54
3.4.5 Especificação do formato das mensagens do protocolo de comunicação . . . 55
3.4.6 Canais de comunicação . . . 57
3.4.7 Ciclos de teste . . . 58
3.4.8 Execução automatizada dos casos de teste . . . 59
3.4.8.1 Modo de seleção simples . . . 59
3.4.8.2 Modo de seleção combinada . . . 61
3.4.9 Importação e exportação de dados . . . 62
3.4.10 Geração dos relatos de teste . . . 63
3.5 A arquitetura da ferramenta . . . 64
3.6 Considerações finais . . . 65

4 SPAC: PROCESSAMENTO E VISUALIZAÇÃO DE DADOS CIENTÍFICOS PARA UMA MISSÃO DE SATÉLITE DE ASTROFÍSICA 67
4.1 Carga de módulos . . . 68
4.2 Extração de dados . . . 69
4.3 Visualização de dados . . . 71
4.4 Integração com placas de aquisição de dados (DAQ) . . . 74
4.5 Considerações finais . . . 75

5 ESTUDO COMPARATIVO DE CUSTO . . . 79
5.1 As configurações de teste usadas no estudo . . . 79
5.2 Metodologia . . . 80
5.3 Avaliação dos resultados . . . 81
5.4 Considerações finais . . . 83

6 CONCLUSÃO . . . 85
6.1 Contribuições do trabalho . . . 85
6.2 Dificuldades encontradas . . . 85
6.3 Limitações e aperfeiçoamentos . . . 86
6.4 Considerações finais e trabalhos futuros . . . 87

REFERÊNCIAS BIBLIOGRÁFICAS . . . 89

A APÊNDICE A - DESENVOLVIMENTO DA QSEE-TAS/SPAC . . . 95
A.1 Hierarquia dos Instrumentos Virtuais (VI's) - Módulos da QSEE-TAS . . . 95
A.2 A Linguagem de Descrição de Mensagens . . . 95
A.3 Arquivo XML para Importação e Exportação . . . 96
A.4 Modelo do Banco de Dados Relacional . . . 99

LISTA DE FIGURAS

Pág.

2.1 Hierarquia dos atributos da qualidade de software. 30
2.2 Um processo de teste de software. 34

3.1 Subsistema de computação de bordo do SWPDC. 45
3.2 Arquitetura de teste com a QSEE-TAS. 47
3.3 Fluxo de trabalho para as atividades de teste com a QSEE-TAS. 48
3.4 Visão conceitual das funcionalidades da ferramenta QSEE-TAS/SPAC. 49
3.5 Interface principal da QSEE-TAS com um projeto de teste carregado. 51
3.6 Propriedades de um projeto de teste. 52
3.7 Mapeamento de um item de teste. 53
3.8 Mapeamento de um caso de teste. 54
3.9 Edição de um passo de teste do tipo Solicitação - Resposta. 55
3.10 Edição de passo de teste do tipo Observação Externa. 56
3.11 Edição da estrutura de mensagens. 57
3.12 Edição de um canal de comunicação do tipo RS-232. 58
3.13 Ciclos de testes: quando e por quem um teste foi aplicado? 58
3.14 Execução de casos de teste em progresso. 60
3.15 Seleção de múltiplos casos de teste para execução combinada. 61
3.16 Uma das telas do assistente de importação e exportação de dados de projetos de teste. 62
3.17 Geração e visualização do relato de teste. 63
3.18 Relato de teste transformado em HTML e visualizado via navegador *web*. . . . 63
3.19 Arquitetura da ferramenta QSEE-TAS/SPAC. 64

4.1 O processo de carga de módulos. 68
4.2 Exemplo do conteúdo do arquivo `modules.conf`. 69
4.3 Recorte de tela enfatizando os módulos SPAC disponíveis. 69
4.4 *Cluster* (registro) que representa um registro de *log*. 70
4.5 Visualização em hexadecimal de uma mensagem de dados científicos codificados em NBE. 72
4.6 Visualizador de dados de *housekeeping*. 73
4.7 Visualizador de dados de adquiridos dos EPPs. 74
4.8 Visualizador de dados na forma de histograma. 75
4.9 Geração de sinais digitais. 76
4.10 Geração de sinais analógicos. 76

5.1 Ambiente de teste para o primeiro cenário. 80

5.2 Ambiente de teste para o segundo cenário. 80
5.3 Resultado da execução dos testes. 82

A.1 Hierarquia dos módulos da QSEE-TAS. 95
A.2 Gramática LDM. 96
A.3 DTD validador do XML para intercâmbio de dados com a QSEE-TAS. 99
A.4 Modelo lógico do banco de dados da QSEE-TAS. 100
A.5 Modelo físico do banco de dados da QSEE-TAS. 101

LISTA DE TABELAS

Pág.

2.1 Termos e conceitos principais relevantes ao trabalho. 32
2.2 Funções atribuídas aos membros de equipes de teste. 33

3.1 Comparação das capacidades desejáveis de um *test harness* e o suporte dado pela QSEE-TAS. 66

5.1 Casos de teste e dados das IST. 81

LISTA DE ABREVIATURAS E SIGLAS

ASN	–	*Abstract Syntax Notation*
BER	–	*Basic Encoding Rules*
BNF	–	*Backus-Naur Form*
CMMI	–	*Capability Maturity Model Integration*
DAQ	–	Placa de Aquisição de Dados
DTD	–	*Document Type Definition*
ECSS	–	*European Cooperation on Space Standardization*
EPP	–	*Event Pre-Processor*
FSoFIST	–	*Ferry-clip with Software Fault Injection Support Tool*
GTSC	–	Geração de Testes com base em StateCharts
GUI	–	*Graphical User Interface*
HEASARC	–	*High Energy Astrophysics Science Archive Research Center*
HXI	–	*Hard X-Ray Imager*
IEEE	–	*Institute of Electrical and Electronic Engineers*
IEC	–	*International Electrotechnical Commission*
INPE	–	Instituto Nacional de Pesquisas Espaciais
ISO	–	*International Standards Organization*
IST	–	Implementação Sob Teste
KeV	–	Quilo Elétron-Volt
LOC	–	*Lines Of Code*
NBO	–	*Network Byte Order*
MEF	–	Máquina de Estados Finitos
MPS.BR	–	Melhoria de Processo de Software Brasileiro
OBDH	–	*On-Board Data Handling Computer*
OSI	–	*Open Systems Interconnection*
QSEE	–	Qualidade do Software Embarcado em Aplicações Espaciais
ROI	–	*Return On Investiment*
SDL	–	*Specification and Definition Language*
SGBD	–	Sistema Gerenciador de Banco de Dados
SPAC	–	Software para Processamento e Análise de Dados Científicos
SQA	–	*Software Quality Assurance*
SWPDC	–	*Software of Payload Data Handling Computer*
TAS	–	Teste Automatizado de Software
TTCN	–	Testing and Test Control Notation
VV&T	–	Verificação, Validação e Teste de Software
XDR	–	*eXternal Data Representation*
XML	–	*eXtensible Markup Language*

1 INTRODUÇÃO

Teste de software é uma atividade essencial na engenharia de software. Segundo (PRESSMAN, 1995), a atividade de teste em projetos de desenvolvimento de software é a mais onerosa. Software embarcado em computadores a bordo de satélites é usualmente complexo, pois atua diretamente no hardware do computador, realiza aquisição de dados de sensores, calcula a correta orientação do satélite no espaço, interage com atuadores do satélite, gerencia os dados obtidos a bordo e é difícil de ser substituído pelo aspecto não tripulado de uma missão de satélite (SILVA et al., 2006). Por causa da natureza autônoma da missão, falhas nesse tipo de software podem provocar a perda de toda a missão se eles não forem revelados durante a fase de testes. Exemplos de relatos de problemas que poderiam ter sido evitados se as falhas do software tivessem sido detectados na fase de teste são encontrados em (AMBROSIO, 2005).

O teste de software embarcado em missões espaciais é uma atividade que consome muito tempo. Dependendo da complexidade do software, muito esforço é empregado em planejamento, configuração do ambiente de teste, execução e emissão de relatos dos testes. Além disso, é desejável (ou até obrigatório) que os relatos de teste estejam em conformidade com normas internacionais como a IEEE-829 (IEEE, 1998) ou ECSS (ESA ECSS, 1998).

O cenário torna-se mais complexo quando é necessário testar software desenvolvido para sistemas críticos. Um satélite científico é decomposto em subsistemas, dentre eles, cita-se como exemplo o subsistema de gestão de bordo denominado *On-Board Data Handling Computer* (OBDH), que é responsável pelo processamento de informação da plataforma e da carga útil do satélite (SANTIAGO et al., 2007). Usualmente um computador da carga útil se comunica com o OBDH para transferir seus dados e, por sua vez, o OBDH se comunica, via subsistema de telecomunicação, com a estação de solo. Portanto, a especificação do teste deve levar em consideração tanto o funcionamento individual de cada subsistema quanto o funcionamento integrado. No exemplo, tanto o OBDH quanto as cargas úteis devem ser testadas no nível unitário (teste de instrumento) e no integrado (teste de subsistema). Finalmente, o teste de sistema simula condições muito próximas das reais em que todos os subsistemas devem ser envolvidos e devem funcionar adequadamente.

Para tentar aumentar a produtividade da atividade de teste de software, algum nível de automação deve ser considerado. Segundo Brian Marick (MARICK, 1998), a automação da execução do teste deve considerar seu custo frente à alternativa manual e o quanto ela está alinhada aos propósitos específicos do software em teste. O objetivo da automação do teste não é eliminar por completo o trabalho manual, nem substituir o valoroso conhecimento dos especialistas no domínio; em vez disso, é um auxílio, um guia (MARICK, 1998).

1.1 Motivação

A automação da execução dos testes encontra justificativa no aumento do retorno sobre o investimento (ROI - *Return On Investiment*). Neste contexto, o ROI é caracterizado pelo aumento da eficiência da execução do teste em relação a sucessivas execuções, quando estas são comparadas com a execução manual. Tom Wissink e Carlos Amaro da divisão de Sistemas Integrados e Soluções da empresa *Lookheed Martin* demonstraram o aumento do ROI pelo emprego de ferramentas automatizadas de execução de testes em sistemas de larga escala e com características similares àquelas encontradas no contexto do software embarcado em plataformas espaciais (WISSINK; AMARO, 2006).

Os trabalhos realizados atualmente na Divisão de Astrofísica (DAS) do INPE incluem projetos de software para controle de cargas úteis para missões em satélites científicos e em balões estratosféricos. Neste contexto, a DAS tem demandado necessidades de testar as funcionalidades tanto do software encomendado a fornecedores externo quanto aqueles elaborados pelos engenheiros do próprio departamento em projetos reais. Além disso, o projeto QSEE (Qualidade do Software Embarcado em Aplicações Espaciais) (SANTIAGO et al., 2007) tem criado um ambiente controlado e propício para experimentação em teste de software embarcado no qual tem-se verificado a importância de execução dos testes de forma automatizada.

O fluxo de dados em missões de Astrofísica é relativamente complexo, pois envolve a manipulação de diversos tipos de pacotes de dados como *housekeeping*, dados de diagnóstico, dados de teste e calibração dos instrumentos. A manipulação desses dados visando extração e visualização em formatos adequados para permitir análises científicas motivou a construção de uma série de módulos para este fim. Fracamente acoplados à QSEE-TAS, os módulos do Software para Processamento e Análises de Dados Científicos (SPAC) (SILVA et al., 2007) permitem, por exemplo, a visualização da distribuição do espectro de energia em raios-X na forma de histogramas, a partir de dados capturados durante os teste da carga útil.

1.2 Objetivo

O objetivo principal desse trabalho foi o desenvolvimento de uma solução para teste funcionais de software embarcados. A ferramenta principal desta solução foi entitulada **Qualidade do Software Embarcado em Aplicações Espaciais - Teste Automatizado de Software** (QSEE-TAS). Sua funcionalidade principal foi centrada, inicialmente, na automatização da execução de testes funcionais e geração da documentação associada ao processo de teste para software embarcado em experimentos de satélites científicos que

adotam o protocolo de comunicação EXP-OBDH, proprietário do INPE, como interface de comunicação com outros computadores a bordo do satélite (SILVA et al., 2006). Uma evolução no desenvolvimento da ferramenta permitiu que ela pudesse ser usada para executar testes e gerar a documentação automaticamente para o software desenvolvido no escopo do projeto QSEE e, atualmente, a ferramenta consegue realizar a execução de testes de software embarcado com protocolos definidos pelo usuário, o que revela sua flexibilidade de uso no processo de teste de outros sistemas.

Os módulos denominados **Software para Processamento e Análises de Dados Científicos** (SPAC) (SILVA et al., 2007), foram desenvolvidos para auxiliar na validação do caráter científico dos instrumentos a bordo de satélites, o que é de fundamental importância, pois a correta manipulação dos dados a bordo é determinante para o sucesso da missão.

Também são apresentados os desafios na sua concepção, a validação da ferramenta, o *feedback* dos colaboradores que a usaram e um estudo empírico sobre uma medida de redução de custo nos testes executados por meio da ferramenta QSEE-TAS/SPAC em comparação a execução dos testes de forma não automatizada. Como objetivos específicos, este trabalho é delimitado pelos seguintes tópicos:

a) estudo da Engenharia de Software no escopo relacionado ao teste de software;

b) estudo do domínio da aplicação em questão, importante para conhecer as necessidades de desenvolvimento de software embarcado em experimentos de Astrofísica de Altas Energias;

c) o desenvolvimento da ferramenta QSEE-TAS/SPAC; e,

d) um estudo de caso no uso da ferramenta de software QSEE-TAS/SPAC, apresentando suas vantagens e desvantagens.

1.3 Estrutura da dissertação

Primeiramente uma visão geral do estado da arte em teste de software no âmbito da Engenharia de Software é apresentada no Capítulo 2. O *Software of Payload Data Computer* (SWPDC), software usado no estudo de caso, é voltado para o domínio da Astrofísica de Altas Energias.

Uma vez conhecendo os aspectos do teste à luz da Engenharia de Software, somadas às necessidades de teste do software no domínio apresentado, o Capítulo 3 apresenta

a ferramenta QSEE-TAS, tanto do ponto de vista do usuário final (o testador) quanto arquitetural. Adicionalmente, são apresentadas as entidades que ela manipula tais como itens de teste e casos de teste, além de outros aspectos técnicos. Esse capítulo descreve os módulos e algumas métricas internas à ferramenta; explica também os limites de uso da ferramenta e até que ponto ela pode ser usada para executar testes de software em domínios similares.

Para o cientista (astrofísicos, no caso), questões como os tipos de quadros que são usados para transmitir cada fóton capturado pelos instrumentos não são interessantes. Ele deseja histogramas, imagens, espectros de luz, que lhe são familiares e têm mais sentido para sua atividade fim. O Capítulo 4 trata este assunto apresentando os módulos SPAC, construídos para aproximar a equipe de teste e cientistas, no sentido de cooperarem na validação do aspecto científico do software embarcado em missões de satélites de astrofísica. Os módulos SPAC acrescentam funcionalidades que ajudam a validar o sistema em teste se valendo, por exemplo, da extração e visualização dos dados a partir dos registros de execução dos testes. Com uma ferramenta capaz de elevar o nível do teste para a percepção do cientista tão logo os testes se iniciem, inconformidades podem ser reveladas mais cedo ajudando a alinhar os aspectos técnicos aos científicos da missão, além de reduzir o esforço de correção das falhas.

Um estudo de caso é apresentado no Capítulo 5. Ele apresenta o quão efetivo foi o uso da QSEE-TAS em comparação ao teste executado manualmente. Os resultados revelam que nem sempre a automação é mais rápida, considerando, por exemplo, fatores como o tempo de aprendizagem do testador, o tempo de configuração do teste na primeira vez, entre outros.

Finalmente, o Capítulo 6 conclui o trabalho descrevendo os objetivos alcançados, as lições aprendidas e as possibilidades de trabalhos complementares no futuros. Detalhes técnicos a respeito do desenvolvimento da ferramenta estão descritos no Apêndice A.

2 TESTE DE SOFTWARE

Teste de software surgiu como disciplina no início dos anos 70, quando, até então, não havia distinção entre depuração e teste. Boris Beizer traz uma enumeração interessante sobre a evolução do pensamento sobre testes em cinco fases (BEIZER, 1990). Beizer começa pela fase zero, onde o teste não tinha importância. Depois, na fase um, o pensamento comum era que o teste deveria provar que o software funciona; na fase dois, isso se inverte, e então o teste deveria mostrar que o software não funciona. Mas, de fato, as fases três e quatro representaram um avanço significativo na concepção do propósito do teste. Na fase três, o teste não deveria provar nada sobre o software, além de reduzir os riscos dele não funcionar quando submetido a entradas de dados inaceitáveis. Finalmente, na fase quatro, o teste é entendido não como um ato, mas uma disciplina mental que deveria levar à produção de software com baixo risco sem muito esforço de teste.

Nos últimos vinte anos, as técnicas em teste de software ganharam avanços significativos por causa do apelo da comunidade por padronização Pressman:1995. Surgiram diversos modelos de melhoria de processos de software como CMMi e MPS.BR que objetivam orientar as instituições a alcançarem excelência na produção de software. Uma parte importante de qualquer processo de software é aquela que visa a garantia da qualidade - *Software Quality Assurance* (SQA), em que é verificada a onerosa atividade de esforço mental que é o teste do produto de software que envolve planejar, executar, analisar e alimentar bases de dados históricas sobre o software em teste. É comum que a atividade de teste empregue de 40 a 50 por cento do esforço despedido em todo o projeto do software, principalmente no caso de software crítico como software para monitoramento de usinas nucleares, equipamentos médicos, controle de vôo e aplicações espaciais (INFOTECH, 1979), (PRESSMAN, 2001).

Este capítulo aborda conceitos introdutórios sobre teste de software, mostra um processo de teste, suas atividades principais e os aspectos relacionados à sua automação.

2.1 Atributos da qualidade de software sob a perspectiva do teste

Segundo Adrion (ADRION et al., 1982), um razoável grau de qualidade do software pode ser alcançado pela aplicação de técnicas de desenvolvimento de software e uso de procedimentos de verificação ao longo de todo seu ciclo de vida. Nesse contexto, o ciclo de vida envolve as etapas de concepção - engenharia do domínio -, análise de requisitos, projeto, construção, teste e aceitação do software. Os procedimentos de verificação, aqui, significam certificar parâmetros de qualidade específicos que podem ser mais qualitativos que quantitativos, pois é difícil se obter uma medida universal que ajude a inferir, por

exemplo, o quanto um software é confiável. Entretanto, medidas quantitativas podem ser obtidas pela aplicação de testes sobre requisitos funcionais e não-funcionais do software, cuja abrangência pode revelar desde problemas de interpretação da especificação até defeitos no código executável. Uma hierarquia de atributos sobre a qualidade de um software é apresentada na Figura 2.1.

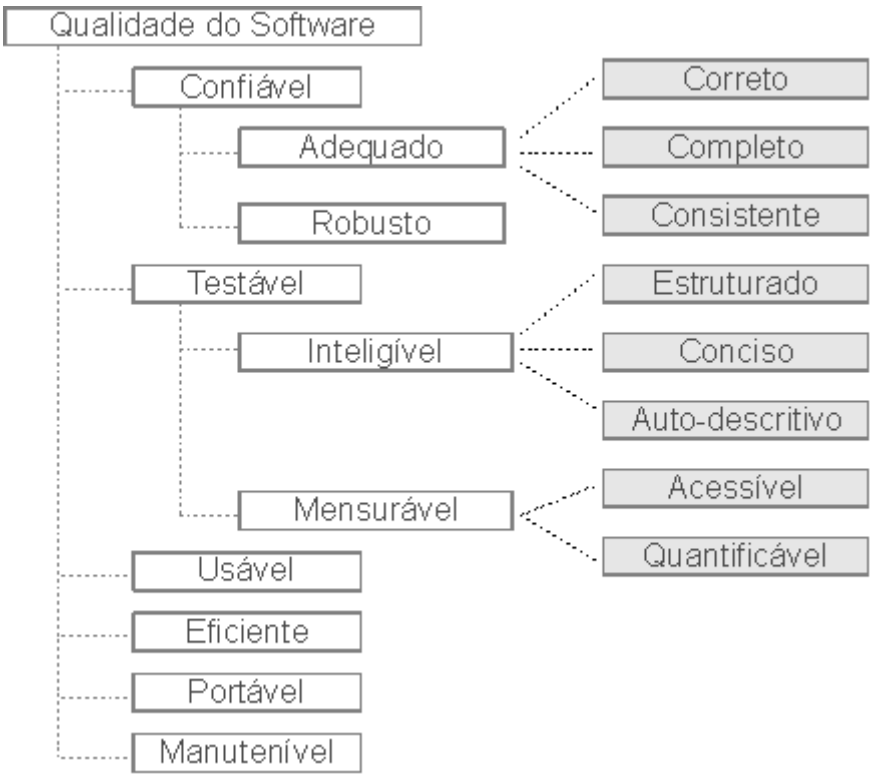


Figura 2.1 - Hierarquia dos atributos da qualidade de software.

Fonte: Adaptado de Adrion et al. (1982, p. 159).

Um software é considerado adequado ("correto") se seus requisitos funcionais foram implementados satisfatoriamente. Entretanto, ele somente é completo do ponto de vista do usuário, se todas as funcionalidades implementadas são suficientes. A robustez de um software é sua capacidade de persistir funcionando adequadamente mesmo na presença de condições não explicitamente especificadas. Por exemplo, se não há mais espaço em disco para efetuar um operação de escrita, ele deve sinalizar tal condição excepcional ao usuário e permanecer funcionando mesmo de forma degradada (HETZEL, 1988).

Se uma seqüência de dados de entrada produzirá uma seqüência de dados de saída (resposta) bem definida numa dada condição, é esperado que o software comporte-se como tal, sem produzir respostas inesperadas, quando a mesma seqüência de dados for introduzida.

Fatores como confiabilidade e testabilidade estão fortemente relacionados ao software crítico a bordo de plataformas espaciais. Para garantir a qualidade do software, o processo deve contemplar, além das atividades de gestão administrativa, atividades de Verificação, Validação e Teste do software (VV&T).

A seguir, nesse capítulo, são mostrados os conceitos básicos sobre Teste de Software. Além disso, descreve-se as atividades de teste ao longo do processo de desenvolvimento de software, identificando oportunidades para automação relacionadas com a geração de casos de teste, execução de casos de teste e análise de resultados.

2.2 Conceitos básicos

Segundo Myers (MYERS, 2004), o objetivo da atividade de teste é revelar falhas no produto de software. Entretanto, é altamente dispendioso submeter o software a todo o conjunto de dados de seu domínio. Assim, é importante elaborar casos de teste que tenham alta probabilidade de revelar falha. Segundo (BEIZER, 1990), um caso de teste é um conjunto de entradas, condições de execução, e resultados experados desenvolvidos com um objetivo particular. Esse objetivo particular pode ser tanto para demostrar que o software funciona como não funciona diante quando submetido a certos cenários de operação. Entretanto, para (MYERS, 2004), bons casos de teste são aqueles que revelam falhas. Para tentar produzir tais casos de teste, a atividade de teste é subdivida em quatro etapas: planejamento, projeto de casos de teste, execução dos casos de teste e avaliação dos resultados (MYERS, 2004). O planejamento envolve a concepção dos itens de teste, ambiente de teste e procedimentos operacionais para a viabilização do teste da Implementação Sob Teste (IST), ou seja, o software sob teste. A etapa de projeto dos casos de teste considera os cenários nos quais a IST será condicionada para executar frente a pré-condições, dados e estímos de entrada, dados de saídas e comportamentos esperados. A execução dos casos de teste toma como base os artefatos do projeto do teste e observa o comportamento da IST à medida que estímulos são enviados a ela. Todos os dados coletados a respeito da interação com a IST são, então, passados para a etapa de análise, quando é feito o julgamento do comportamento da IST que atribui um veredito ao teste. Esse julgamento é geralmente auxiliado por um oráculo de teste - que pode ser humano ou um software - cuja função é inferir se a IST passou ou não no teste.

Segundo Howden (HOWDEN, 1987), o teste de software pode classificado, dentre outras formas, em teste baseado no programa (*program-based testing*) ou teste estrutural e teste baseado na especificação (*specification-based testing*) ou teste funcional. No teste baseado no programa a estrutura interna do programa (código fonte) é o alvo da execução dos casos de testes. Este tipo de teste também é conhecido como teste caixa-branca (ou caixa

de vidro - *glass-box testing*) e tem como vantagem o fato de exercitar os caminhos possíveis no código do programa. Entretanto, ele é dependente da linguagem de programação. Ao contrário, o teste baseado na especificação, também conhecido como teste caixa-preta, não considera as estruturas internas do programa. Em vez disso, os casos de teste são elaborados com base na especificação do comportamento do software. Uma vantagem é a independência de linguagem de programação - independente de paradigma (procedimental, orientado a objetos, etc.) - ao custo de um esforço maior na concepção dos casos de teste, dado que os documentos que especificam o comportamento do software são geralmente escritos em linguagem natural (informal), o que pode incorrer em ambigüidades e interpretações incorretas. Contudo, esses tipos de testes são complementares. Em geral, existem diversos tipos de teste (abordagens) ao teste de software como, por exemplo, o teste baseado em modelos, teste de mutação, testes orientado a objetos e de componentes, entre outros. Uma ampla introdução sobre o teste de software considerando-se diferentes técnicas pode ser encontrada em (DELAMARO et al., 2007).

Para uniformização de nomenclatura e entendimento deste trabalho, a Tabela 2.1 apresenta um lista com alguns termos e conceitos associados a testes de software.

Tabela 2.1 - Termos e conceitos principais relevantes ao trabalho.

Em Inglês	**Em Português**	**Universo**	**Descrição**
Mistake	Erro humano	-	Uma ação humana que produz um resultado incorreto.
Fault	Defeito	Físico	Incorreção em passo, processo ou definição de dados. É a manifestação no software de um engano cometido pelo desenvolvedor. Um *bug*.
Error	Erro	Informação	Diferença entre o valor obtido e o valor esperado. Exemplo: A distância calculada deveria ser 30 metros (esperado) em vez de 30,1 metros (obtido). Diferença (erro) de 0,1 metro
Failure	Falha	Usuário	Incapacidade de fornecer o serviço conforme especificado.
Test Harness	Ferramental de teste	-	Compêndio de ferramentas de software que habilita a execução automatizada dos testes.

Fonte: Adaptado de IEEE (1990).

Os tópicos a seguir introduzem as atividades de geração de casos de teste, execução e análise de resultados do teste, que fazem parte de um processo de Verificação, Validação e Teste (VV&T).

2.3 Verificação, validação e teste de software - VV&T

No contexto deste trabalho, os termos **verificação** e **validação** têm o mesmo significado apontado por Storey (STOREY, 1996). Verificação é o conjunto de atividades com o objetivo de checar se o software implementa corretamente uma função específica. A validação é entendida como o processo para determinar se o software atende seu propósito de funcionamento em termos de requisitos do usuário[1].

Em aplicações críticas, como no caso de satélites, a validação acompanha o teste do software nas diferentes configurações de hardware como, por exemplo, os modelos de engenharia, de qualificação e de vôo (ESA ECSS, 1998), durante a evolução do projeto dos computadores da plataforma e da carga útil do satélite.

Um processo de VV&T geralmente atribui papéis (funções) ao pessoal da equipe de teste. Verifica-se na literatura (PRESSMAN, 2001) (BEIZER, 1990) os papéis de Gerente de Teste, Líder da Equipe de Teste, Arquiteto de Teste e Testador. A Tabela 2.2 apresenta uma compilação dos principais perfis e atividades atribuídos aos recursos humanos envolvidos com teste de software.

Tabela 2.2 - Funções atribuídas aos membros de equipes de teste.

Perfil	**Descrição e atividades relacionadas**
Gerente de Testes	Coordena o processo de teste. Esta atividade envolve a preparação da documentação formal do processo de teste com uma visão ampla dos sistemas a serrem testados considerando riscos, estimativas de custo e prazo, aderência a padrões internacionais e da empresa e elaboração de plano de teste.
Arquiteto de Testes	Viabiliza o teste do ponto de vista tecnológico e ambiental, ferramentas, metodologias a serem empregadas e elaboração dos casos de teste.
Testador	Responsável por exercitar os sistemas em teste aplicando os casos de teste acordo com o plano de testes; considera os procedimentos para execução, análise dos resultados dos testes e atribui veredi-tos.

[1]Segundo (BOEHM, 1981), "Verificação: Estamos construindo certo o produto?" e "Validação: Estamos construindo o produto certo?"

2.4 Um processo de teste e suas principais atividades

O objetivo do processo de teste é guiar o fluxo de atividades de teste e a produção de artefatos para avaliar a qualidade do software desenvolvido (ESA ECSS, 1998). Um processo de teste pode ser dividido em cinco etapas: planejamento, geração de casos de teste, execução dos casos de teste, análise dos resultados e geração relatos de teste.

A Figura 2.2 ilustra este processo que se inicia com base na especificação do software. Neste contexto o software é a IST.

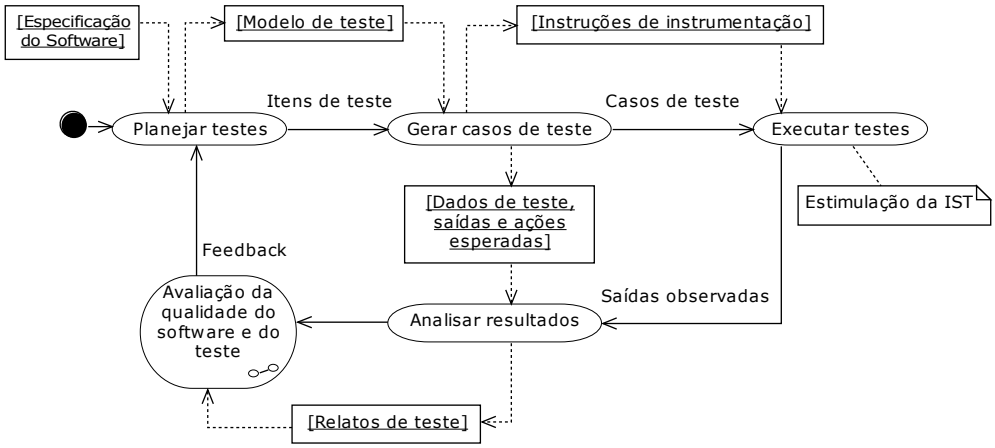


Figura 2.2 - Um processo de teste de software.

Com o objetivo de gerar casos de teste que tenham alta probabilidade de revelar falhas, o arquiteto de testes pode fazer uso de modelos que representem o comportamento esperado do software frente a um conjunto de possíveis dados de entrada. Isso pode ser feito tanto manualmente - por meio do preenchimento de planilhas, por exemplo - quanto feito por meio de ferramentas automatizadas que integram a modelagem à geração dos casos de teste.

Em seguida os casos de teste são convertidos para uma representação de máquina viabilizando sua execução. Adicionalmente, são geradas tabelas com regras para inferir se um dado caso de teste executou e revelou ou não algum comportamento indesejado ou não especificado. Durante a execução dos casos de teste os resultados são coletados e remetidos à análise para atribuição do veredito. Nesta etapa, os resultados observados são comparados com saídas e/ou comportamentos esperados. É considerado que a IST passou no caso de teste se toda saída observada está de acordo com a saída esperada correspondente; caso contrário, a IST não passou no caso teste.

À medida que os casos de teste são executados e avaliados, todo o histórico da execução é registrado. O conjunto desses registros com os vereditos atribuídos a cada caso de teste é o relato de teste.

Por fim, análises subseqüentes podem ser feitas para avaliar a efetividade do próprio processo de teste, cujos resultados servem como parâmetro (feedback) para a elaboração de casos de teste melhores. O processo termina quando nenhuma falha é encontrada na IST. Entretanto, podem ser exigidos diversos ciclos no processo de teste dependendo de fatores como a maturidade dos requisitos, qualidade do projeto de software da IST e a quantidade de recursos disponíveis (prazo, dinheiro, pessoal).

2.4.1 Geração de casos de teste

A geração de casos de testes emprega diversas técnicas para produzir casos de testes que vão desde a concepção manual até a geração baseada em modelos formais da especificação do software. O objetivo desta etapa é produzir casos de teste com boa probabilidade de revelar falha. Para este fim, um modelo de falhas (STOREY, 1996) pode ser empregado para produzir casos de teste mais efetivos. Isso é muito importante porque, do ponto de vista prático, é inviável submeter à IST todas as entradas possíveis do seu domínio (AMBROSIO et al., 2002).

Um caso de teste especifica um cenário especfífico no qual um ou mais itens de teste devem ser exitados durante a execução do teste. Os itens de teste, por sua vez, estão ligados aos requisitos (funcionais e não funcionais) do software. Um caso de teste representa uma possibilidade de uso do software, por mais estranho que este uso possa ser do ponto de vista do usuário e podem ser classificados como: usos normais, usos excepcionais e tolerância a faltas (ESA ECSS, 1998).

Os casos de testes que exercitam os usos normais do software são aqueles que materializam as condições necessárias para o exercício de uma funcionalidade especificada do software. Seu objetivo é verificar se o software realiza certa funcionalidade de acordo com o fluxo básico de operação especificado em sua documentação.

Ao contrário dos usos normais, os casos de teste dos usos excepcionais têm por meta verificar se o comportamento do software é satisfatório quando este é submetido a entradas excepcionais especificadas na documentação.

Os casos de teste de tolerância a falhas tentam revelar o quão recuperável é o software quando este é levado a condições de falhas. Nesse caso, geralmente são empregadas técnicas de injeção de falha, tanto por software quanto por hardware, empregando-se algum tipo

de instrumentação especial para introduzir perturbações adequadamente.

Atualmente existem inúmeras ferramentas de software que geram casos de testes; algumas com suporte comercial e outras ainda embrionárias em projetos de código aberto. Um exemplo de ferramenta que gera casos de teste com base na especificação formal do comportamento do software por meio de Máquina de Estados Finitos (MEF) é a ConDado, como posto em (MARTINS et al., 1999) e (AMBROSIO et al., 2005). Por meio dela, os conjutos de testes são extraídos percorrendo os estados e transições no modelo gerando seqüências de teste que dependem do critério de teste previamente selecionado. A ferramenta Telelogic TTCN Suite ™é um exemplo de ferramenta comercial que gera testes para protocolos de comunicação. A especificação dos testes é descrita na linguagem *Testing and Test Control Notation version 3* (TTCN-3), extraída da descrição formal do protocolo em *Specification and Definition Language* (SDL). Uma introdução interessante dessa ferramenta pode ser encontrada em (TELELOGIC, 2007) que relata casos de sucesso sobre testes de implementações do protocolo *Blue Tooth* (IEEE 802.15.1), usando SDL e TTCN-3. Outras ferramentas aceitam modelos em *Statecharts* como entrada e produzem seqüências de teste abstratas aplicando por meio da aplicação de critérios de testes (todas transições, ou todas as configurações, etc.). Como exemplo, citam-se as comerciais Reatic (REACTIVE-SYSTEMS, 2008) e Qtronic (CONFORMIQ-SOFTWARE-INC, 2008), e as acadêmicas GTSC (SANTIAGO et al., 2008) e a WebPerformCharts (ARANTES et al., 2008).

2.4.2 Execução de casos de teste

Executar casos de teste requer interagir com a IST estimulando-a com os dados de entrada e capturando as saídas. A execução dos casos de testes transforma os procedimentos de teste especificados nos casos de teste numa representação executável, seja na forma de instruções não automáticas - como observar um sinal ou clicar num botão na interface gráfica -, seja na forma de instruções executáveis que estimulam a IST com os dados de entrada automaticamente.

A automação da execução dos casos de teste é importante para reduzir a intervenção humana no processo de teste, já que a quantidade de casos de teste tende a ser volumosa; a execução manual, além de repetitiva, é propensa a erros. Uma técnica comum é o uso de programas que convertem a representação simbólica dos procedimentos de teste em instruções executáveis, formando *scripts* de teste capazes de estimular a IST e coletar as respostas aos estímulos automaticamente. Na execução automatizada de testes, o objetivo é transformar as especificações dos procedimentos de testes em interações lógicas com a IST. Para este propósito, existem linguagens específicas para representar tais interações (GIESSLER; BAUMGARTEN, 1994). O foco dessas linguagens é expressar a sintaxe e a

semântica das instruções de teste.

2.4.3 Análise dos resultados de teste

O objetivo da análise dos resultados de teste é verificar se a IST passou ou não passou à execução de um caso de teste. A execução do teste geralmente leva a coleta de dados potencialmente volumosos. Para que um veredito seja dado ao caso de teste, todo ou grande parte do volume de dados que rastreia a execução do teste deve ser considerado. Dependendo da granularidade do caso de teste, diversas técnicas podem ser empregadas para analisar os resultados. Entre tais técnicas, pode-se citar desde a simples verificação por expressões regulares nas respostas observadas até técnicas sofisticadas de mineração de dados (LAST et al., 2003) e análise temporal do comportamento da IST função dos estímulos aplicados com base em relógios relativos (PROBERT et al., 1992).

Além de ser essencial para atribuição de veredito aos casos de teste, a análise dos resultados do teste serve também para avaliar a qualidade do próprio caso de teste. A atribuição automática do veredito pode ser assistida por um **Oráculo de Teste** (*Test Oracle*). Segundo (HOWDEN, 1987), um oráculo de teste é um mecanismo (especificação de programa, tabela de exemplos), ou simplesmente o conhecimento do testador no domínio do sistema em teste. É por meio dele que se decide se o que está sendo testado passa ou não passa no teste. Um oráculo de teste perfeito deveria se comportar como uma implementação confiável da IST. Verifica-se que há diversos pontos chave a considerar no projeto de um oráculo. Abaixo, uma enumeração não exaustiva desses pontos chave (BINDER, 2000):

- Como obter o resultado esperado e o resultado real;
- A disponibilidade do resultado esperado: antes, durante ou depois da execução do teste;
- O armazenamento dos resultados reais e esperados para dar suporte aos testes de regressão;
- O tipo avaliação: comparação direta ou indireta;
- O escopo do teste;
- O grau de dependência da plataforma (hardware, sistema operacional);
- Os tipos de saída observáveis: figuras de *bitmap*, elementos de tela (*widgets*), cadeias de caracteres, arquivos, bancos de dados, pacotes de rede, fluxo de bits dependente de dispositivos, saídas analógicas, movimentos mecânicos, entre outras reações físicas;

- Formatos de codificação das saídas observáveis;
- Freqüência de aquisição das saídas observáveis;

A automação da análise dos resultados dos testes vai depender do formalismo e do grau de automação empregado para executar os testes. Isso leva ao problema do **oráculo de teste** (*test oracle*) que, como posto por Beizer (BEIZER, 1990), diz que a automação do teste depende da habilidade de detectar, via programação, quando a IST falha no teste (ou quando o teste é bem sucedido), o que restringe a capacidade de automação. A seguinte questão deve ser levantada: se há um programa de computador capaz de dizer se IST passou ou não no teste, então, por que não jogar fora a IST e colocar este programa em seu lugar?

2.5 Teste de regressão

O teste de regressão (ou re-teste) se faz necessário quando há variação de versão da IST quando, por exemplo, há a correção de um defeito em relação à versão anterior (SOMMERVILLE, 2003). Assim, deve-se avaliar se a correção do defeito foi bem sucedida e não provocou defeitos em funcionalidades existentes que passaram por testes anteriores. O teste de regressão é uma atividade cara, pois, à medida que o software evolui, inúmeros novos casos de teste devem ser executados e aqueles obsoletos, arquivados. Portanto, o sucesso do teste de regressão dependerá da qualidade das técnicas de priorização de casos de teste.

Diversas pesquisas para melhorar a efetividade do teste de regressão têm sido conduzidas. Alguns exemplos incluem os trabalhos de Elbaum e sua equipe (ELBAUM; ROTHERMEL, 2001), sobre a prioziação dos casos de teste com base em custo de falha; o de Park (PARK et al., 2008), que usa dados históricos da execução dos testes com base no custo do teste; e, os de (SRIKANTH; WILLIAMS, 2005), que faz a priorização com base nos requisitos funcionais.

2.6 Esquemas de representação e codificação de mensagens

Alguma forma de representação de estímulos e respostas da IST deve ser levada em consideração pela ferramenta de teste. Uma linguagem de representação dos estímulos pode ser expressa por meio de gramáticas formais como, por exemplo, a BNF[2]. Assim, técnicas de compilação podem ser empregadas para transformar tal representação em linguagem de máquina intermediária ou executável, gerando chamadas como `send()` e `receive()` que estimulam a IST e coletam suas respostas, respectivamente. No caso específico das mensagens definidas no protocolo de comunicação, isto pode ser entendido como um esquema

[2]Backus-Naur Form, usada para especificar gramáticas livres de contexto.

de processamento de Unidades de Dados de Protocolo (UDPs) (HOLZMANN, 1991).

Segundo a especificação em camadas do modelo OSI, a *Abstract Syntax Notation* (ASN.1) (ISO, 2002) pode ser usada para especificar a estrutura das mensagens trocadas entre entidades de uma mesma camada N (fim a fim)[3]. Uma vantagem desta abordagem é o desacoplamento da representação da mensagem do seu formato de transmissão que é, por sua vez, processado pela camada imediatamente inferior (N - 1). Diversos esquemas de codificação são encontrados na literatura. Citam-se como exemplos a (*Basic Encoding Rules - BER*), a *Network Byte Order* (NBO), a *LightWeight Encoding Rules* (LWER), a *eXternal Data Representation* (XDR) e a Codificação Baseada em XML (XER). Um esquema de codificação é empregado para converter a representação abstrata da mensagem na seqüência de bytes que é transmitida para IST. Relatos de experiências interessante a respeito do emprego da ASN.1 com diversos tipos de codificação, tanto em binário quanto em XML, são encontradas em (TANTIPRASUT et al., 1997) e (IMAMURA; MARUYAMA, 2001). Um compilador de ASN.1 para XML relativamente popular, de código aberto e gratuito pode ser encontrado em (WALKIN, 2007).

2.7 Trabalhos relacionados

Nesta seção alguns trabalhos relacionados sobre execução de teste e arquiteturas de teste são apresentados. Chanson e sua equipe (CHANSON et al., 1990) propuseram uma arquitetura Ferry-Clip com entidades ativas (*Active-Ferry*) e passivas (*Passive-Ferry*) nos canais de *ferry*. Em (MARTINS; MATTIELLO-FRANCISCO, 2003) as autoras propuseram uma extensão à esta arquitetura - ferryinjection - adicionando mecanismos para injeção de falha para abordar a valiação de sistemas críticos como aplicações espaciais.

Takahashi e seu grupo, em (TAKAHASHI et al., 1993) desenvolveram uma arquitetura de teste de interoperabilidade, o Método de Teste de Interconectividade Coordenada e Distribuída (DCITM), e um sistema de teste de conformidade baseado nessa arquitetura. No DCITM, UTs e ISTs são colocados no mesmo sistema e o UT opera o PCO de forma autônoma. Isto difere um pouco da arquitetura *Ferry-Clip* em que UTs enviam instruções aos *Passive-Ferries* para que estes, por sua vez, operem os PCOs de camadas superiores (acima das ISTs). Como resultado, o DCITM requer menos dados para enviar e receber na coordenação dos testes se comparada à abordagem Ferry-Clip.

Lima e Cavalli (LIMA; CAVALLI, 1997) propuseram uma arquitetura de teste geral baseada em CORBA para serviços de telecomunicações com foco na execução de testes em sistemas distribuidos. Tal arquitetura possui dos conjuntos de componentes: componente testador

[3] A ASN.1 é descrita em BNF como pode ser observado na norma (ISO, 2002).

e componente sub teste que estão (possivelmente) distribuídos em diferentes lugares. O componente testador recebe informações sobre a configuração do objeto e sobre a suite de teste que é obtida por meio de algum método método formal de geração de casos de teste com base especificações formais.

Wissink e Amaro (WISSINK; AMARO, 2006), da Lockheed Martin Corporation demostram o aumento de Retorno Sobre Investimento (ROI) por meio do uso de ferramentas para execução automática de testes em sistemas de larga escala.

Tornar a ferramenta de teste suficientemente genérica para viabilizar sua comunicação com uma IST que implementa um protocolo específico representa um desafio técnico longe de ser trivial. Nota-se que, como exposto na Seção 2.6, esquemas de representação e codificação são importantes. Mas isto ataca apenas uma parte do problema. A outra parte é a representação do comportamento, da dinâmica do protocolo e sua relação com as camadas de protocolo de níveis mais baixos.

A execução automatizada do teste, que a princípio pode parecer simples, requer a implementação de uma ou mais camadas de protocolos, chamadas de implementação de referência. No contexto do modelo de referência OSI (IS-9646), verifica-se na literatura a arquitetura *Ferry Clip*, que inspirou diversos trabalhos como em (DAVIS et al., 1991) - que descreve uma arquitetura portável para teste de protocolos -, e em (AMBROSIO et al., 2004) - que mostra experiências na adaptação da arquitetura *Ferry Clip* em aplicações espaciais -, e (MARTINS; MATTIELLO-FRANCISCO, 2003) que aborda aspectos de injeção de falhas por software na *Ferry-clip with Software Fault Injection Support Tool* (FSoFIST). Outros desafios incluem a geração automática de dados de teste, que tenta escolher dados que, além de ser uma boa representação do domínio de dados em questão, também deve ter boa probabiliade de revelar falhas (DELAMARO et al., 2007, p. 269).

Nos trabalhos pesquisados, qualquer que seja a arquitetura de teste, verifica-se que a execução automatizada é um desafio técnico multidisciplinar que envolve aspectos práticos não triviais como, por exemplo, sincronização de processos, compilação (tradução) e gerenciamento de filas de mensagens, e, possivelmente, lidar com múltiplos protocolos de comunicação em diversos níveis de detalhe.

2.8 Considerações finais

Este capítulo apresentou conceitos sobre teste de software pertinente ao escopo deste trabalho. Apresentou, ainda, trabalhos relacionados que ajudaram na elicitação dos requisitos de software para a construção da ferramenta. Além da definição dos conceitos, foram apresentos os desafios técnicos que a construção de uma ferramenta para teste

automatizado de sistemas críticos embarcados tem que lidar. O Capítulo 3, a seguir, apresenta a QSEE-TAS e o contexto no qual ela foi concebida; apresenta sua arquitetura e principais funcionalidades, mostrando como foram resolvidos muitos dos desafios técnicos envolvidos.

3 QSEE-TAS: UMA FERRAMENTA PARA EXECUÇÃO AUTOMÁTICA DE CASOS DE TESTE E GERAÇÃO AUTOMÁTICA DA DOCUMENTAÇÃO DO PROCESSO DE TESTE

A ferramenta QSEE-TAS foi desenvolvida para auxiliar na execução dos testes funcionais do software no contexto do projeto Qualidade do Software Embarcado em Aplicações Espaciais (QSEE) (SANTIAGO et al., 2007) e tem desempenhado um papel importante no processo de teste funcional da IST principal - o SWPDC (*Software for the Payload Data Handling Computer*) -, além dos simuladores que completam o subsistema de computação de bordo adotado no escopo deste projeto. A QSEE-TAS ajuda no mapeamento dos casos de teste para uma forma executável capaz de interagir com as ISTs por meio de seus diversos protocolos de comunicação, permitindo que o testador especifique detalhes inerentes à execução dos testes de aplicações embarcadas como o tipo de canal (via) de comunicação com a IST (i.e. RS-232, TCP/IP, Porta paralela), o formato das mensagens do protocolo e temporização. Em resumo, se um software embarcado usa algum protocolo do tipo *send/receive* cujas mensagens podem ser transmitidas e recebidas sobre os canais de comunicação oferecidos, então, ele pode ser testado usando a QSEE-TAS.

Uma vez que detalhes específicos para execução dos testes são configurados, o testador pode se concentrar no cadastramento dos casos de teste que exercitarão a IST em função dos itens a serem testados de acordo com o plano de teste. Atualmente a QSEE-TAS não dá suporte direto para geração automática de casos de teste sendo necessária à inclusão manual dos mesmos. Entretanto, os casos de testes já incluídos podem participar de vários ciclos de teste. O grau de reuso desses casos de teste vai depender da forma em que foram concebidos. Todavia, verificou-se economia significativa de tempo, principalmente nos testes de regressão. O Capítulo 5 dá detalhes do estudo de caso em que tal economia foi verificada.

Este capítulo descreve as principais características da ferramenta do ponto de vista do usuário (testador), exemplificando a inclusão de casos de teste, execução e emissão de relatórios de testes (ou relatos de teste). A organização deste capítulo está disposta da seguinte forma: a próxima seção descreve resumidamente o projeto QSEE, que gerou as condições necessárias para experimentações em engenharia de software para plataformas espaciais, delimitando o contexto deste trabalho. A Seção 3.2 descreve uma proposta de arquitetura de teste com multíplas ISTs. A Seção 3.3 descreve o fluxo de trabalho básico proposto para uso da ferramenta QSEE-TAS, ilustrando os passos necessários para execução dos testes. A Seção 3.4 descreve o que a ferramenta é capaz de fazer com foco nas principais entidades que ela gerencia, tais como itens de teste, casos de teste, comunicação com a IST, entre outros. A Seção 3.5 descreve os aspectos arquiteturais da ferramenta,

ilustrando suas partes principais. Ao final, a Seção 3.6 descreve resumidamente o que é exposto neste capítulo e apresenta algumas considerações finais. Detalhes técnicos a respeito do desenvolvimento da QSEE-TAS estão descritos no Apêndice A

3.1 O projeto QSEE

O contexto delimitado pelo projeto Qualidade de Software Embarcado em Aplicações Espaciais (QSEE) fez sugir as condições que levaram ao projeto e construção da ferramenta QSEE-TAS. Resumidamente, o projeto QSEE criou um ambiente de experimentações interessantes em diversas áreas da engenharia de software, produzindo diversos produtos e lições aprendidas. Fomentado pela Financiadora de Estudos e Projetos (FINEP) via Ação Transversal - Software 06/2004 e em parceria com a empresa DBA Engenharia de Sistemas LTDA, Instituto de Computação da UNICAMP, os objetivos do projeto QSEE incluem (INPE.CEA, 2007):

- Transferir para a indústria brasileira do setor de software os conhecimentos adquiridos no INPE com o desenvolvimento de software para a área espacial, em particular os ambientes e técnicas de validação e verificação utilizados na integração dos softwares embarcados em cargas úteis de satélites científicos e balões estratosféricos;

- Atualizar a atual metodologia de desenvolvimento de software para cargas úteis de satélites científicos e balões estratosféricos que estão sendo desenvolvidos na área Ciências Espaciais e Atmosféricas (CEA) do INPE;

- O desenvolvimento e validação de uma metodologia para que o INPE possa aceitar software embarcado fornecido pelo setor privado, o qual se fundamenta em revisões técnicas formais entre cliente e fornecedor, testes independentes do produto em desenvolvimento baseados nas normas da European Cooperation for Space Standardization (ECSS).

O projeto QSEE pode ser considerado a primeira iniciativa no intuito de habilitar uma empresa brasileira de desenvolvimento de software a desenvolver software para computadores de bordo de cargas úteis de satélites (SANTIAGO et al., 2007).

O instrumento em questão é o *Payload Data Computer* (PDC), com seu software embarcado (SWPDC). Os requisitos de hardware e software para o SWPDC foram formulados pelo INPE. Duas equipes de profissionais de senioridade plena em computação foram treinadas no INPE. Os requisitos foram entregues para que as duas equipes desenvolvessem

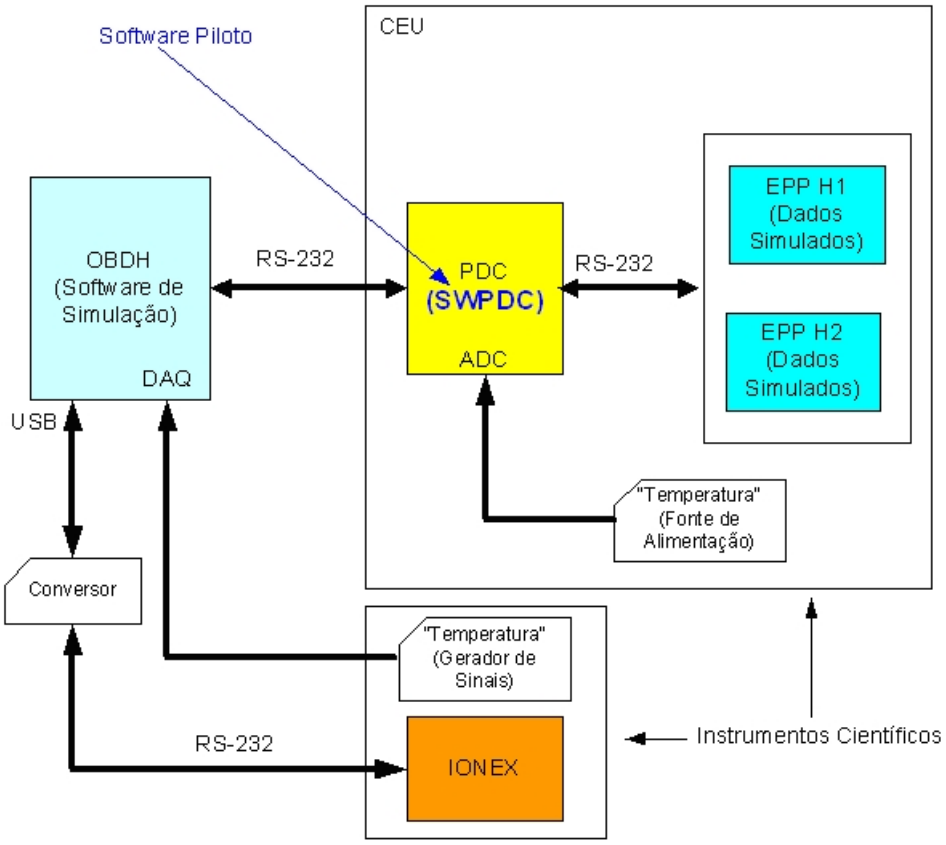


Figura 3.1 - Subsistema de computação de bordo do SWPDC.

Fonte: Adaptado de INPE.CEA (2007).

o software. A equipe DBA construiria sua implementação cumprindo os tramites das atividades de uma Fábrica de Sofware (FSW) aderente ao modelo de maturidade *Capability Maturity Model Integration* (CMMI) nível 3. A equipe INPE faria uso de metodologia de desenvolvimento de software embarcado para computadores de instrumentos científicos estabelecida na instituição, com base em normas ECSS, para desenvolver sua implementação, sem o apoio de processos de FSW.

Na concepção do projeto QSEE dois fornecedores com processos e metodologias diferentes, desenvolveram softwares de acordo com o mesmo conjunto de especificações fornecido pelo cliente. Isso tem demonstrado ser extremamente valioso no intuito de perceber o esforço que uma empresa de software deve realizar para entrar no domínio espacial assim como ver os pontos positivos e negativos dos processos de desenvolvimento de software de ambas as instituições para que tais processos possam sofrer melhorias (SANTIAGO et al., 2007).

O subsistema de computação que contextualiza o SWPDC está ilustrado na Figura 3.1. A arquitetura do Subsistema de Computação para o projeto QSEE. Esta arquitetura é

um *downsizing* da arquitetura do satélite MIRAX (BRAGA et al., 2004) real.

Em resumo, o SWPDC é um software que gerencia a aquisição de dados de diagnóstico, dados de teste, e dados científicos, a partir das câmeras EPP-HXI-1 e EPP-HXI-2 (EPP H1 e H2, na Figura 3.1) dados de *housekeeping* e *dump* de memória, provenientes do próprio PDC. Além disso, o SWPDC possui funcionalidades como carregamento de novos programas (*firmware update*), modos de operação (segurança, nominal, e diagnóstico), ligamento e desligamento das câmeras HXI[1] e coletas periódicas de temperatura em dois sensores.

Portanto, para auxilar o processo de aceitação do software, uma ferramenta que permitisse a execução e relatos automatizados de testes de software embarcado deveria ser construida para fazer o papel do OBDH, ilustrado na Figura 3.1. Além disso, ela deveria ser concebida para que os testes pudessem ser efetuados tanto no nível de instrumento (SWPDC) quanto no nível de subsistema (SWPDC e outras cargas úteis). Em resposta a essas necessidades, a ferramenta QSEE-TAS foi concebida. As próximas seções tratam o fluxo de trabalho introduzido pela ferramenta e suas funcionalidades.

3.2 Arquitetura de teste com a QSEE-TAS

A arquitetura de teste com a QSEE-TAS é apresentada na Figura 3.2. Nesta abordagem, o Sistema Sob Teste (SST) engloba a IST e quaisquer outros equipamentos necessários ao teste como Simuladores do Ambiente de Teste (SAT). A quantidade de ISTs que podem ser envolvidas numa mesma execução de testes dependerá somente da quantidade de interfaces de comunicação (RS-232, USB, TCP/IP, etc.) disponíveis no computador *host* em que a QSEE-TAS está instalada (SANTIAGO et al., 2008).

Os Pontos de Observação e Controle (PCOs) são os meios pelos quais pode-se observar o comportamento das ISTs. Para cada IST, a QSEE-TAS tem acesso a um PCO associado ao Canal Sob Teste (CST). O CST é o canal de comunicação de dados entre a QSEE-TAS e a IST. Há ainda o canal especial do Mecanismo de Injeção de Falha (MEIF) que provê comunicação com os SATs com o objetivo de influência-los a alterar seu comportamento para simular condições de falha. Neste caso específico, o canal MEIF é uma conexão TCP/IP sobre uma rede local que leva instruções especiais para que cada SAT perturbe a IST de um modo específico. Esta característica é importante porque o software embarcado em aplicações críticas deve ser submetido a avaliações de fidedignidade (*Dependability*) como tolerância a falhas.

Geralmente, sistemas embarcados fazem uso de canais analógicos e digitais para receber

[1] *Hard X-Ray Imager* - imageadoras de raios-X duros

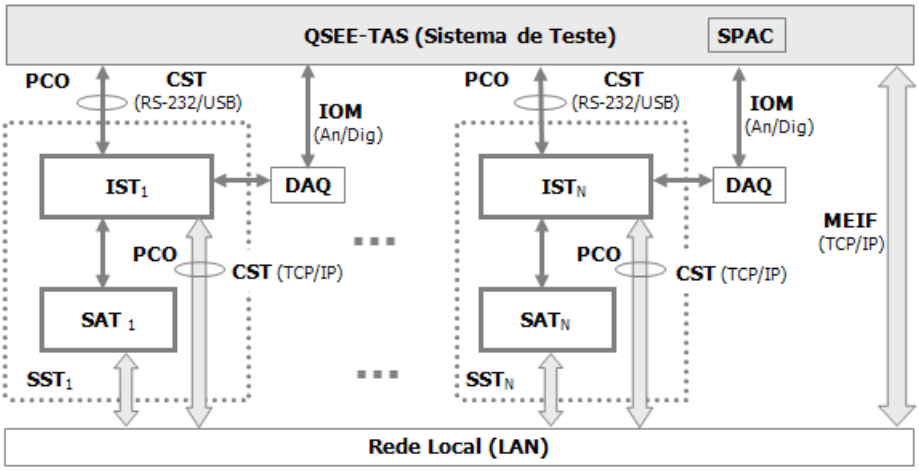


Figura 3.2 - Arquitetura de teste com a QSEE-TAS.

informações de sensores e enviar informações a atuadores, por exemplo. Para tratar esse tipo de comunicação, representada na Figura 3.2 pelas Placas de Acquisição de Dados (DAQs), a QSEE-TAS possui um módulo externo chamado Processamento e Análise de Dados Científicos (SPAC) - detalhado no Capítulo 4 - que trada as portas de E/S analógicas e digitais (IOM).

A QSEE-TAS não foi concebida para testar protocolos de comunicação em implementações multi-camada como aqueles que seguem o modelo de referência OSI. Portanto, não faz sentido aqui o conceito de *Upper Tester* (UT), que controla e observa o PCO da camada N+1 e o *Lower Test* (LT), que controle e observa o PCO da camada N-1 (CHANSON et al., 1990). Também é importante mensionar que os elementos SAT e DAQ além do MEIF e os canais IOM são opcionais na arquitetura proposta. O uso desses elementos depende da IST.

3.3 Fluxo de trabalho

Para orientar o testador na produção de projetos de teste com a QSEE-TAS, um fluxo de trabalho foi proposto. A Figura 3.3 mostra uma ordem de execução de possível para o desenvolvimento de testes com a QSEE-TAS.

É importante perceber que um projeto de teste pode ser criado vazio ou a partir dos dados importados de outro projeto. No caso de um projeto de teste vazio, atividades como cadastro de dados das propriedades da IST, itens e casos de teste, ciclos de teste, canais de comunicação, mensagens de interação com a IST e passos de teste devem ser efetuadas antes da execução do teste. Entretanto, uma vez que um projeto de teste é baseado em outro, por meio da importação dos dados, a execução dos testes pode proceder tão logo a

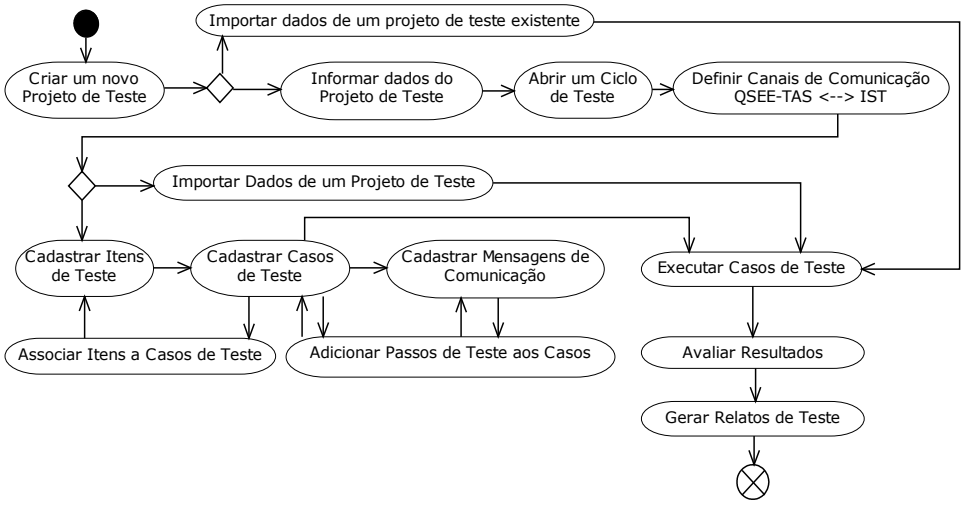


Figura 3.3 - Fluxo de trabalho para as atividades de teste com a QSEE-TAS.

atividade de importação seja concluída.

O testador não é obrigado a seguir exatamente esse fluxo de trabalho. De fato, para que o testador aplique um teste, basta que o projeto de teste tenha ao menos um caso de teste com um ou mais passos de teste associados. Entretanto, para a composição de um relato de teste, o testador deve iniciar pelo menos um ciclo de teste, antes de iniciar a execução. A Seção 3.4 descreve as principais funcionalidades da ferramenta, detalhando o que pode ser feito nas atividades apresentadas na Figura 3.3.

3.4 Funcionalidades

A ferramenta QSEE-TAS foi concebida para apoiar a execução de testes funcionais de software embarcado em cargas úteis de satélites científicos. As versões iniciais da ferramenta forneciam suporte especificamente à execução e relato automatizados de testes de software embarcado em experimentos de satélites científicos que adotam o protocolo de comunicação OBDH-Exp, proprietário do INPE, como interface de comunicação com outros computadores a bordo do satélite (SILVA et al., 2006).

Entretanto, um padrão foi identificado e implementado para dar suporte à especificação de diversos tipos de mensagens definidas por outros protocolos. A Figura 3.4 mostra conceitualmente a ligação entre as funcionalidades principais da QSEE-TAS e dos módulos (*plug-ins*) SPAC. O Capítulo 4 descreve mais detalhes sobre os módulos SPAC.

De acordo com a Figura 3.4, a ferramenta QSEE-TAS gerencia as seguintes entidades básicas:

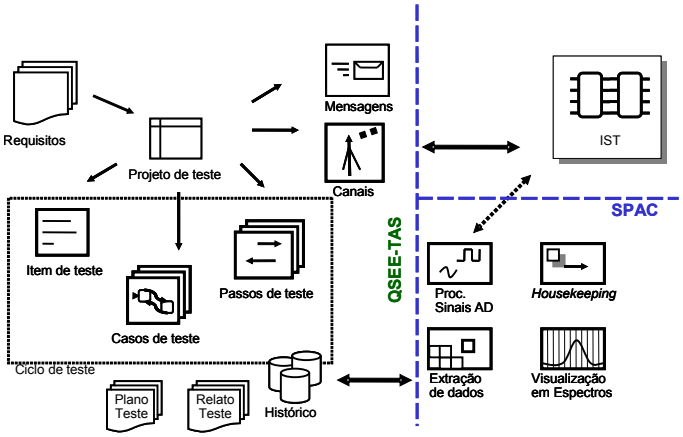


Figura 3.4 - Visão conceitual das funcionalidades da ferramenta QSEE-TAS/SPAC.

- **Projeto de teste**: unidade de informação principal. Mantém todas as informações relacionadas ao teste e cria um contexto global para os itens, casos e passos de teste, ciclo de teste, mensagens (primitivas) de protocolos de comunicação e canais de comunicação;

- **Item de teste**: representa o que será testado. Um item de teste descreve o requisito que deve ser testado e uma visão geral sobre os casos de uso do software e implementam tal requisito. Isto leva a formação dos cenários pelos quais o requisito será testado. Na concepção da ferramenta, um item de teste está relacionado com um ou mais requisitos, um ou mais casos de teste e, eventualmente, com outros itens de teste;

- **Caso de teste**: representa um cenário em que itens de testes podem ser exercitados. Um caso de teste deve reproduzir a seqüência de passos (instruções) necessária para exercitar um caso de uso da IST. A documentação do caso de teste inclui a descrição do critério para a determinação do veredito (i.e. passou ou falhou). Um caso de teste possui um ou mais passos de teste;

- **Passos de teste**: um conjunto não vazio de instruções executáveis que estão relacionadas a um caso de teste específico. Um passo de teste pode ser simplesmente proceder com uma ação ou observação externa à ferramenta (i.e. inspecionar o estado de uma tela ou interagir manualmente com algum equipamento da bancada de testes), ou ser uma ação automática (i.e. enviar uma mensagem de comunicação para a IST);

- **Canais**: representam as vias de comunicação de dados fim a fim entre a ferramenta QSEE-TAS e a IST. Um canal pode encapsular várias camadas de

protocolos segundo o modelo de referência OSI (MILLER, 1981) e pode ser interpretado como plataforma para os *Service Access Points* (SAPs) cuja semântica depende do tipo de mensagem que é enviada ou recebida por ele (ZIMMERMANN, 1988);

- **Mensagem**: um conjunto de zero ou mais campos de dados. Uma mensagem representa uma primitiva de comunicação de dados da camada de protocolo representada por um canal. Uma tupla de mensagens (S, R) representa a solicitação com a respectiva resposta esperada; um conjunto de tuplas constitui a seqüência de passos de teste de um dado caso de teste. A QSEE-TAS mantém listas de mensagens com estruturas pré-definidas pelo testador, chamadas de Biblioteca de Mensagens. Uma Biblioteca de Mensagens pode ser reusada em diversos projetos de teste, sendo particularmente útil na definição de passos de teste repetitivos. Se uma mensagem é utilizada com freqüência, o testador pode colocá-la na Biblioteca de Mensagens para que outros passos de teste a usem;

- **Ciclo de teste**: compreende os itens, casos e passos teste e o histórico de execução dos testes numa dada época. Um ciclo de teste deve ser aberto quando se decide iniciar um conjunto de execuções de teste (sessões de teste) que servirá de base para a geração dos relatos. O testador pode decidir encerrar um ciclo de teste quando algo de relevante impactar o projeto como, por exemplo, a mudança na versão da IST ou de algum documento associado aos testes, mudança na formação da equipe de testes, entre outros, e;

- **Histórico**: toda atividade relevante da QSEE-TAS é gravada no histórico do projeto de teste. Assim, pode-se, por exemplo, recuperar a conversação da QSEE-TAS com uma IST e auditar problemas que eventualmente ocorram durante o teste.

  Uma visão mais abrangente sobre o relacionamento entre as entidades de dados da QSEE-TAS está apresentada Seção A.4 (pág. 99). Lá encontram-se uma visão do modelo ER (Entidade-Relacionamento) lógico (Figura A.4) e físico (Figura A.5).

No contexto da ferramenta QSEE-TAS, o plano de teste refere-se a um relatório contendo a organização do projeto de teste, sem informações do histórico ou de dados relativos à execução de casos de teste. Nele consta uma matriz de rastreabilidade dos itens de teste e casos de teste que é útil para realizar análises de impacto ou estimar o esforço da aplicação dos testes em caso de mudança nos requisitos da IST ou no ambiente de teste.

O relato de teste é munido de mais informações se comparado ao plano de teste, pois ele incorpora o histórico de execução dos casos de teste que, por sua vez, incorpora os itens que participaram de um dado ciclo de teste.

3.4.1 Projeto de teste

O Projeto de Teste é a principal entidade mantida pela QSEE-TAS, oferecendo ao testador interfaces específicas para mantê-las. A partir da interface gráfica principal (Figura 3.5), o testador tem acesso às informações do Projeto de Teste.

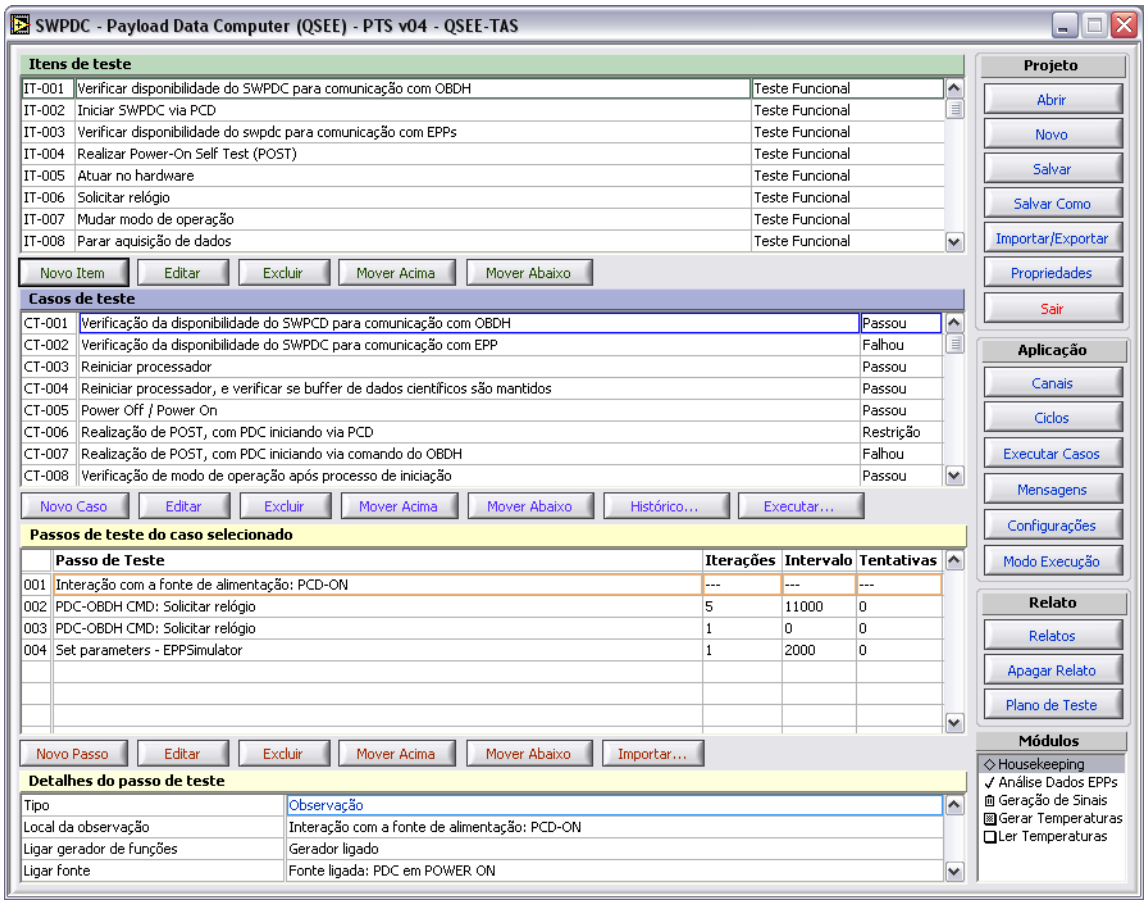


Figura 3.5 - Interface principal da QSEE-TAS com um projeto de teste carregado.

Como apresentado na Figura 3.5, existem grades (tabelas) para acesso aos Itens de Teste, Casos de Teste, Passos de Teste e Detalhes dos Passos de Teste. Esta visão permite o acesso às operações de edição dessas entidades.

Os botões, no lado direito, estão organizados para guiar o acesso às operações que afetam o projeto de teste globalmente, nas seguintes categorias:

- **Projeto**: edição de propriedades do projeto teste, importação e exportação de dados;

- **Aplicação**: edição de configurações, canais de comunicação, mensagens de protocolos de comunicação, e execução massiva dos casos de teste;

- **Relato**: emissão de relatos de teste;

- **Módulos**: acesso aos módulos externos (*plug-ins*) que não fazem parte da ferramenta principal (QSEE-TAS) que somente é visível caso haja algum módulo registrado.

O projeto de teste é constituído de dados como o nome e versão da IST, código localizador do documento de especificação de requisitos de software, identificação das pessoas que compõem a equipe de teste. Alguns desses dados são geralmente obtidos a partir dos documentos de especificação de requisitos. Um exemplo de entrada desses dados é apresentado na Figura 3.6. Eles são importantes para a abertura do ciclo de teste e, portanto, para a geração do relato de teste.

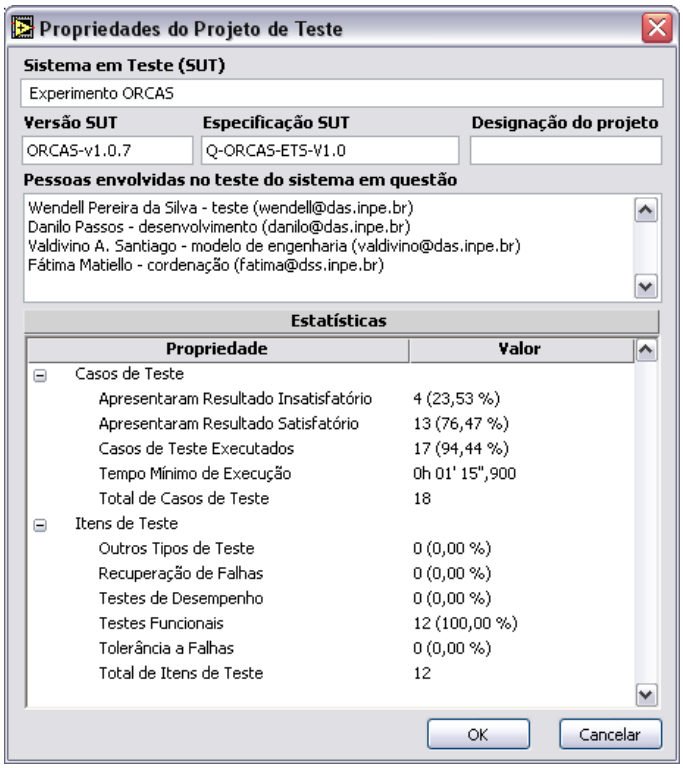


Figura 3.6 - Propriedades de um projeto de teste.

3.4.2 Itens de teste

Uma vez que as propriedades do projeto são informadas, o testador descreve os itens de teste. Os itens de teste estão relacionados com os requisitos funcionais e não funcionais da IST. O testador deve escolher uma classificação para o item como sendo um teste funcional, ou teste de tolerância a falhas, ou de teste de desempenho. A Figura 3.7 apresenta um exemplo de mapeamento do item *Carga de dados em memória*, classificado como item de teste funcional, associado aos casos de teste CT-14, CT-15 e CT-16.

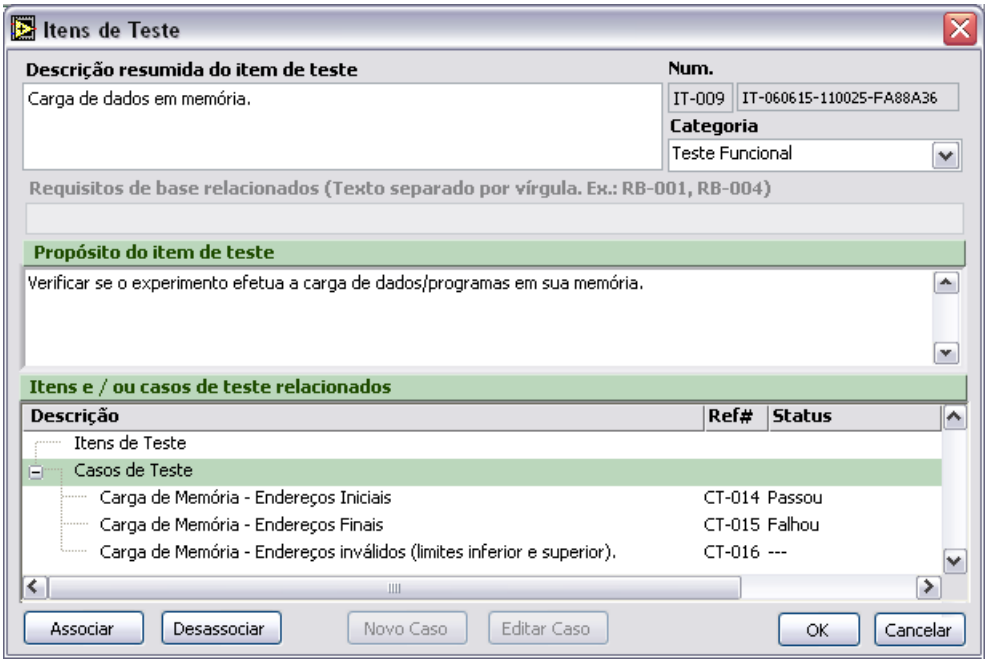


Figura 3.7 - Mapeamento de um item de teste.

3.4.3 Casos de teste

Na QSEE-TAS, um caso de teste deve descrever o cenário no qual um item de teste pode ser exercitado. Aqui, exercitar significa executar um ou mais passos de teste capazes de estimular a IST, seja na forma de procedimento manual, seja por *script* automatizado montado pela QSEE-TAS. Assim, um caso de teste determina a ordem de interação com a IST, considerando características como temporização, sinalização, eventos e verificação de condições *pass/fail*. Essa ordem de interação é representada pelo conjunto de passos de teste que dita as instruções associadas a cada caso de teste. Desta forma, o testador não deve se preocupar em como o caso de teste será executado. Em vez disso, ele especifica quais são os passos que compõem um caso. Detalhes da edição de passos em um caso de teste estão descritos na Seção 3.4.4.

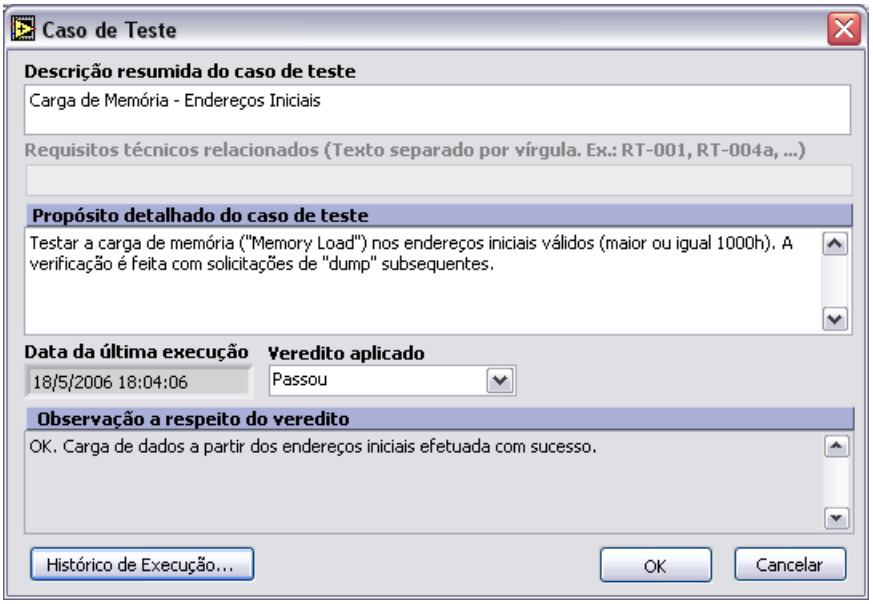


Figura 3.8 - Mapeamento de um caso de teste.

A tela de entrada de dados que documenta um caso de teste em linguagem natural é apresentada na Figura 3.8. O testador deve preencher pelo menos a descrição resumida do caso de teste. Os dados na parte inferior da tela, abaixo do campo **Propósito detalhado do caso de teste**, são visíveis apenas se o caso foi executado ao menos uma vez.

3.4.4 Atribuição dos passos de teste

A inclusão de um caso de teste é concluída quando pelo menos um passo de teste lhe é atribuído. A QSEE-TAS permite a inclusão de dois tipos de passos de testes: Solicitação - Resposta, ou Observação Externa.

O passo de teste do tipo Solicitação-Resposta é destinado à comunicação com uma dada IST, permitindo que o testador escolha (ou inclua uma nova) mensagem de solicitação, de acordo com o protocolo da IST, que deve ser enviada por algum canal de comunicação. Adicionalmente, uma mensagem de resposta esperada pode ser associada indicando o que deve ser obtido do canal de comunicação em resposta à solicitação. É possível, ainda, especificar atrasos no envio da solicitação e estipular quantas vezes a solicitação será enviada à IST (útil para simular solicitações duplicadas). Quando uma Solicitação - Resposta é executada na fase de execução do caso de teste, se uma resposta esperada tiver sido especificada, sua estrutura será conferida com a resposta recebida pelo mecanismo de checagem de conjuntos regulares (oráculo de teste). O canal de comunicação não precisa ser o mesmo para a solicitação e resposta esperada. Canais de comunicação distintos podem ser especificados neste caso. A Sessão 3.4.8 descreve em detalhes a execução dos testes. A

declaração dos tipos de canais de comunicação oferecidos pela QSEE-TAS está descrita na Sessão 3.4.6.

Uma Observação Externa permite ao testador especificar um *check-list* contendo instruções manuais que, por alguma razão, não podem ser traduzidas em interações automáticas. Cita-se, como exemplo, os procedimentos de configuração da bancada de teste como simuladores, checagem de instrumentação (fontes de alimentação, sinais em osciloscópios), entre outros.

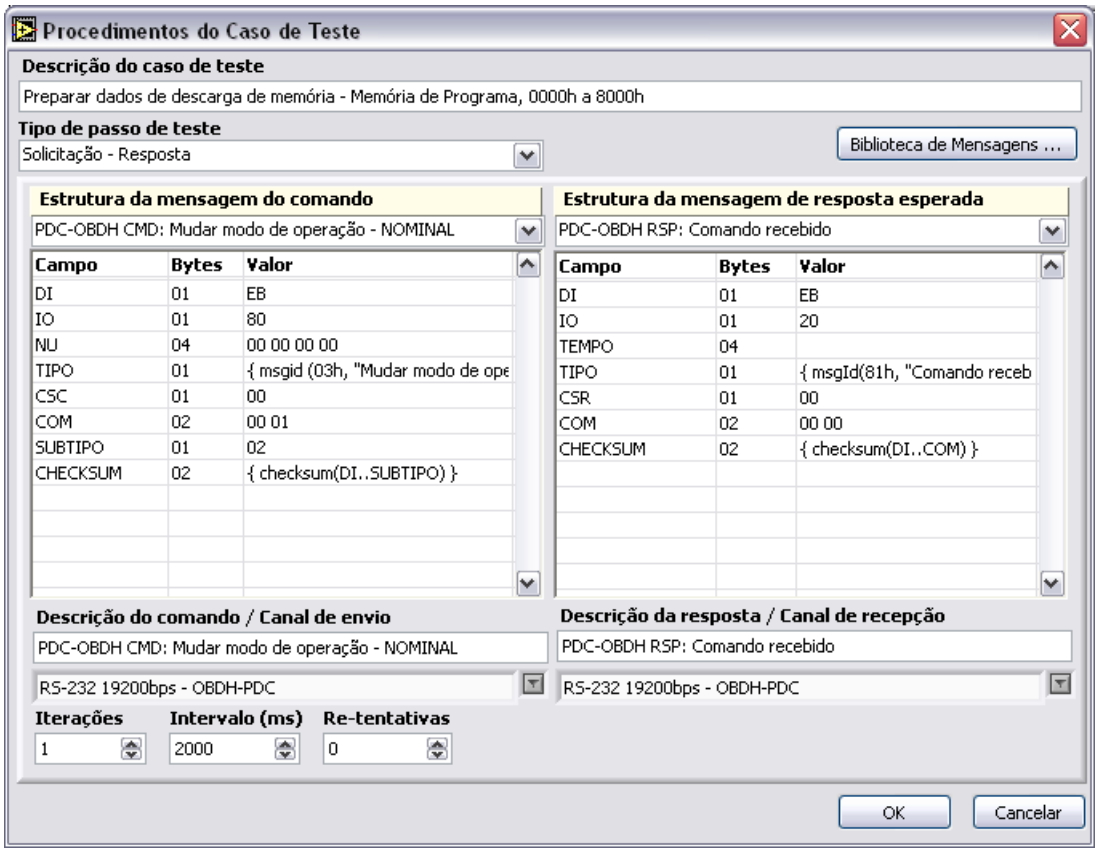


Figura 3.9 - Edição de um passo de teste do tipo Solicitação - Resposta.

A Figura 3.9 apresenta um esboço da tela de edição de um passo de teste do tipo Solicitação - Resposta, enquanto que a Figura 3.10 ilustra um exemplo de entrada de dados de Observações Externas à ferramenta de teste.

3.4.5 Especificação do formato das mensagens do protocolo de comunicação

Em geral, um protocolo de comunicação pode ter sua especificação dividida em duas partes: a especificação das primitivas (mensagens) de comunicação e a dinâmica de troca de primitivas (HOLZMANN, 1991). A dinâmica de troca de mensagem é descrita pelo

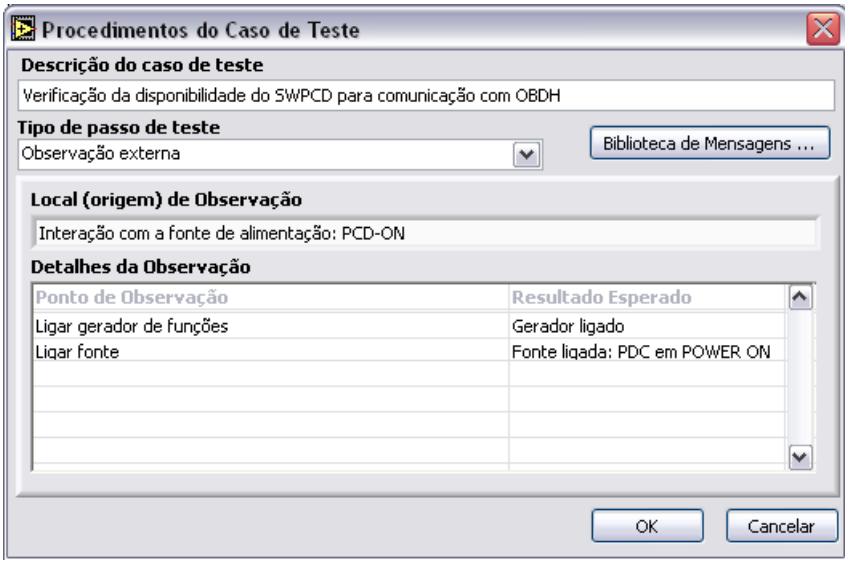


Figura 3.10 - Edição de passo de teste do tipo Observação Externa.

conjunto de interações entre a ferramenta de teste e a IST associados a um caso de teste. Esta seção aborda como as mensagens do protocolo são especificadas na QSEE-TAS.

Uma das formas de interação da QSEE-TAS com a IST é por meio da troca de mensagens. Com base num sistema simples de representação e codificação de mensagens, a QSEE-TAS permite que o testador descreva as mensagens que a IST deve estar apta a entender. Para isso, a QSEE-TAS define uma linguagem experimental chamada Linguagem de Definição de Mensagens (LDM). Além da estrutura da mensagem, a LDM permite definir a semântica associada aos seus campos. Porém, a interface gráfica tenta não expor detalhes desnecessários da LDM ao testador, fazendo uso de assistentes de edição. A LDM é descrita em detalhes na seção A.2.

A Figura 3.11 apresenta um esboço da tela de edição de mensagens. Por meio dela, o testador deve informar para cada campo um identificador (nome do campo), tamanho e conteúdo. O identificador deve ser único. O tamanho representa a quantidade de bytes ocupada pelo campo; se for um tamanho fixo, um número inteiro (em base decimal) deve ser informado; se for variável, o valor deve ser descrito na forma de intervalo $m..n$, onde m é o tamanho mínimo e n é o tamanho máximo do campo. O valor do campo, por *default*, assume representação hexadecimal, sempre com dois dígitos para representar cada byte. Por exemplo, `EB 92 AF`; caso o valor não respeite o tamanho previamente especificado, ele será truncado (se exceder ao tamanho), ou será preenchido com zeros à direita (caso seja menor). O valor também pode ser representado como uma *string*, com os caracteres entre aspas; exemplo: “SET FAULT-INJECTION ON\r\n”.

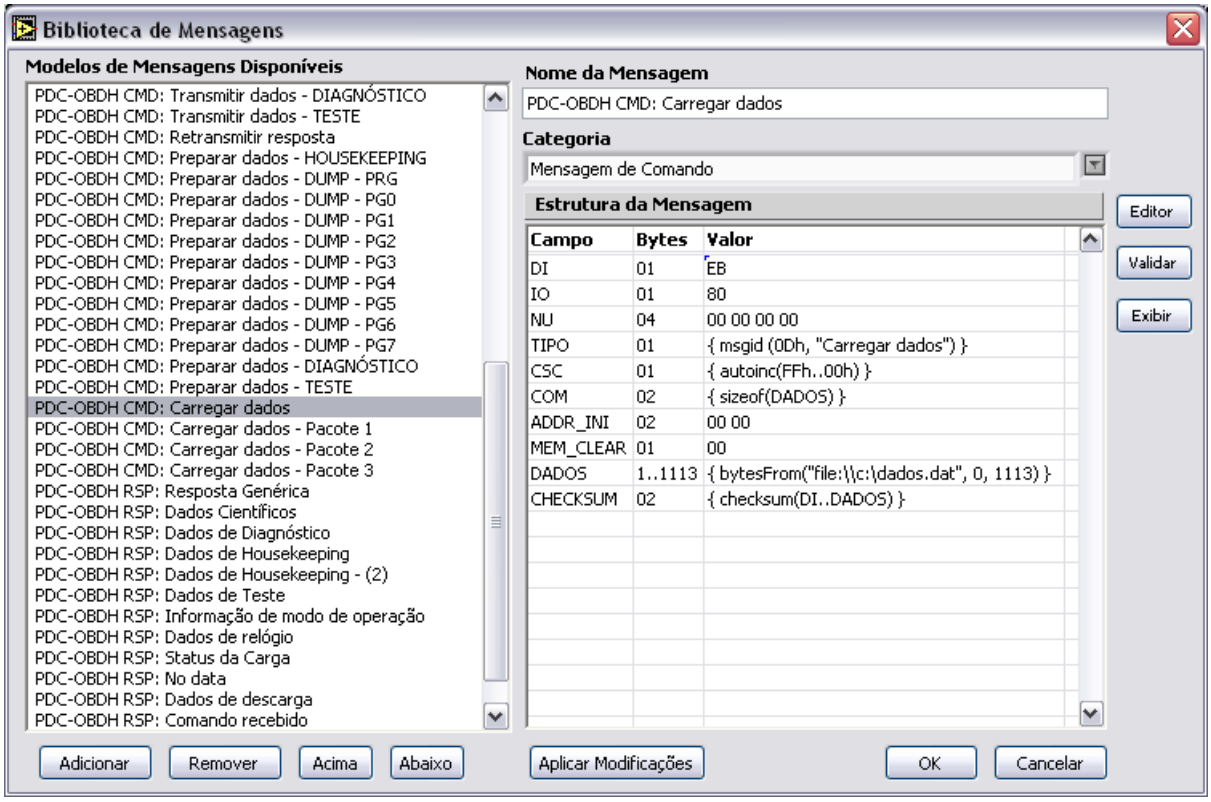


Figura 3.11 - Edição da estrutura de mensagens.

3.4.6 Canais de comunicação

Um canal de comunicação, no contexto da QSEE-TAS, é uma abstração que encapsula as operações necessárias ao envio e recebimento de dados fim a fim. Existe, até o momento, suporte para os seguintes tipos de canais de comunicação:

- RS-232: Comunicação serial via interface com padrão RS-232 (UART), ou cabo adaptador USB:RS-232;
- Porta Paralela: Comunicação via porta paralela do microcomputador (LPT);
- TCP/IP: Transporte de dados confiável fim a fim orientado a conexão.

Por meio da tela de edição dos canais, o testador pode definir mais de uma instância de cada tipo de canal. Por exemplo, um canal `C01` pode ser definido como sendo do tipo RS-232, associado à porta COM1, com taxa de 19200 bps, como ilustrado na Figura 3.12.

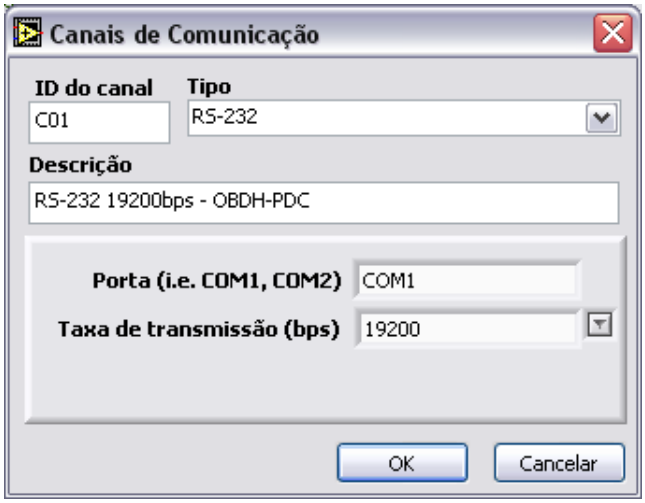


Figura 3.12 - Edição de um canal de comunicação do tipo RS-232.

3.4.7 Ciclos de teste

Antes de iniciar uma sessão de teste na qual se deseja que a ferramenta rastreie corretamente a aplicação dos testes, um ciclo de teste deve ser aberto. O objetivo de um ciclo de teste é responder a seguinte pergunta: quem aplicou quais casos de testes, de qual versão da IST, numa dada época? Com isso, é possível rastrear a evolução do teste, além de permitir que relatos de teste estejam focados num ciclo de teste específico, em vez de todos os testes já aplicados a uma dada IST. A Figura 3.13 mostra um esboço da tela de manipulação de ciclos de teste com suas operações relacionadas.

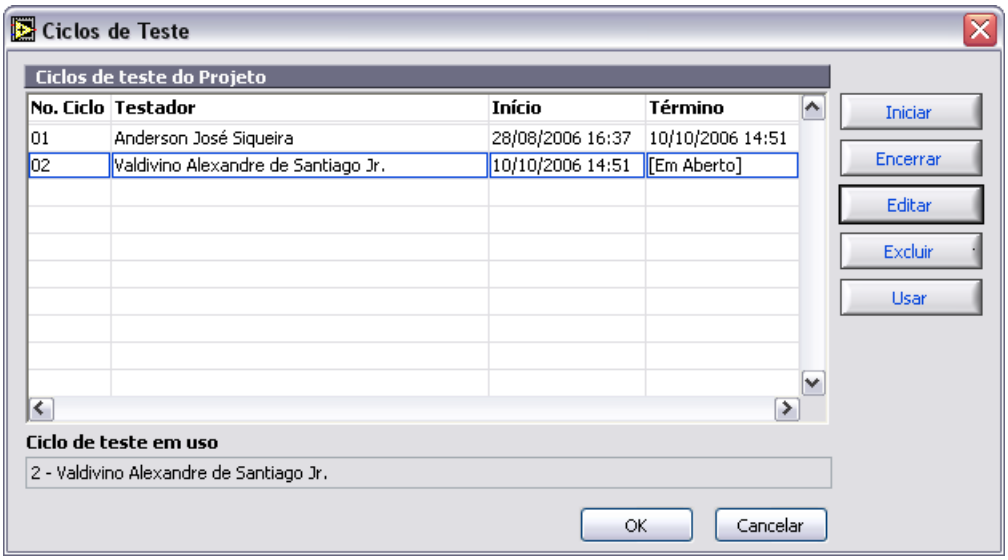


Figura 3.13 - Ciclos de testes: quando e por quem um teste foi aplicado?

O encerramento (ou fechamento) do ciclo marca o fim de uma sessão de teste. A partir de então, não é mais possível rastrear o teste para este ciclo, o que obrigará o testador a abrir novos ciclos para rastrear a aplicação de baterias de teste subseqüentes.

3.4.8 Execução automatizada dos casos de teste

A execução automatizada dos testes funcionais é a principal característica da ferramenta QSEE-TAS. Uma execução de teste funcional na QSEE-TAS pode ser realizada se, no projeto de teste, existir pelo menos um caso de teste contendo pelo menos um passo de teste. Em termos práticos, porém, uma IST possui dezenas ou mesmo centenas de funcionalidades testáveis o que leva à elaboração (ou geração) de inúmeros casos de teste.

Durante o projeto QSEE, a QSEE-TAS apoiou ao teste unitário do SWPDC ainda em sua fase de desenvolvimento, tornando repetível o teste de componentes de forma isolada ou integrada. (SANTIAGO et al., 2007) Como exemplo, cita-se o teste do componente que verifica a sintaxe dos comandos do protocolo de comunicação PDC-OBDH (SANTIAGO; MATTIELLO-FRANCISCO, 2006) implementado no SWPDC. Este fato inspirou a criação de dois modos de execução dos testes funcionais: um modo que fosse simples e rápido (do tipo *click-and-run*) e outro que permitisse a concatenação de casos de testes previamente criados (*select-and-run*). O primeiro foi chamado de Modo de Seleção Simples e o segundo de Modo de Seleção Combinada.

3.4.8.1 Modo de seleção simples

No modo de seleção simples, o testador seleciona um caso de teste e invoca a tela de execução. A tela de execução, mostrada na Figura 3.14, permite que o testador interaja com os passos de teste do caso em questão (mudando a ordem de execução, incluindo ou alterando novos passos, etc.) ou simplesmente inicie a execução de todos os passos de teste.

Quando uma execução está em progresso, toda atividade é gravada. Exemplo de dados que são gravados durante a execução incluem: a marca temporal da atividade (*timestamp*) (com resolução em milissegundo), as mensagens enviadas e recebidas, e o julgamento do comparador da resposta esperada contra a resposta recebida da IST (*pass, no pass*). A atividade do teste em execução aparece no campo “Registro de atividade de teste” da tela de execução.

Como o próprio nome indica, o campo “Parecer final do testador a respeito da execução do caso de teste” é útil para o testador descrever com detalhe o veredito atribuído à IST para o caso de teste em questão. O veredito pode ser um dos seguintes:

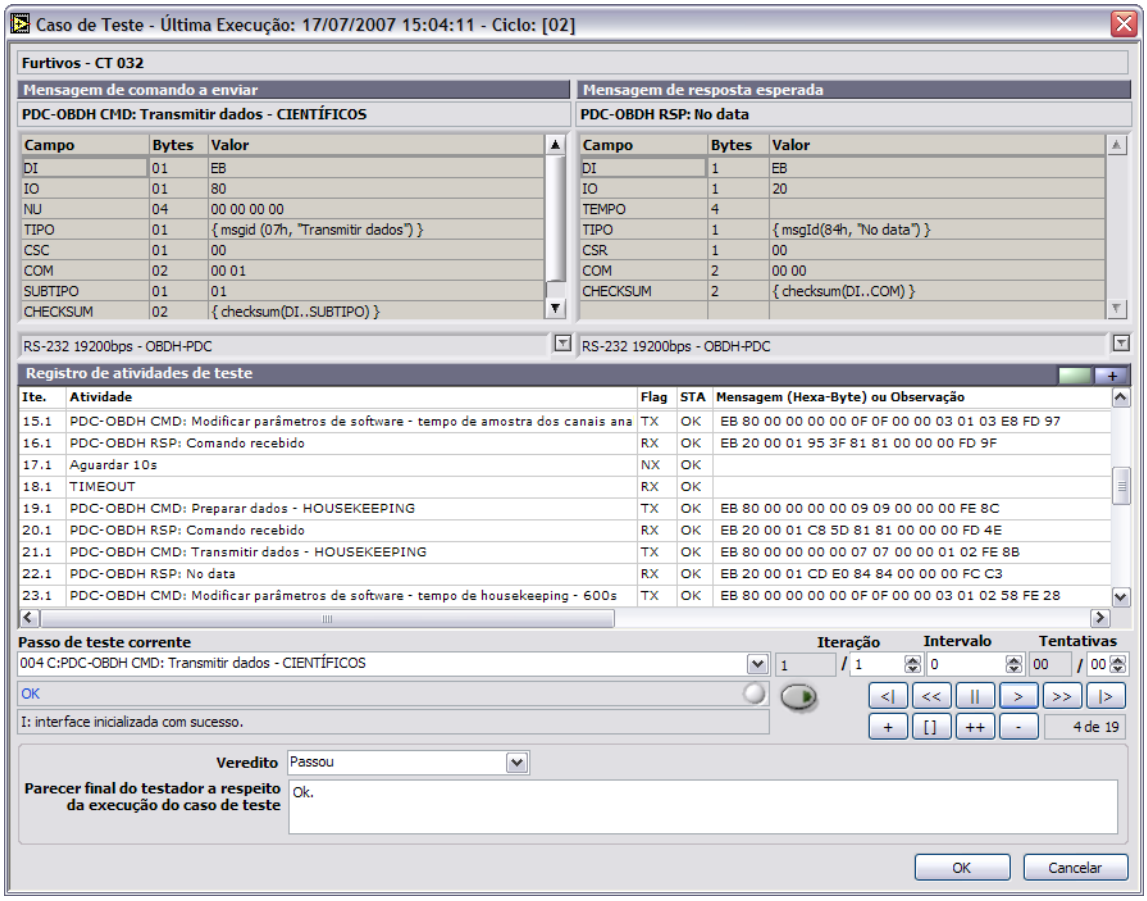


Figura 3.14 - Execução de casos de teste em progresso.

- *Passou*: a IST **não** manifestou anomalia comportamental observável (nehuma falha encontrada) em relação ao resultado esperado;

- *Falhou*: a IST manifestou anomalia comportamental observável (falha revelada) em relação ao resultado esperado;

- *Inconclusivo*: usado para descrever situações quando há dúvidas sobre a validade do caso de teste ou no comportamento observável da IST. Experiências mostraram que isso ocorre, por exemplo, quando há discrepância na interpretação da especificação dos requisitos na visão da equipe de teste em relação à equipe de desenvolvimento. Geralmente este tipo de veredito é discutido em reuniões (SANTIAGO et al., 2007), e tende a convergir para passou, ou falhou, ou passou com restrições;

- *Passou com restrições*: usado quando observa-se que seria desejável que a IST se comportasse de outra forma, que não a observada, entretanto o comportamento observado não pôde, por alguma razão, ser julgado como uma falha. Por exemplo,

omissões de implementação de requisitos desejáveis podem levar a marcação deste veredito, que depende da interpretação da equipe de teste.

Existe, na literatura, referências ao veredito *error* (erro) (BINDER, 2000), usado para denotar falha no ambiente de teste. Entretanto, decidiu-se por não colocar tal veredito, pois se entende que o relato de teste deve conter o rastro dos testes que puderam ser executados.

3.4.8.2 Modo de seleção combinada

Para facilitar a execução de casos de teste de forma combinada, uma interface específica foi criada. No Modo de Seleção Combinada, a ferramenta oferece a possibilidade de juntar casos de teste distintos, em qualquer ordem definida pelo testador. Com isso, é possível criar novos casos de teste mais complexos com base na concatenação de outros casos de teste. A Figura 3.15 ilustra a seleção de múltiplos casos de teste.

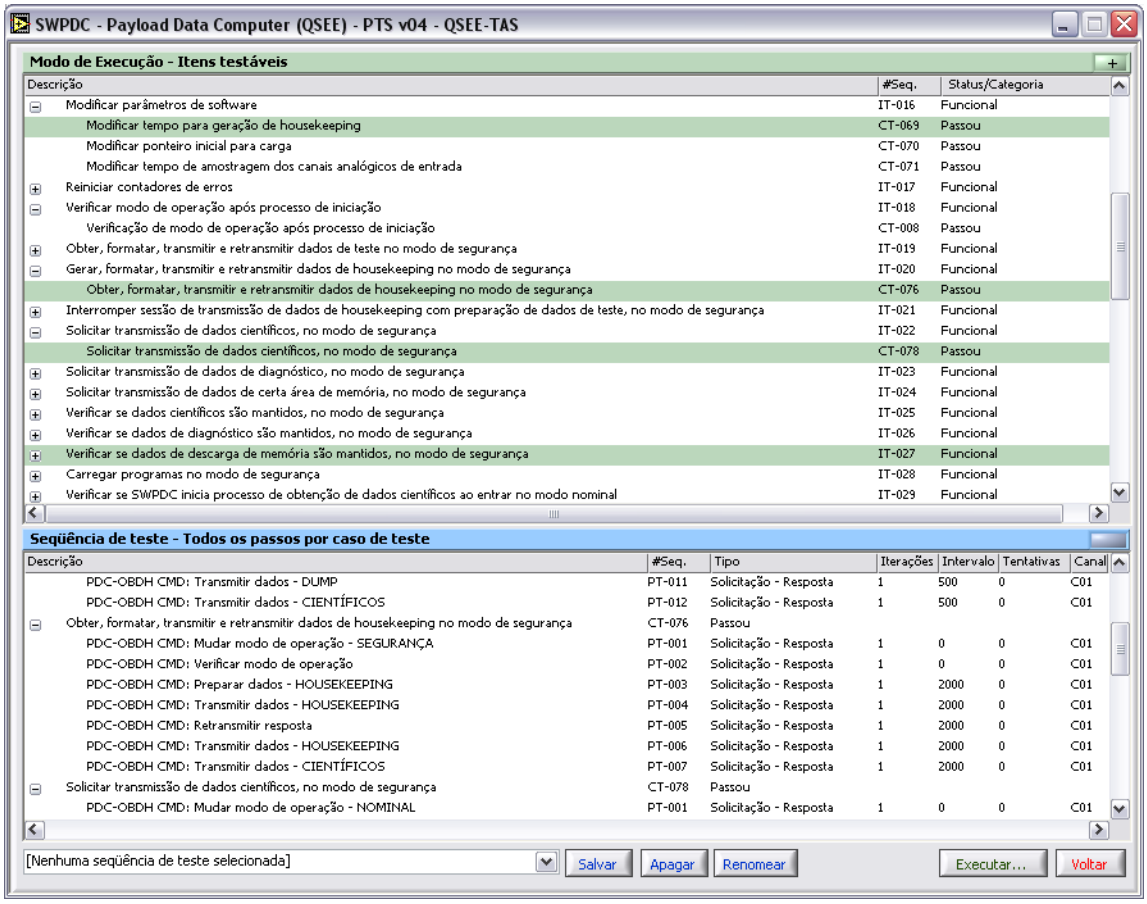


Figura 3.15 - Seleção de múltiplos casos de teste para execução combinada.

Uma vez que a concatenação dos casos de teste é feita, o testador pode comandar a execução. Ao comandar a execução, os casos de teste são executados um a um, porém seguindo a ordem em que foram concatenados, similarmente à execução no modo de seleção simples.

3.4.9 Importação e exportação de dados

A funcionalidade de importação e exportação de dados tem dois objetivos. O primeiro é permitir a troca de dados com outras ferramentas; o segundo, é o armazenamento de projetos de teste em um formato independente de plataforma. Para isso, um arquivo XML foi definido. Para diferenciar arquivos XML que representam projetos de teste da QSEE-TAS de outros arquivos XML, seu nome deve terminar com a extensão ".tes.xml".

Seguindo a recomendação definida pelo W3C [2], a estrutura de um arquivo ".tes.xml" é definida por meio de um arquivo DTD (*Document Type Definition*). Uma listagem completa deste arquivo é apresentada na Seção A.3 (Figura A.3). A Figura 3.16 mostra um esboço da tela do assistente de importação e exportação. Por meio dela o testador pode optar por exportar (ou importar) todo o projeto de teste ou somente os casos de teste.

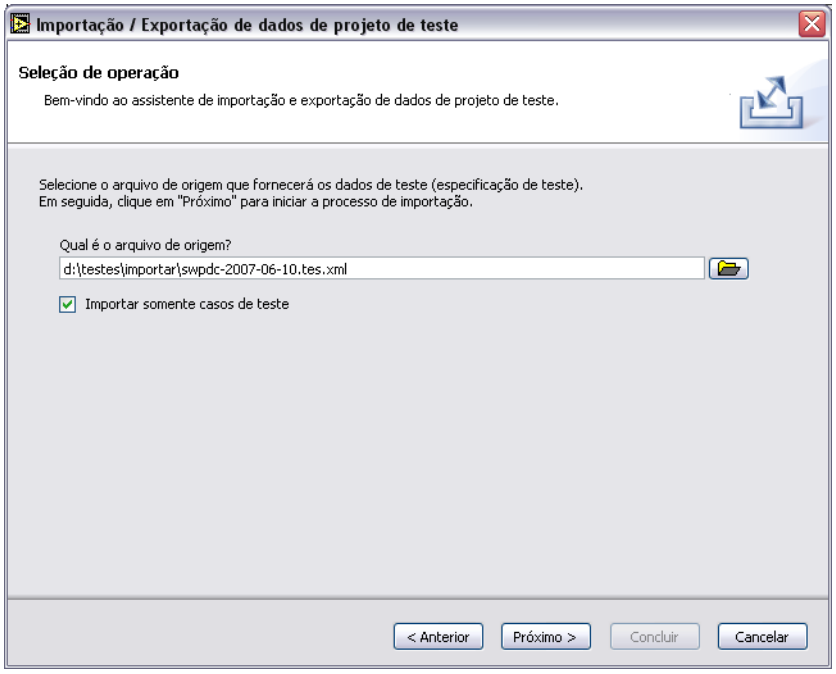


Figura 3.16 - Uma das telas do assistente de importação e exportação de dados de projetos de teste.

[2]World Wide Web Consortium: orgão que cuida dos padrões da *Web* (http://w3c.org).

Esta funcionalidade tem se mostrado útil para construir projetos de teste que combinam dados de projetos de teste de diversas ISTs, auxiliando na preparação de testes de integração.

3.4.10 Geração dos relatos de teste

Um relato de teste pode ser gerado tão logo haja um ciclo de teste aberto que contenha algum caso de teste já executado. Os relatos de teste são gerados em formato XML. Para tornar a visualização do relato mais confortável ao usuário final, *scripts* de transformações XSL estão disponíveis e são executados automaticamente para converter XML em páginas HTML. A Figura 3.17 ilustra o processo de geração dos relatos de teste enquanto que a Figura 3.18 mostra um fragmento de um relato de teste visualizável por meio de um navegador *Web*.

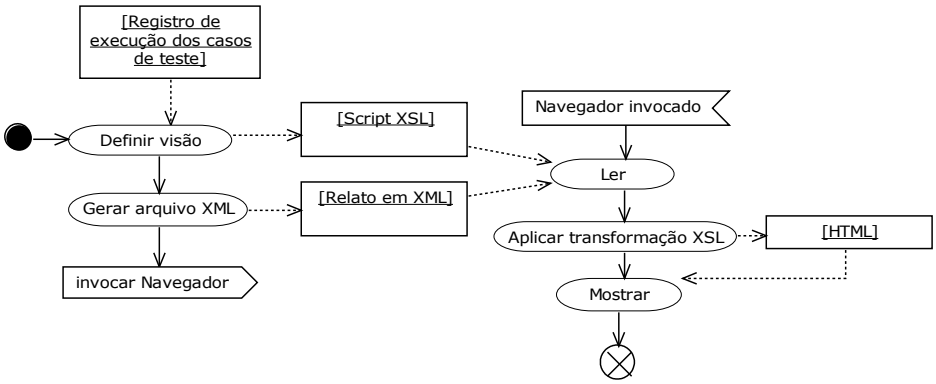


Figura 3.17 - Geração e visualização do relato de teste.

CT-003 — 17/08/2006 14:43:47: Reset microcontroller
Verdict: PASS
Microcontroller was re-stared when requested and the scentific data once acquired was preserved.

Test Case Execution Log:

Timestamp	Flg	Sta	Log data description history
14:40:45,233	TX	OK	EB 80 00 00 00 00 0F 00 00 03 01 00 3C FE 46
14:40:46,578	RX	OK	EB 20 00 04 32 93 81 00 00 00 FD AB
14:40:47,608	TX	OK	EB 80 00 00 00 00 0F 00 00 03 02 90 00 FD F1
14:40:48,936	RX	OK	EB 20 00 04 3B DC 81 00 00 00 FD 59
⋮			⋮
14:41:07,828	RX	OK	EB 20 00 04 85 A1 87 00 00 67 10 0A ED 00 00 80 00 00 00 80 00 02 0F FE 04 60 00 01 00 01 80 00 00 01 80 00 02 10 06 04 60 00 01 00 02 80 00 00 02 80 00 02 10 0E 04 60 00 01 00 03 00 00 00 03 00 00 02 10 16 04 60 00 01 00 03 80 00 00 03 81 A2 02 10 1E 04 60 00 01 00 03 00 00 00 03 00 00 02 10 36 04 60 00 00 00 03 04 61 00 03 04 61 0B 4E F1 AE

Figura 3.18 - Relato de teste transformado em HTML e visualizado via navegador *web*.

O requisito para visualização do relato de teste é ter um navegador com suporte a transformações XSLT instalado na mesma estação de trabalho (*host*) que a QSEE-TAS, como por exemplo, as versões mais atuais dos Mozilla Firefox (versão > 1.5) e Internet Explorer (versão > 5.0).

3.5 A arquitetura da ferramenta

A arquitetura da ferramenta QSEE-TAS tem similaridades com a arquitetura de um *Test Harness* proposta por Knirk (KNIRK, 1996), também referenciada por (BINDER, 2000, p. 963). A Figura 3.19 representa a arquitetura da QSEE-TAS.

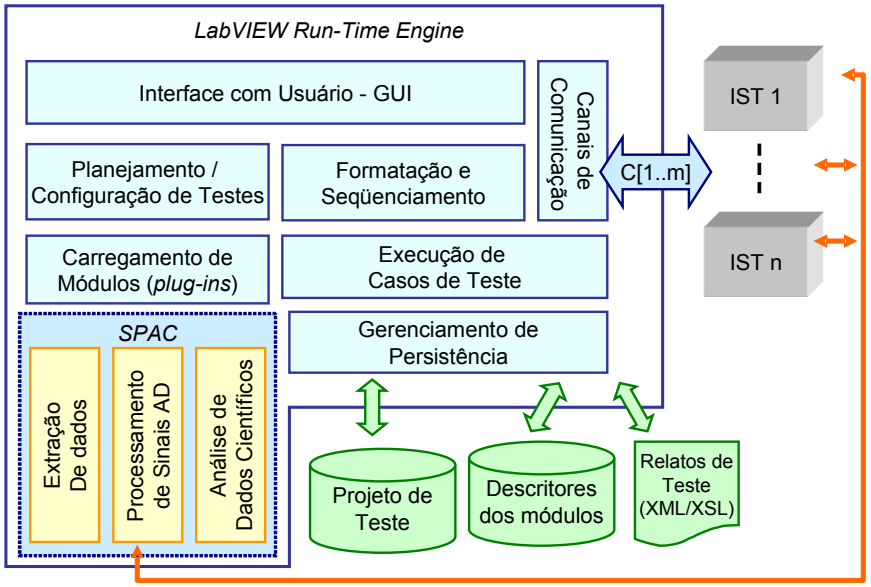


Figura 3.19 - Arquitetura da ferramenta QSEE-TAS/SPAC.

A QSEE-TAS foi concebida para executar sobre o ambiente de tempo de execução (*Run-time*) do *LabVIEW*™. A seguir, são descritas as características principais de cada parte da arquitetura.

- **Interface com Usuário**: Parte dedicada aos componentes de interface com usuário. Foi necessária a implementação de visões de interface gráfica personalizadas com base nos componentes existentes no *LabVIEW*™, como, por exemplo, a visão de registro de atividade da execução dos testes, assistentes (*wizards*) para importação e exportação de dados do projeto de teste.

- **Planejamento / Configuração de Testes**: responsável pela iniciação e condicionamento do projeto de teste no ambiente de tempo de execução da ferra-

menta. Viabiliza a execução dos casos de teste por meio da leitura dos arquivos de configuração;

- **Formatação e Seqüenciamento**: converte as instruções do conjunto de passos de teste num formato adequado para interagir com a IST. Esta parte lida com características de temporização, formatação das expressões regulares para checagem dos formatos de quadros de transmissão e iniciação de estruturas de dados para enviar ou receber dados por meio dos canais de comunicação;

- **Canais de Comunicação**: responsável pelas estruturas de dados e operações sobre os canais de comunicação que ligam a QSEE-TAS à IST. Cada IST pode ter vários canais de comunicação. Por outro lado, um canal de comunicação pode ligar várias ISTs, no caso de um barramento, em que o mecanismo de endereçamento é função da camada de enlace;

- **Execução de Casos de Teste**: munido com os dados providos pela Formatação e Seqüenciamento e pelos Canais de Comunicação, esta parte é responsável por executar os procedimentos operacionais declarados em cada passo de teste. Nesta parte, toda interação entre QSEE-TAS e as IST's é gravada no arquivo de *log* de execução do caso de teste;

- **Gerenciamento de Persistência**: cuida do armazenamento persistente dos dados do projeto de teste. Atualmente, a persistência é feita em arquivos binários cujo formato é definido pelo *LabVIEW*™;

- **Carregamento de Módulos (*plug-ins*)**: realiza a carga dinâmica dos instrumentos virtuais (programas em *LabVIEW*™) externos à ferramenta principal (QSEE-TAS). Os módulos externos são registrados em arquivos descritores de módulos que contêm metadados para que a QSEE-TAS os carregue e os execute. Exemplos desses módulos são aqueles que compõem a ferramenta SPAC, descrita no Capítulo 4.

3.6 Considerações finais

Este capítulo apresentou as funcionalidades e a arquitetura da QSEE-TAS. Em resumo, a ferramenta oferece um subconjunto das funcionalidades recomendadas pela literatura (BEIZER, 1990; BINDER, 2000; IEEE, 1998); a Tabela 3.1 apresenta um comparação entre essas funcionalidades a as capacidades da QSEE-TAS.

A descrição das capacidades agregadas à ferramenta QSEE-TAS continua no Capítulo 4 que apresenta os módulos SPAC. Carregados dinamicamente pela QSEE-TAS, os módu-

Tabela 3.1 - Comparação das capacidades desejáveis de um *test harness* e o suporte dado pela QSEE-TAS.

Capacidade	**Suporte da QSEE-TAS**
Implementação de casos de teste	SIM
Iniciação do ambiente de teste	SIM
Construção e controle de *stubs*	NÃO
Instrumentação e análise de cobertura de teste	NÃO
Interface com a IST para envio de mensagens	SIM
Controle de execução do teste	SIM
Comparador automatizado	SIM
Registro de resultado dos testes (*logging*)	SIM
Geração automática de dados de entrada	NÃO
Produção automática de resultados esperados (*test oracle*)	SIM (Parcialmente)
Uso de um sistema existente como oráculo	NÃO
Suporte a testes de regressão	SIM
Rastreamento de procedimentos executados manualmente	SIM
Suporte a documentação de teste	SIM
Interoperabilidade com outras ferramentas	SIM
Simulação do ambiente alvo	SIM
Suporte a depuração	NÃO

Fonte: Adaptado de Binder (2000, p. 961).

los SPAC evidenciam a capacidade de extensão da ferramenta principal (QSEE-TAS) e ajudam na validação de requisitos não funcionais como, por exemplo, a análise de dados científicos e a verificação do desempenho da IST.

4 SPAC: PROCESSAMENTO E VISUALIZAÇÃO DE DADOS CIENTÍFICOS PARA UMA MISSÃO DE SATÉLITE DE ASTROFÍSICA

Como apresentado no Capítulo 3, a QSEE-TAS auxilia na execução automatizada dos testes. Esta atividade tende a gerar uma grande quantidade de dados a partir da captura do fluxo de dados das transações entre a QSEE-TAS e a IST. Por isso, existe a necessidade de processar esses dados condicionando-os como entrada de outras etapas do processo como, por exemplo, a validação do caráter científico do instrumento, uma vez que o seu correto funcionamento está relacionado com a real utilidade da missão como um todo.

Nesta perspectiva, não basta que cada transação entre *test harness* (QSEE-TAS) e IST tenham ocorrido com sucesso, sintática e semanticamente. Além disso, é necessário verificar se o conjunto de dados obtido está de acordo com requisitos de desempenho. Por exemplo, considere-se que a IST deva adquirir dados de um sensor com período de s milissegundos, os quais são transmitidos em pacotes de dados por meio de solicitações periódicas à IST. Uma vez que esta série temporal é obtida via transações do protocolo de comunicação (solicitações - respostas) bem sucedidas, deve-se verificar o conteúdo dos dados obtidos (leitura do sensor) possui estampas de tempo adequadas que derivem aquele período de aquisição de s milissegundos, ou se eventuais perdas de sincronismo ou de pacotes de dados são toleráveis. Além disso, quando aplicável, transformações e visualizações desses dados tornam-se importantes para manter os especialistas no domínio e *stakeholders* informados a respeito da atividade de teste, com artefatos que lhes são familiares.

Para a equipe de desenvolvimento e para os cientistas responsáveis pelo instrumento científico, é importante haver uma ferramenta que permita a interação com os instrumentos ao nível de percepção do usuário final, aliada à execução de casos de teste e geração de relatos de testes de forma automática. Estas características podem ajudar a diminuir o tempo gasto nos diversos testes unitários, integrados e de regressão, além de poder alinhar os focos científico e técnico empenhados ao software (SILVA et al., 2007).

Portanto, justificadamente, esta tarefa de análise e visualização de dados é passível de automação. Este capítulo descreve módulos integráveis à ferramenta QSEE-TAS que foram construídos especialmente para auxiliar na extração, transformação e visualização de dados provenientes do SWPDC, que é uma carga útil projetada e construída no contexto do projeto QSEE (SANTIAGO et al., 2007), e que faz o papel da IST no processo de VV&T. O conjunto de módulos foi chamado de Software para Processamento e Análise de Dados Científicos (SPAC).

Este capítulo está estruturado da seguinte forma: a Seção 4.1 descreve o processo de re-

gistro dos módulos na QSEE-TAS, que os disponibiliza ao usuário por meio de um *menu* específico. A Seção 4.2 descreve o processo de seleção de registros a partir das transações realizadas entre a QSEE-TAS e a IST. A Seção 4.3 mostra os módulos capazes de visualizar dados de *housekeeping*, como *logs* de eventos, contadores de erro e amostras de temperaturas, dados do instrumento controlado pelo SWPDC, como dados científicos (histogramas de fontes de raios-X), e dados de diagnóstico e teste da eletrônica do instrumento. Na Seção 4.4 apresentam-se os módulos desenvolvidos para interagir com ISTs por meio de canais Analógico/Digitais (AD) de placas de aquisição de dados. Por fim, a Seção 4.5 apresenta as conclusões e considerações finais deste capítulo.

4.1 Carga de módulos

A QSEE-TAS oferece suporte a módulos externos, não ligados diretamente aos módulos internos da ferramenta principal (QSEE-TAS). A Figura 4.1 ilustra o processo de carga de módulos.

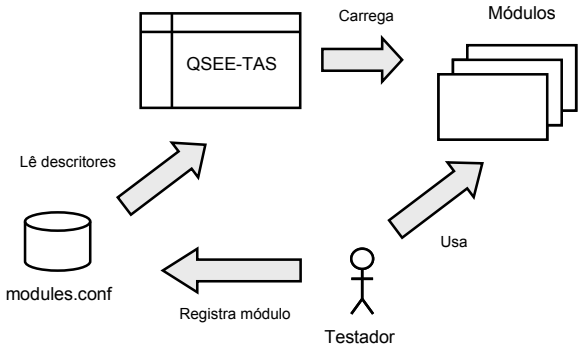


Figura 4.1 - O processo de carga de módulos.

O processo de carga do módulo é simples: resume-se à leitura do arquivo descritor de módulos, que aponta para os instrumentos virtuais [1] principais de cada módulo.

Uma vez lidos, os rótulos textuais de cada módulo são apresentados no menu Módulos - posicionado na parte inferior direita da tela principal da QSEE-TAS (Figura 4.3) -, na ordem em que eles são encontrados no arquivo descritor. A Figura 4.2 ilustra o conteúdo de um arquivo descritor de módulos, que deve ser nomeado `modules.conf` e estar presente no mesmo diretório que o arquivo principal da QSEE-TAS (`qsee-tas.vi`).

Caso o arquivo alvo do módulo não exista, ele será ignorado. Neste exemplo, note-se que o primeiro descritor informado pelos identificadores `modules[0].label` e `modules[0].file`

[1] Programa em Linguagem G, construído com o *LabVIEW*™

```
#Módulos registrados para acesso via QSEE-TAS
[modules]
modules[0].label = Housekeeping
modules[0].file  = ./spac/SPAC - Relatos de Eventos.vi

modules[1].label = Análise Dados EPP
modules[1].file  = ./spac/SPAC - Analise Dados EPPs.vi

modules[2].label = Gerador de Sinais
modules[2].file  = ./spac/SPAC - Gerador de Sinais.vi

modules[3].label = Gerar Temperaturas
modules[3].file  = ./spac/SPAC - Temperaturas.vi

modules[4].label = Ler Temperaturas
modules[4].file  = ./spac/SPAC - Leitor de temperaturas.vi
```

Figura 4.2 - Exemplo do conteúdo do arquivo `modules.conf`.

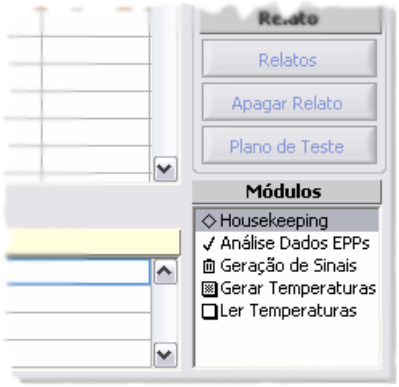


Figura 4.3 - Recorte de tela enfatizando os módulos SPAC disponíveis.

(Figura 4.2) são mapeados na primeira opção do menu Módulos (Figura 4.3). Este mapeamento ocorre similarmente para os demais módulos registrados em `modules.conf`.

4.2 Extração de dados

Como explicado na Seção 3.5, a QSEE-TAS armazena toda atividade relevante da execução dos testes. Esta seção trata dos detalhes do registro de atividade (*log*), e sobre como as informações contidas nele podem ser processadas de acordo com os propósitos de cada módulo. A Figura 4.4 ilustra o formato do arquivo de *log* usado para gravar as atividades de execução dos testes, usando três campos de dados:

- **timestamp**: data e hora da geração do registro. Tem tamanho fixo de 64 bits e e representa a quantidade de milisegundos desde a zero hora do dia 01/01/1904, hora de *Greenwich* (NI, 1998);

- **tipo**: *string* que representa o tipo de registro, ou seja o conteúdo do campo "dado"; alguns exemplos incluem:
  - **'TX_OK'**: mensagem de dados transmitida com sucesso;
  - **'RX_OK'**: mensagem de dados recebida com sucesso;
  - **'INFO'**: mensagem textual de caractere informativo;
- **dado**: informações da atividade registrada; pode ser um texto informativo ou uma mensagem de dados trocadas entre a QSEE-TAS e a IST durante o teste;

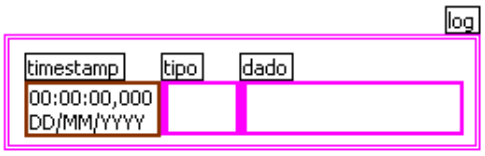


Figura 4.4 - *Cluster* (registro) que representa um registro de *log*.

O módulo de extração de dados da SPAC pode ser entendido como uma função que recebe como entrada uma quádrupla (d_i, d_f, t, q), onde:

- d_i, d_f são as datas inicial e final que representam o período da busca;
- t é o tipo de registro, e;
- q é uma expressão regular que representa o padrão do conteúdo da mensagem no campo 'dado';

e produz como saída o conjunto de registros de *log* que atendem à especificação da busca.

Por exemplo, se a quádrupla ("10/01/2006 13:30:00,000", "10/01/2006 14:30:00,000", "RX_OK", "`(EB04h)[.]{4}(85h)[.]+`") fosse submetida ao módulo de extração de dados, seriam recuperados todos os registros de *log* com as mensagens de dados recebidas das 13:30 às 14:30 horas do dia 10 de janeiro de 2006, iniciadas com a seqüência de bytes `EB04h`, seguido de uma combinação qualquer de quatro bytes, seguido do byte `85h` e terminando com um ou mais bytes quaisquer. Em outras palavras, segundo o protocolo de comunicação PDC-OBDH (SANTIAGO; MATTIELLO-FRANCISCO, 2006), seria equivalente a buscar, no período especificado, todas as mensagens de dados científicos do conjunto imageador de raios-X EPP HXI-1.

Filtragens de dados mais complexas podem ser feitas por outros módulos usando este mecanismo simples de busca seqüencial. Como exemplo, cita-se o desemcapsulamento de dados feito pelo módulo de visualização de dados de *housekeeping* (Figura 4.6), que procura dados em partes da mensagem que necessitam de lógica mais elaborada, como amostras de temperaturas, por exemplo. Este é um exemplo de filtragem orientada ao conteúdo dos campos de determinadas mensagens para produzir matrizes (tabelas) de valores. A vantagem deste método é permitir trabalhar com grandes volumes de registros com pouco uso de memória RAM, porém, ao custo de longas buscas seqüenciais no arquivo de *log*.

Entretanto, otimizações são feitas quando o primeiro registro é encontrado e quando o valor da data inicial permanecer o mesmo ou for sempre maior que o último valor em buscas subseqüentes. Quando isto ocorre, não é necessário ler todos os registros a partir do início do arquivo, pois sua ordem natural é cronologicamente crescente. Outra otimização possível, é mover o ponteiro de leitura do arquivo de *log* contra a ordem cronológica, até encontrar o primeiro registro com *timestamp* aderente ao critério de busca. Contudo, se alguma busca não fizer referência a um período específico, seu tempo de processamento poderá ser bem longo, dependendo do tamanho do arquivo de *log*.

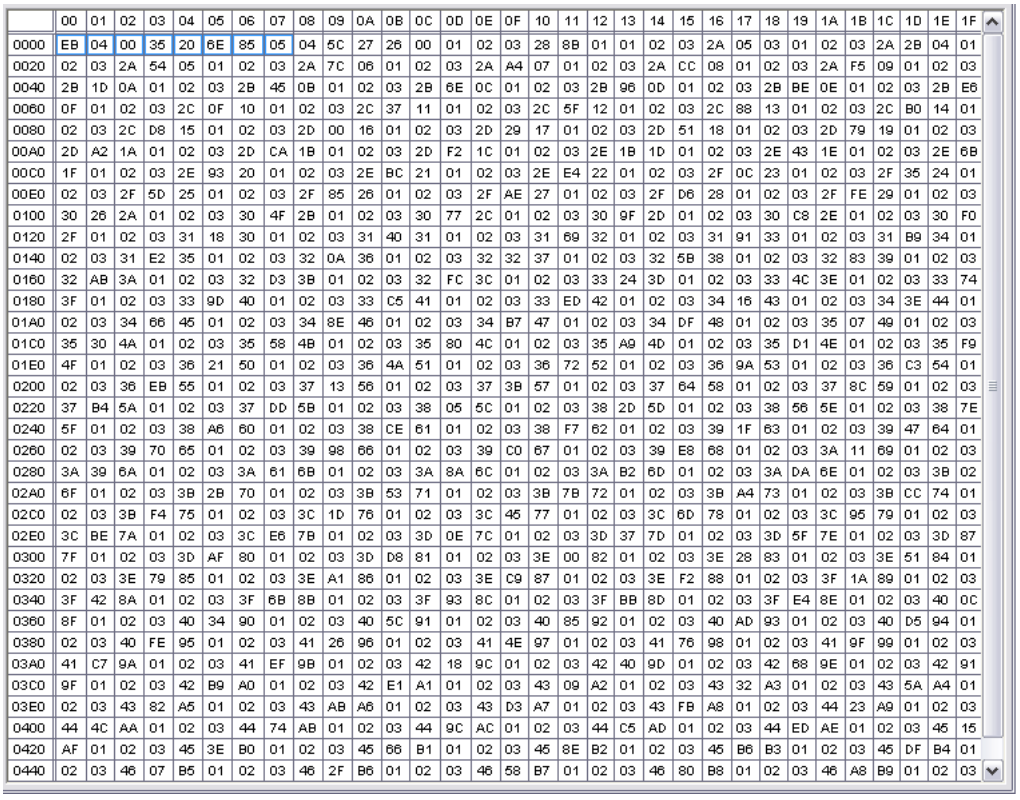
4.3 Visualização de dados

A visualização dos diversos tipos de dados gerados durante os ciclos de teste é uma funcionalidade importante, pois ela serve como uma boa ferramenta de análise.

O projeto de satélites científicos necessita expor seus riscos em dois aspectos principais: o tecnológico e o científico. Portanto, o suporte a facilidades de análise de dados acopladas ao ferramental de testes tem bom potencial para revelar possíveis problemas em fases iniciais do projeto, reduzindo, assim, o custo das adequações.

Antes da construção dos módulos SPAC era possível visualizar apenas os dados em seu formato "bruto", tal como ele está codificado no pacote de dados para transmissão (*Network Byte Order*), como apresentado na Figura 4.5, tornando lenta a validação do aspecto científico da IST.

Integrando o conjunto de módulos SPAC, no contexto do projeto QSEE, foram produzidos módulos para visualização de dados provenientes do SWPDC que incluem: dados provenientes das câmeras de raios-X (científico, diagnóstico e teste) e dados de *housekeeping*, conforme mensionado na Seção 3.1. A Figura 4.6 mostra a interface do visualizador de dados de *housekeeping*. Por meio dele, é possível avaliar a evolução do estado do SWPDC fazendo uso dos gráficos que mostram a evolução da coleta de temperaturas dos dois ter-



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
0000	EB	04	00	35	20	6E	85	05	04	5C	27	26	00	01	02	03	28	8B	01	01	02	03	2A	05	03	01	02	03	2A	2B	04	01
0020	02	03	2A	54	05	01	02	03	2A	7C	06	01	02	03	2A	A4	07	01	02	03	2A	CC	08	01	02	03	2A	F5	09	01	02	03
0040	2B	1D	0A	01	02	03	2B	45	0B	01	02	03	2B	6E	0C	01	02	03	2B	96	0D	01	02	03	2B	BE	0E	01	02	03	2B	E6
0060	0F	01	02	03	2C	0F	10	01	02	03	2C	37	11	01	02	03	2C	5F	12	01	02	03	2C	88	13	01	02	03	2C	B0	14	01
0080	02	03	2C	D8	15	01	02	03	2D	00	16	01	02	03	2D	29	17	01	02	03	2D	51	18	01	02	03	2D	79	19	01	02	03
00A0	2D	A2	1A	01	02	03	2D	CA	1B	01	02	03	2D	F2	1C	01	02	03	2E	1B	1D	01	02	03	2E	43	1E	01	02	03	2E	6B
00C0	1F	01	02	03	2E	93	20	01	02	03	2E	BC	21	01	02	03	2E	E4	22	01	02	03	2F	0C	23	01	02	03	2F	35	24	01
00E0	02	03	2F	5D	25	01	02	03	2F	85	26	01	02	03	2F	AE	27	01	02	03	2F	D6	28	01	02	03	2F	FE	29	01	02	03
0100	30	26	2A	01	02	03	30	4F	2B	01	02	03	30	77	2C	01	02	03	30	9F	2D	01	02	03	30	C8	2E	01	02	03	30	F0
0120	2F	01	02	03	31	18	30	01	02	03	31	40	31	01	02	03	31	69	32	01	02	03	31	91	33	01	02	03	31	B9	34	01
0140	02	03	31	E2	35	01	02	03	32	0A	36	01	02	03	32	32	37	01	02	03	32	5B	38	01	02	03	32	83	39	01	02	03
0160	32	AB	3A	01	02	03	32	D3	3B	01	02	03	32	FC	3C	01	02	03	33	24	3D	01	02	03	33	4C	3E	01	02	03	33	74
0180	3F	01	02	03	33	9D	40	01	02	03	33	C5	41	01	02	03	33	ED	42	01	02	03	34	16	43	01	02	03	34	3E	44	01
01A0	02	03	34	66	45	01	02	03	34	8E	46	01	02	03	34	B7	47	01	02	03	34	DF	48	01	02	03	35	07	49	01	02	03
01C0	35	30	4A	01	02	03	35	58	4B	01	02	03	35	80	4C	01	02	03	35	A9	4D	01	02	03	35	D1	4E	01	02	03	35	F9
01E0	4F	01	02	03	36	21	50	01	02	03	36	4A	51	01	02	03	36	72	52	01	02	03	36	9A	53	01	02	03	36	C3	54	01
0200	02	03	36	EB	55	01	02	03	37	13	56	01	02	03	37	3B	57	01	02	03	37	64	58	01	02	03	37	8C	59	01	02	03
0220	37	B4	5A	01	02	03	37	DD	5B	01	02	03	38	05	5C	01	02	03	38	2D	5D	01	02	03	38	56	5E	01	02	03	38	7E
0240	5F	01	02	03	38	A6	60	01	02	03	38	CE	61	01	02	03	38	F7	62	01	02	03	39	1F	63	01	02	03	39	47	64	01
0260	02	03	39	70	65	01	02	03	39	98	66	01	02	03	39	C0	67	01	02	03	39	E8	68	01	02	03	3A	11	69	01	02	03
0280	3A	39	6A	01	02	03	3A	61	6B	01	02	03	3A	8A	6C	01	02	03	3A	B2	6D	01	02	03	3A	DA	6E	01	02	03	3B	02
02A0	6F	01	02	03	3B	2B	70	01	02	03	3B	53	71	01	02	03	3B	7B	72	01	02	03	3B	A4	73	01	02	03	3B	CC	74	01
02C0	02	03	3B	F4	75	01	02	03	3C	1D	76	01	02	03	3C	45	77	01	02	03	3C	6D	78	01	02	03	3C	95	79	01	02	03
02E0	3C	BE	7A	01	02	03	3C	E6	7B	01	02	03	3D	0E	7C	01	02	03	3D	37	7D	01	02	03	3D	5F	7E	01	02	03	3D	87
0300	7F	01	02	03	3D	AF	80	01	02	03	3D	D8	81	01	02	03	3E	00	82	01	02	03	3E	28	83	01	02	03	3E	51	84	01
0320	02	03	3E	79	85	01	02	03	3E	A1	86	01	02	03	3E	C9	87	01	02	03	3E	F2	88	01	02	03	3F	1A	89	01	02	03
0340	3F	42	8A	01	02	03	3F	6B	8B	01	02	03	3F	93	8C	01	02	03	3F	BB	8D	01	02	03	3F	E4	8E	01	02	03	40	0C
0360	8F	01	02	03	40	34	90	01	02	03	40	5C	91	01	02	03	40	85	92	01	02	03	40	AD	93	01	02	03	40	D5	94	01
0380	02	03	40	FE	95	01	02	03	41	26	96	01	02	03	41	4E	97	01	02	03	41	76	98	01	02	03	41	9F	99	01	02	03
03A0	41	C7	9A	01	02	03	41	EF	9B	01	02	03	42	18	9C	01	02	03	42	40	9D	01	02	03	42	68	9E	01	02	03	42	91
03C0	9F	01	02	03	42	B9	A0	01	02	03	42	E1	A1	01	02	03	43	09	A2	01	02	03	43	32	A3	01	02	03	43	5A	A4	01
03E0	02	03	43	82	A5	01	02	03	43	AB	A6	01	02	03	43	D3	A7	01	02	03	43	FB	A8	01	02	03	44	23	A9	01	02	03
0400	44	4C	AA	01	02	03	44	74	AB	01	02	03	44	9C	AC	01	02	03	44	C5	AD	01	02	03	44	ED	AE	01	02	03	45	15
0420	AF	01	02	03	45	3E	B0	01	02	03	45	66	B1	01	02	03	45	8E	B2	01	02	03	45	B6	B3	01	02	03	45	DF	B4	01
0440	02	03	46	07	B5	01	02	03	46	2F	B6	01	02	03	46	58	B7	01	02	03	46	80	B8	01	02	03	46	A8	B9	01	02	03

Figura 4.5 - Visualização em hexadecimal de uma mensagem de dados científicos codificados em NBE.

mistores (ao topo da interface), os históricos dos relatos de eventos gerados pelo SWPDC, contadores de erro de memória e processador, estado dos imageadores HXI-1 e HXI-2, entre outros (SILVA et al., 2007).

Note-se que os controles para filtragem dos dados estão situados na parte mais inferior da tela. Este módulo utiliza as facilidades providas pelo módulo de extração de dados, descrito na Seção 4.2, para obter os dados diretamente do registro de atividade da execução dos testes. Sua responsabilidade envolve o processamento e visualização num formato adequado.

Outro módulo importante é o visualizador de dados adquiridos pelos imageadores, cuja interface principal está ilustrada na Figura 4.7. O objetivo deste módulo é decompor os pacotes dados científicos e calcular medidas de qualidade da aquisição de dados feita pela IST. Com ele é possível verificar eventuais problemas de perdas pacotes, por meio da análise das diferenças nas marcas de tempo entre cada pacote.

Além disso, os dados adquiridos podem ser visualizados na forma de histograma, o que facilita a comparação do espectro adquirido pela IST com aquele introduzido durante a execução dos testes (por meio de simulação ou por fontes reais de raios-X). Um exemplo

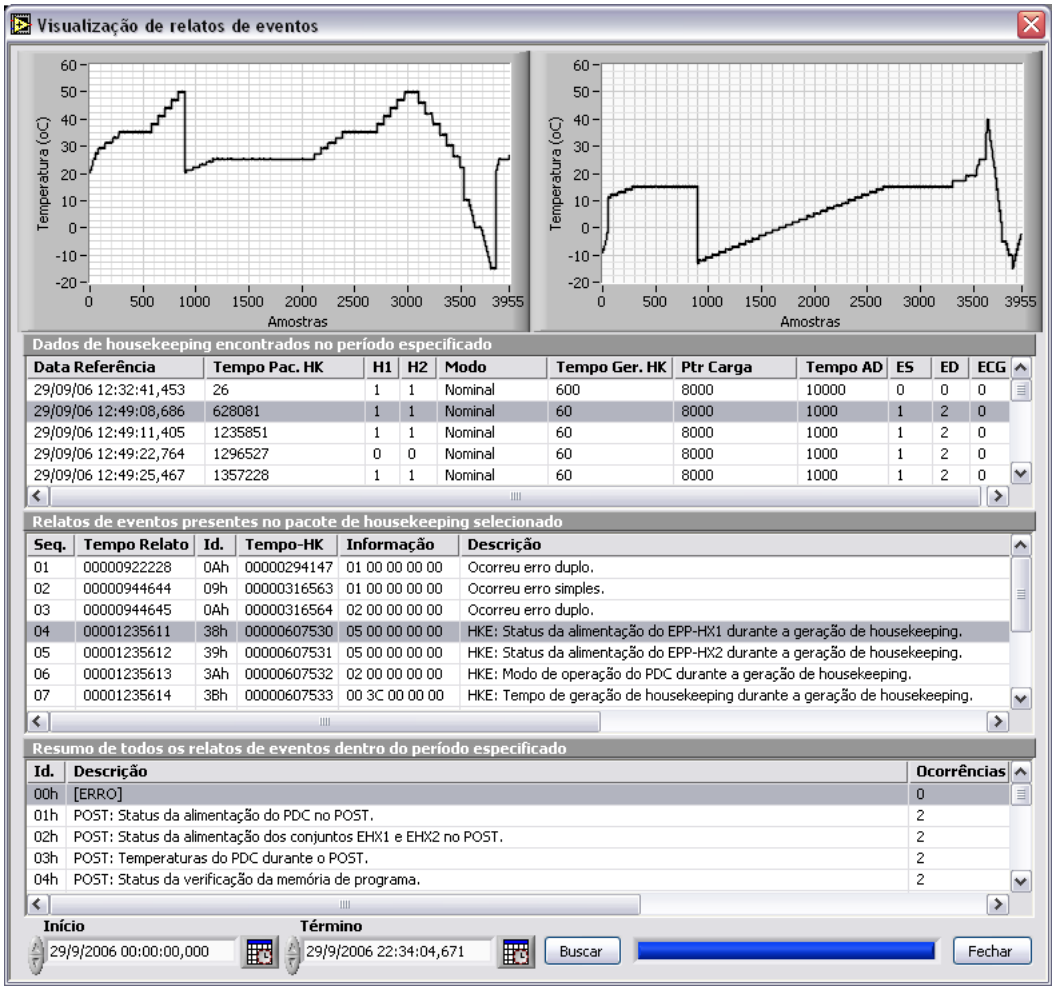


Figura 4.6 - Visualizador de dados de *housekeeping*.

desta visualização está apresentada na Figura 4.8, ilustrando o que foi adquirido durante um teste de aquisição de dados científicos do SWPDC, mediante o uso de uma fonte de raios-X simulada da Nebulosa da Constelação do Caranguejo, na faixa de 0 a $225KeV$.

Por meio do histograma, o testador poderia inferir, por exemplo, que apesar das perdas de alguns pacotes de dados, foi verificado que os dados simulados corresponderam ao que foi adquirido pelo SWPDC, dentro de limites aceitáveis de qualidade como, por exemplo, se a quantidade de amostras adquiridas numa certa região deste espectro é a esperada para um dado ciclo de teste. A visualização dos dados científicos na forma de histograma representa um valor agregado especial para o ferramental de testes, uma vez que ele é extremamente útil para identificar discrepâncias entre os dados de entrada simulados e aqueles efetivamente adquiridos pelo software em teste. Isto, portanto, ajuda a alinhar os focos científico e técnológicos do projeto do software.

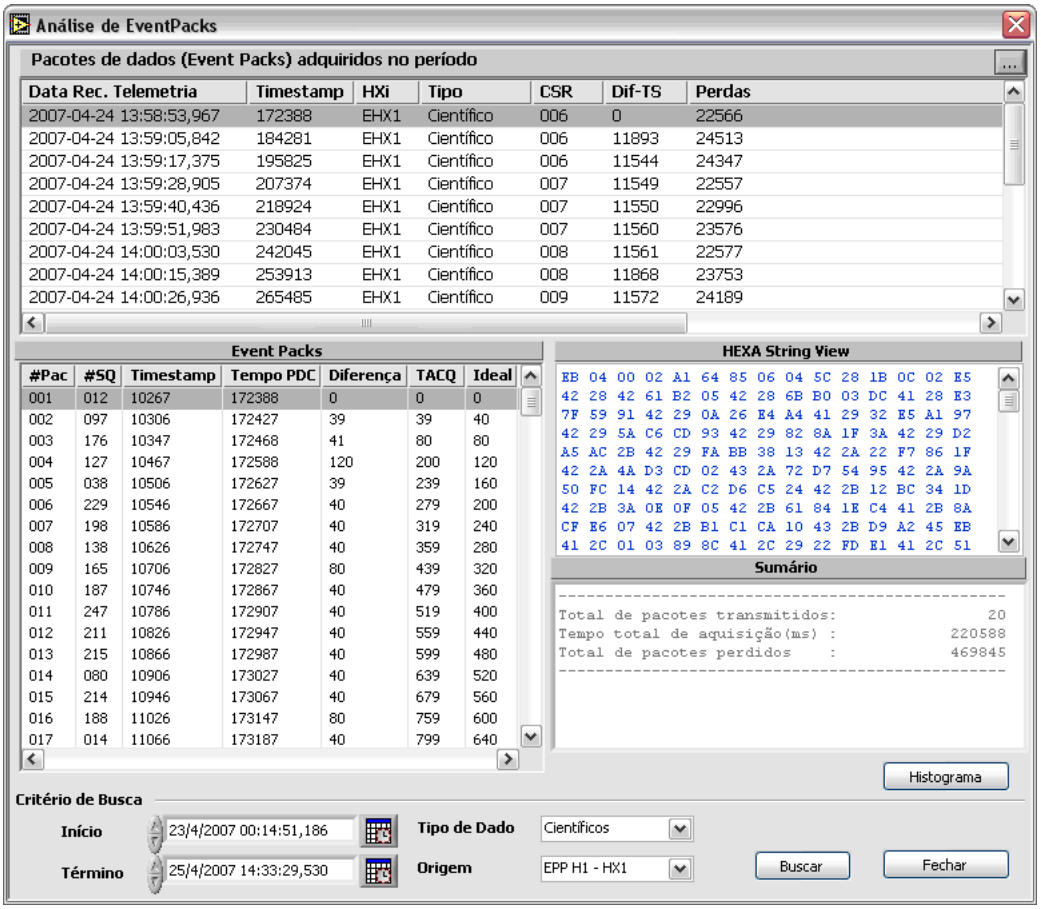


Figura 4.7 - Visualizador de dados de adquiridos dos EPPs.

4.4 Integração com placas de aquisição de dados (DAQ)

O teste de software embarcado geralmente requer a integração do ferramental de teste (*test harness*) com o hardware alvo da IST. Um tipo de integração comum é aquele provido por placas de Aquisição de Dados (DAQ) que permitem operações diversas com sinais analógicos e digitais. Tais características complementam o ambiente de teste, possibilitando simular condições de funcionamento do componente em teste (hardware e software embarcado) muito próximas ou até idênticas aos modelos de hardware de engenharia, de qualificação e de vôo. Portanto, o suporte a placas DAQ é importante para a integração adequada do software de execução do teste à IST. Na tentativa de suprir esta necessidade, módulos SPAC foram concebidos para interagir com placas DAQ, usando a interface de software fornecedida pelo próprio fabricante deste equipamento.

As interfaces principais para aquisição de dados estão apresentadas nas Figuras 4.9 e 4.10. A primeira permite criar ondas quadradas em qualquer porta digital disponível na placa DAQ, inclusive em mais de uma porta simultaneamente, bastando especificar a duração

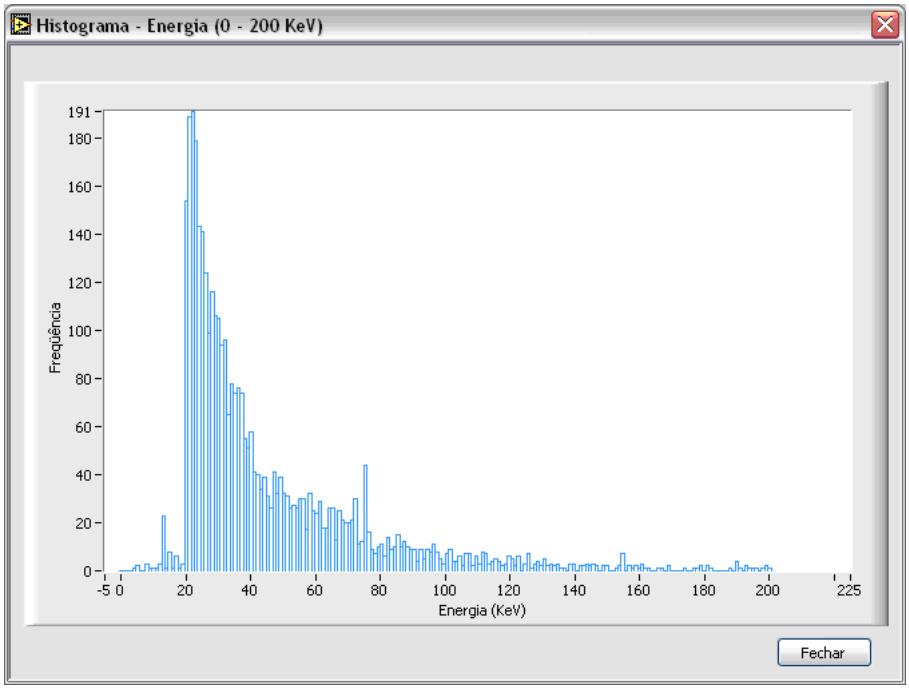


Figura 4.8 - Visualizador de dados na forma de histograma.

do pulso (em milissegundos) e a quantidade de oscilações. Isto pode ser útil, por exemplo, para estimular pinos de interrupções no hardware, sinais de *clock*, ou qualquer outro tipo de sinal ON/OFF. A segunda permite variar tensões em portas analógicas. Isto permite, por exemplo, simular automaticamente variações de temperatura, aplicando as tensões geradas nos canais analógicos apropriados do sistema em teste.

Sabe-se que a relação entre tensão e temperatura vai depender da curva característica do termistor que se deseja simular. Por isso, tanto as tensões quanto a duração de aplicação de cada tensão nas respectivas portas são editáveis, permitindo facilmente o reuso desta funcionalidade.

A versão atual deste módulo oferece suporte à programação de sinais analógicos e digitais apenas para a série DT9800 da *Data Translations, Inc.* (DATA-TRANSLATIONS, 2006).

4.5 Considerações finais

Este capítulo apresentou os módulos SPAC, detalhando o funcionamento do mecanismo de extração de dados e características principais que auxiliam na validação de requisitos não funcionais da IST. Foram apresentadas interfaces que permitem a visualização de diversos tipos de dados definidos pelo SWPDC. A SPAC tem sido usada para validar um software embarcado em instrumento científico como posto na Seção 3.1. As funcionalidades específicas da SPAC apresentadas neste capítulo incluem: a configuração de registro de

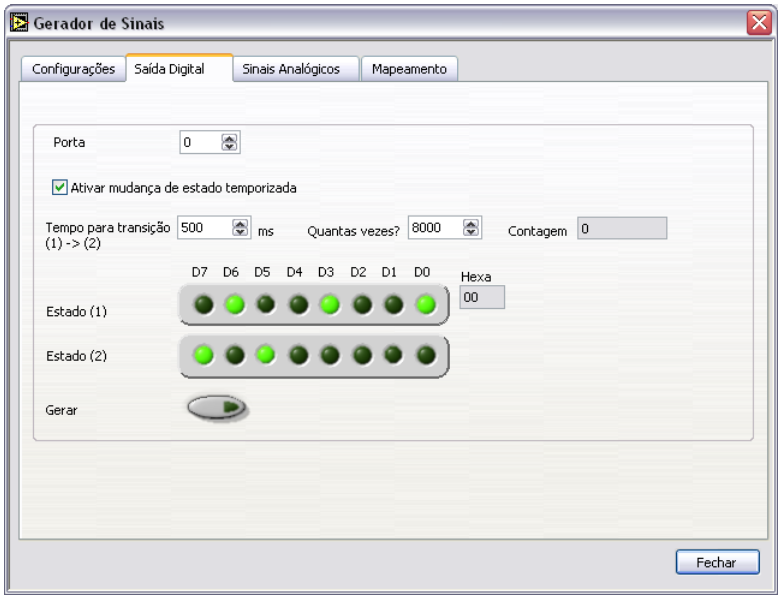


Figura 4.9 - Geração de sinais digitais.

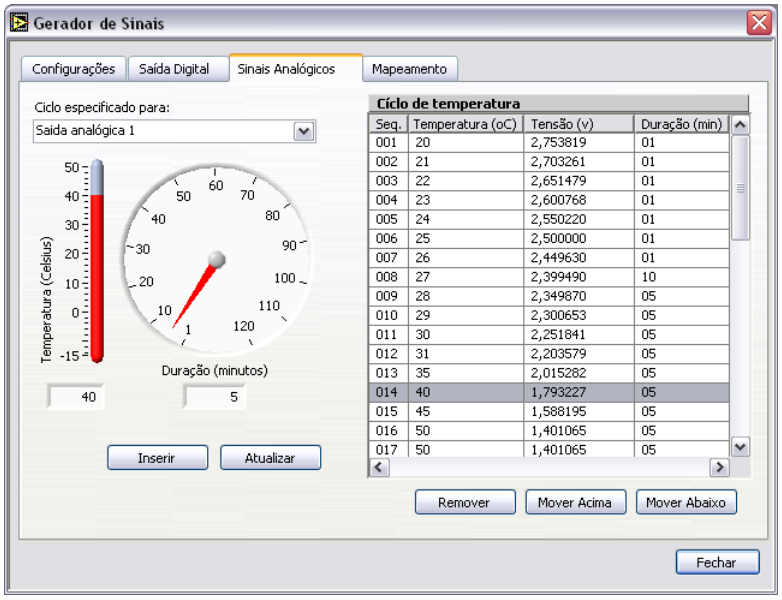


Figura 4.10 - Geração de sinais analógicos.

cada módulo baseada em arquivos descritores textuais, manipulação de canais digitais e analógicos, recuperação dos resultados da execução dos casos de testes, extração de dados de acordo com o tipo de informação (dados científicos, de diagnóstico, de teste, de descarga de memória e de housekeeping), visualização de dados científicos, por meio de espectros de energia, e visualização de temperaturas e relatos de eventos (logs).

Baseado num esquema simples de carregamento de módulos externos, os módulos SPAC

são fracamente acoplados a QSEE-TAS possibilitando sua extensão bastando, para isto, registrar os módulos para que estes possam ser invocados a partir da interface principal do projeto de teste. Embora a SPAC seja específica para o SWPDC, outros módulos podem ser desenvolvidos para atender às especificidades de outras ISTs, sem impacto na elaboração do projeto de teste como um todo. É o caso do suporte à integração de placas DAQ que amplia a capacidade de interação com o hardware, importante no teste software embarcado.

5 ESTUDO COMPARATIVO DE CUSTO

Este capítulo apresenta uma experiência de uso da ferramenta QSEE-TAS/SPAC no processo de teste dos simuladores dos instrumentos APEX, IONEX e EPP do software embarcado SWPDC, todos desenvolvidos no contexto do projeto QSEE (SANTIAGO et al., 2007). O objetivo deste estudo foi a medição e posterior comparação do tempo empreendido na execução de casos de testes manualmente e o tempo da execução automatizada assistida pela ferramenta QSEE-TAS/SPAC.

Apesar de o resultado parecer óbvio em favor da execução automatizada, nem sempre resultados significativo foram observados. Para este estudo de caso específico, a experiência apontou as situações em que o uso da ferramenta foi realmente significativo: no teste de regressão usando todos os casos de teste. Uma breve introdução sobre testes de regressão é apresentada na Seção 2.5.

Este capítulo está organizado da seguinte forma: a Seção 5.1 apresenta as IST usadas no estudo de caso e o setup do ambiente de teste; a Seção 5.2 relata como o experimento foi conduzido e quais foram as métricas utilizada; a Seção 5.3 são apresentados os resultados obtidos; por fim, a Seção 2.7 apresenta alguns trabalhos relacionados. Este estudo foi publicado e apresentado em (SANTIAGO et al., 2008).

5.1 As configurações de teste usadas no estudo

O ambiente de teste foi montado para o teste em bancada envolvendo três IST em dois cenários distintos. Os três simuladores alvos do teste estão descritos como se segue:

- **APEX** (*Alpha, Protons and Eletron Flux Monitoring Experiment*): é um experimento científico para monitorar o fluxo de partículas subatômicas na altitude da magnetosfera interna;
- **IONEX** (*Ionospheric Experiment*) é um experimento científico para monitoramento de irregulares e bolhas de plasma na ionosfera;
- **EPP** (*Event Pre-Processor*) é o pré-processador de eventos de incidência de fótons de raios X do satélite MIRAX (Monitor e Imageador de Raios X).

O primeiro cenário (Figura 5.1) envolve a execução manual dos casos de teste, assistida apenas pelo ferramental de teste construído especificamente para cada IST. O testador deve, então, estimular as IST usando tais interfaces - de acordo com plano de teste -, verificar se o resultado recebido está de acordo com o esperado, e anotar o veredito num arquivo textual, ou seja, o relato de teste.

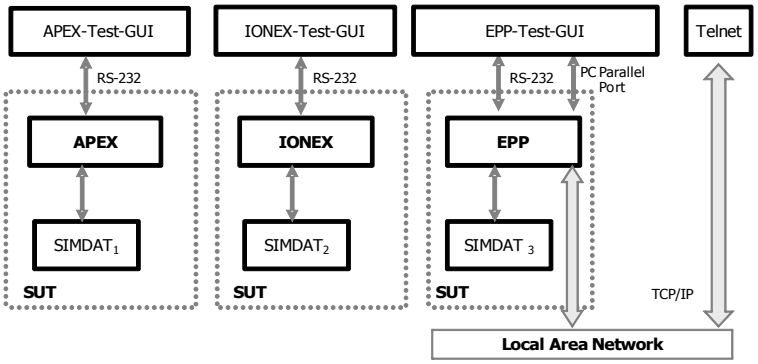


Figura 5.1 - Ambiente de teste para o primeiro cenário.

O segundo cenário (Figura 5.2) envolve a execução dos casos de teste com o auxílio da QSEE-TAS, seguindo o fluxo de trabalho proposto - descrito na Seção 3.3.

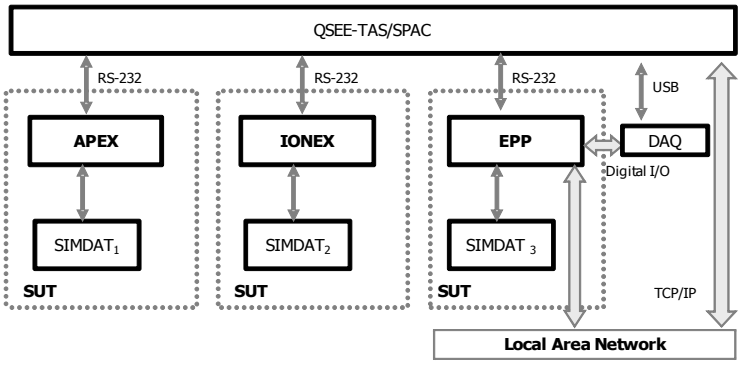


Figura 5.2 - Ambiente de teste para o segundo cenário.

Todos os simuladores imitam o comportamento do hardware real do ponto de vista da coleta de dados e cada um o faz seguindo especificações e protocolos de comunicação diferentes.

5.2 Metodologia

Alguns atributos foram obtidos para elucidar características das IST e da *test suite*. Dos simuladores, destacou-se o tamanho em linhas de código (LOC) e a linguagem de implementação. O tamanho da *test suite* envolveu a quantidade de casos de testes executados (CTE) e o maior número de passos de teste (MaxPTE) encontrado por caso de teste (CHANSON et al., 1990). A Tabela 5.1 resume esses dados.

Os testes foram executados em momentos separados para cada cenário. Isso foi feito na tentativa de isolar e resolver problemas técnicos no ambiente de teste que eventualmente

Tabela 5.1 - Casos de teste e dados das IST.

IST	CTE	MaxPTE	LOC	Linguagem
APEX	22	105	3028	Java e C/C++
IONEX	24	21	1362	C
EPP	16	10	1813	C/C+++

pudessem ocorrer num cenário e contaminasse o outro. Cada cenário foi executado duas vezes, cada vez em uma versão diferente das IST a fim de caracterizar a segunda execução como um teste de regressão tomando-se todos os casos de teste. A medição do tempo foi executada manualmente por meio de cronômetro manual simples nos dois cenários, mesmo que na execução automatizada a coleta do tempo pudesse ser inferida pelo relato de teste automatizado (registro de eventos). Além disso, esperou-se que as mesmas falhas reveladas pelos testes executados manualmente fossem reveladas pela execução automatizada.

5.3 Avaliação dos resultados

Para mostrar a utilidade da QSEE-TAS em termos de custo em comparação a Execução Testes Manuais (ETM) - sem auxilio automatizada, é necessário definir o significado de custo. A medição do custo do teste num contexto experimental como este não é tão direta. Geralmente, o tamanho do conjunto de teste (*Test Set Size* - TSS) tem sido adotado como uma medida razoável, baseanda na suposição de que o custo é proporcional a esse tamanho. Assim a TSS que pode ser obtida pela simples contagem do número de casos de teste no conjunto de teste (BRIAND et al., 2004). Entretanto, casos de teste distintos possuem diferentes quantidades de passos de teste. Então, uma outra opção seria contar a quantidade de passos de teste em todo o conjunto.

Portanto, considerou-se o custo como sendo a quantidade de tempo despendida para executar todo o conjunto de teste usando uma ou outra ferramenta. Em outras palavras, dado o mesmo conjunto de teste, quanto tempo será gasto para executar, analisar e apontar o veredito de todos os casos de teste usando-se a QSEE-TAS e a ETM? Assim, quanto menor o tempo de execução do teste menor será o custo.

Após a execução dos testes nos dois cenários, chegou-se aos resultados apresentados no gráfico da Figura 5.3.

A comparação do primeiro cenário envolvendo a IST APEX mostra que a execução manual excedeu em aproximadamente um quarto de hora a execução automatizada. Esse tempo, em grande parte se deu em função das atividades de preparação extra exigidas pela ferramenta. Entretanto, no teste de regressão a execução automatizada obteve melhor

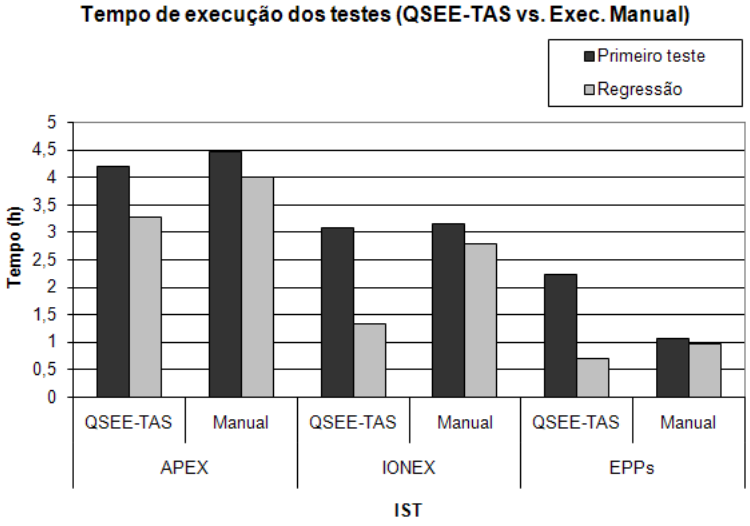


Figura 5.3 - Resultado da execução dos testes.

resultado, reduzindo em 45 minutos (12,5%) o tempo do teste em relação ao teste manual. Observou-se que esse tempo foi obtido, em grande parte, porque não havia a necessidade de mudar de interface a todo instante, como acontece na execução manual (ora tela de teste, ora tela do editor de textos). E isso acabou sendo uma constante nos demais testes. Outro fator que reduziu o tempo, nesta IST em particular, foi a programação prévia dos complexos procedimentos de calibração do APEX que, quando executados manualmente, exigem muita intervenção do testador.

No caso do IONEX, o tempo do primeiro teste foi praticamente o mesmo, com leve perda de tempo no caso da execução automatizada. Entretanto, o teste de regressão revela uma economia de aproximadamente 3 horas a favor da execução automatizada.

No caso do EPP, embora tanto a IST quanto a *suite* de testes fosse menor que em relação às duas primeiras, seu procedimento de preparação inicial é mais complexo, pois exige-se a presença de dois canais de comunicação, cada um com um protocolo específico: RS-232 e TCP/IP. O primeiro canal é a interface de coleta de dados e o segundo oferece uma interface de linha de comandos (enviados protocolo Telnet) para interferir no comportamento da IST para testar suas capacidades de simulação de falhas.

Com base nestas explanações, os benefícios são claros em favor do uso da QSEE-TAS, especialmente no teste de regressão. Precisamente, a redução de custo observada foi de 18,7% no APEX 27,8% no EPP e 52.5% no IONEX na primeira execução. Além disso, comparando-se os resultados entre a primeira execução e o teste de regressão a redução de custo mostra-se ainda maior: 22,1% no APEX, 56.8% no IONEX e 68,6% no EPP.

5.4 Considerações finais

Em resumo, o teste de regressão consumindo todos os casos de teste em relação a primeira execução, obteve melhor resultado se executado com o auxilio da ferramenta. Uma vez que o teste de regressão é executado muitas vezes durante o ciclo de desenvolvimento da maioria dos projetos de software, a ferramenta pode economizar significativamente o tempo gasto nesta atividade, com a mesma probabilidade de encontrar falhas. Isso compensaria o investimento nas preparações extras.

6 CONCLUSÃO

Este capítulo apresenta um resumo sobre as atividades do curso de mestrado que levaram a produção deste trabalho. Além disso, são descritos os resultados atingidos e as dificuldades encontradas estruturada da seguinte forma: a Seção 6.1 os objetivos atingidos e os resultados alcançados; as dificuldades na realização deste trabalho até agora são citadas na Seção 6.2; as limitações da versão atual da QSEE-TAS/SPAC estão descritas na Sessão 6.3, e; por fim, a Seção 6.4 apresenta as conclusões finais e perspectivas sobre trabalhos futuros.

6.1 Contribuições do trabalho

O desenvolvimento deste trabalho teve os seguintes resultados:

- desenvolvimento de uma solução de software para auxílio ao planejamento, execução automatizada de testes funcionais e geração de documentação relacionada ao processo de teste para software embarcado em computadores de instrumentos de satélites científicos e balões estratosféricos;
- agregação de valor para atividade de execução de testes funcionais, com base em protocolos de comunicação, uma vez que mais ganhos de produtividade foram verificados;
- uma arquitetura de software aberta a integração com outras ferramentas (geração automática de casos de teste, por exemplo);
- aumento do reuso dos casos de teste na execução dos testes de regressão, obtidos no contexto do projeto QSEE (SANTIAGO et al., 2007).
- desenvolvimento de uma linguagem formal para especificação de testes com base na definição das mensagens do procolo de comunicação com o software embarcado.

6.2 Dificuldades encontradas

Durante o desenvolvimento deste trabalho algumas dificuldades foram encontradas. Dentre elas, enumeram-se as principais:

- A curva de aprendizagem para produzir programas não triviais com o LabVIEW representou uma dificuldade. Entretanto, a inércia na aprendizagem da linguajem G, base para o LabVIEW, foi ultrapassada com a prática.

- Embora o LabVIEW possua componentes para integração com banco de dados relacionais, eles estão disponíveis por licenciamento extra, não contemplado na época do desenvolvimento deste trabalho. Hoje, a integração da QSEE-TAS com banco dados relacionais tem se tornado um fator chave para a disponibilização dos projetos de teste de forma centralizada e segura.

6.3 Limitações e aperfeiçoamentos

Existem algumas limitações da QSEE-TAS/SPAC que merecem ser citadas e são foco de aperfeiçoamentos. Além comparação das capacidades desejáveis para um *test harness* e aquelas implementadas pela QSEE-TAS apresentados na Seção 3.6, cita-se especificamente as seguintes limitações:

- Projetos de teste maiores que 80 MB reduzem drasticamente o desempenho da ferramenta em função do consumo excessivo de memória. Neste caso, recomenda-se dividir os casos de teste em arquivos de projetos de teste menores. Uma otimização seria a carga de dados sob demanda (*lazy loading*);

- A precisão dos *timers* para sincronizar os passos de teste é teoricamente de 1 ms (em ambiente Windows). Entretanto, foi verificado que este tempo pode ser 20 ms ou mais dependendo do tamanho da mensagem a ser enviada para a IST e da carga do sistema hospedeiro da QSEE-TAS. Neste caso, recomenda-se executar o processo da QSEE-TAS com prioridade mais alta; Um possível otimização seria usar os mecanismos de *loops* temporizados do $LabVIEW^{\mathrm{TM}}$(*timed loops*) para encapsular o processo das mensagens;

- O canal do tipo TCP/IP pressupõe que a IST faz o papel de servidor (modo *listening*), conectando-se a IST ao início da execução dos testes e fechando a conexão somente ao final da execução. Isto deixa o canal de comunicação sujeito a eventos ainda não controláveis pela ferramenta no TCP/IP;

- Se a quantidade de trocadas entre a QSEE-TAS e a IST for maior que, aproximadamente, 20 MB numa única sessão de execução de teste, haverá problemas de desempenho para visualizar ao manipular histórico de execução; isto vai requerer a implementação de técnicas de paginação sob demanda em futuras versões, para reduzir o consumo de memória para históricos de teste muito longos;

- Suporte apenas para *scripts* de teste lineares - passo-a-passo, sem laços condições;

- A extensão da QSEE-TAS por meio de módulos externos restringe-se a módulos feitos somente em *LabVIEW*™. Módulos feitos com outras linguagens não são aceitos, a menos que o acesso seja feito por meio de um encapsulador (*wrapper*) construído com *LabVIEW*™.

6.4 Considerações finais e trabalhos futuros

As funcionalidades contempladas na versão atual da QSEE-TAS/SPAC têm aumentado sensivelmente o grau de automação do processo de teste de aplicações embarcadas, facilitando a análise do comportamento da IST. Por meio dela, a IST pode ser imersa num ambiente em que é permitido checar requisitos funcionais como, por exemplo, a sintaxe dos comandos do protocolo de comunicação, até requisitos não funcionais, como robustez e confiabilidade, por meio da extração e desempacotamento dos dados com visualizações apropriadas. Tais funcionalidades podem apoiar a execução de teste de software embarcado em dispositivos com requisitos similares aos encontrados no domínio de aplicações espaciais.

Os módulos SPAC têm se mostrado especialmente úteis para eliminar tarefas repetitivas e propensas a erros, como a verificação de requisitos de desempenho da IST, na qual a análise de perda de dados é feita baseada nos registros de resposta aos diversos estímulos enviados a IST num dado intervalo de tempo. O testador busca e visualiza, por exemplo, eventuais falhas de seqüenciamento nas respostas ou na temporização. A visualização dos dados científicos tem ajudado a validar o aspecto científico do software usado no estudo de caso, o que é importante do ponto de vista do cientista.

Além disso, a ferramenta poderá oferecer subsídio para aderência a atividades em processos de melhoria da qualidade de software. Por exemplo, em versões futuras da ferramenta, o item de teste poderá ser associado a um ou mais requisitos do software, descritos no documento de especificação de requisitos da IST. Tal característica será útil para a geração de matrizes de rastreabilidade entre Itens de Teste e Requisito da IST, o que é especialmente útil nas análises de impacto de alterações de requisitos na IST frente aos casos de teste já cadastrados. Para trabalhos futuros, são propostos os seguintes tópicos:

- Projeto de teste mantido diretamente no SGDB;
- Integração com ferramentas de geração automática de casos teste;
- Melhoramentos na rastreabilidade entre itens de teste e casos de teste;
- Suporte para outros tipos de placas de aquisição de dados (módulos da SPAC);

- Geração e execução de scripts de teste externos (construídos em com outras linguagens);
- Exposição de funcionalidades da ferramenta na forma de API, acessível externamente via ActiveX, por exemplo;
- Construção de novos *scripts* XSL para gerar visões de relatórios diferentes, dependendo da escolha do usuário, entre outros.
- Utilização do banco de dados gerado sobre os testes para efetuar análises de confiabilidades das IST ao longo do tempo (YAMADA et al., 1986)

Em geral, o objetivo de desenvolver uma ferramenta de software capaz de automatizar a execução dos testes, possibilitando acompanhar a evolução dos testes foi alcançado. A QSEE-TAS/SPAC tem sido continuamente usada e melhorada no contexto dos projetos realizados na CEA em que existe demanda para atividades de VV&T. Apesar das limitações, ela pode ser usada para apoiar a execução de testes de forma sistemática em software embarcado que suporte os tipos de canais de comunicação disponíveis pela ferramenta e cujas mensagens de comunicação possam ser codificadas segundo a linguagem LDM.

REFERÊNCIAS BIBLIOGRÁFICAS

ADRION, W. R.; BRANSTAD, M. A.; CHERNIAVSKY, J. C. Validation, verification, and testing of computer software. **ACM Computing Surveys (CSUR)**, ACM Press, New York, NY, v. 14, n. 2, p. 159–192, 1982. ISSN 0360-0300. 29, 30

AMBROSIO, A. M. **COFI: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais**. 209 p. Tese (Doutorado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2005–07–01 2005. Disponível em: <http://urlib.net/sid.inpe.br/MTC-m13@80/2005/09.06.13.34>. 25

AMBROSIO, A. M.; CARVALHO, S. V. d.; VIJAYKUMAR, N. L.; MARTINS, E. Automatic test case generation of the behavior of communication software systems. In: CARVALHO, S. V. d.; VIJAYKUMAR, N. L.; MARTINS, E. (Ed.). **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais, 2002. Disponível em: <http://urlib.net/lac.inpe.br/lucio/2002/12.02.08.09>. 35

AMBROSIO, A. M.; MARTINS, E.; SILVA, C. S.; VIJAYKUMAR, N. L. On the use of test standardization in space communication applications. In: INTERNATIONAL CONGRESS OF SPACE OPERATIONS (SpaceOps2004), 8., 2004, Montreal, Canada. **Proceedings...** Montreal, Canada: Canadian Space Agency (CSA), 2004. p. 1–10. 40

AMBROSIO, A. M.; MARTINS, E.; VIJAYKUMAR, N. L.; CARVALHO, S. V. Systematic generation of test and fault cases for space application validation. In: ESA DATA SYSTEM IN AEROSPACE (DASIA), 9., 2005, Edinburgh, Scotland. Noordwijk. **Proceedings...** [S.l.]: ESA Publications, 2005. 36

ARANTES, A. O.; VIJAYKUMAR, N. L.; SANTIAGO, V. A.; CARVALHO, A. R. Automatic test case generation through a collaborative web application. In: INTERNET AND MULTIMEDIA SYSTEMS AND APPLICATIONS (EUROIMSA), 1., 2008, Innsbruck, Austria. **Proceedings...** Cambridge, Massachusetts, USA: ACTA Press, 2008. p. 612–616. 36

BEIZER, B. **Software testing techiques**. 2. ed. Boston, MA, USA: International Thomson Computer Press, 1990. 550 p. 29, 31, 33, 38, 65

BINDER, R. V. **Testing object-oriented systems**: models, patterns and tools. Upper Saddle River, NJ, USA: Pearson Education, 2000. 1191 p. 37, 61, 64, 65, 66

BOEHM, B. W. **Software engineering economics**. USA: Prentice Hall, 1981. 33

BRAGA, J.; ROTHSCHILD, R.; HEISE, J.; STAUBERT, R.; REMILLARD, R.; D´AMICO, F.; JABLONSKI, F. J.; HEIND, W.; MATTESON, J.; KUULKERS, E.; WILMS, J.; KENDZIORRA, E. Mirax: a brazilian x-ray astronomy satellite mission. **Advances in Space Research**, v. 34, n. 12, p. 2657–2661, 2004. Disponível em: <http://urlib.net/sid.inpe.br/marciana/2004/12.03.15.04>. 46

BRIAND, L. C.; LABICHE, Y.; WANG, Y. Using simulation to empirically investigate test coverage criteria based on statechart. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 26., 2004, Washington, DC, USA. **Proceedings...** [S.l.]: IEEE Computer Society, 2004. p. 86–95. ISBN 0-7695-2163-0. 81

CHANSON, S. T.; LEE, B. P.; PARAKH, N. J.; ZENG, H.-X. Design and implementation of a ferry clip test system. In: THE IFIP WG6.1 INTERNATIONAL SYMPOSIUM ON PROTOCOL SPECIFICATION, TESTING AND VERIFICATION, 9., 1990, Amsterdam, The Netherlands. **Proceedings...** [S.l.]: North-Holland Publishing Co., 1990. p. 101–118. ISBN 0-444-88343-6. 39, 47, 80

CONFORMIQ-SOFTWARE-INC. **Conformiq tools for model-driven quality assurance**. 2008. Web. Disponível em: <http://www.conformiq.com/>. Acesso em: 01 out. 2008. 36

DATA-TRANSLATIONS. **USB data acquisition**. 2006. Disponível em: <http://www.datx.com/products/dataacquisition/usb/dt9810.asp>. Acesso em: 26 nov 2006. 75

DAVIS, W. B.; CHANSON, S.; FELD, G. N.; VUONG, S.; ITO, M. Architecture and design of a portable protocol tester. **IEEE Network Magazine**, v. 8, p. 24 – 31, 1991. 40

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. Porto Alegre, RS, BRA: ELSEVIER, 2007. 394 p. (Campus SBC). ISBN: 978-85-352-2634-8. 32, 40

ELBAUM, A. M. S.; ROTHERMEL, G. Incorporating varying test costs and fault severities into test case prioritization. In: THE INTERNAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'01), 23., 2001. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2001. 38

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). **Space engineering verification**: european space agency for the members of ecss publication, ECSS-E-10-02A. Noordwijk, The Netherlands, 1998. 144 p. 25, 33, 34, 35

GIESSLER, A.; BAUMGARTEN, B. **OSI conformance testing methodology and TTCN**. Amsterdam, NETHERLANDS: ELSEVIER, 1994. 328 p. 36

HETZEL, W. C. **The complete guide to software testing**. 2. ed. [S.l.]: QED Information Sciences, 1988. 280 p. 30

HOLZMANN, G. J. **Design and validation of computer protocols**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1991. ISBN 0-13-539925-4. 39, 55

HOWDEN, W. E. **Software engineering and technology**: functional program testing and analysis. New York, NY, USA: McGraw-Hill, 1987. 31, 37

IMAMURA, T.; MARUYAMA, H. Mapping between ASN.1 and XML. In: SYMPOSIUM ON APPLICATIONS AND THE INTERNET, 1., 2001, Japan. **Proccedings**. New York, NY, USA: IEEE Computer Society, 2001. 39

INFOTECH. **State of the art report**: software testing. England, UK, 1979. Vol. 1. 29

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). **IEEE 610.12 standard glossary of software engineering terminology**: an american national standard. New York, NY, USA, 1990. 48 p. 32

_____. **IEEE standard for software test documentation**: an american natational standard. New York, NY, nov. 1998. 48 p. 25, 65

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS. CIêNCIAS ESPACIAIS E ATMOSFéRICAS (INPE.CEA). **Projeto QSEE**: qualidade do software embarcado em aplicações espaciais. [S.l.], Nov. 2007. Disponível em: <`www.cea.inpe.br/qsee`>. Acesso em: 29 nov. 2007. 44, 45

INTERNATIONAL STANDARD ORGANIZATION (ISO). **ISO 8824-1 abstract syntax notation one (ASN.1)**: specification of basic notation. [S.l.], 2002. 39

KNIRK, W. D. Software testing process inprovements. In: INTERNATIONAL CONFERENCE ON TESTING COMPUTER SOFTWARE, 13., 1996, Washington, DC. **Proceedings...** Washington, DC, USA: USPDI, 1996. 64

LAST, M.; FRIEDMAN, M.; KANDEL, A. The data mining approach to automated software testing. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING SIGKDD, 9., 2003, Washington, DC, USA. **Proceedings...** New York, NY, USA: ACM Press, 2003. p. 388–396. 37

LIMA, L. P.; CAVALLI, A. R. Test execution of telecommunications services using CORBA. In: INTERNATIONAL CONFERENCE ON FORMAL METHODS FOR OPEN OBJECT-BASED DISTRIBUTED SYSTEMS, 1., 1997, Canterbury, UK. **Proceedings...** New York, NY: Springer, 1997. 39

MARICK, B. **When should a test be automated**. 1998. Disponível em: <citeseer.ist.psu.edu/marick98when.html>. Acesso em: 19 mai. 2007. 25

MARTINS, E.; MATTIELLO-FRANCISCO, M. de F. A tool for fault injection and conformance testing of distributed systems. In: LATIN AMERICAN SYMPOSIUM ON DEPENDABLE COMPUTING, 1., 2003, Berlin / Heidelberg, Germany. **Proceedings...** New York, NY: Springer, 2003. v. 2847/2003, p. 282–302. ISBN 0302-9743 (Print) 1611-3349 (Online). 39, 40

MARTINS, E.; SABIãO, S. B.; AMBROSIO, A. M. ConData: a tool for automating specification-based test case generation for communication systems. **Software Quality Jornal**, v. 8, n. 4, p. 303–320, Dec 1999. 36

MILLER, L. J. **The ISO reference model of open systems interconnection**: a first tutorial. [S.l.]: Xerox Corporation, Nov 1981. 50

MYERS, G. J. **The art of software testing**. 2. ed. USA: John Wiley and Sons, Inc., 2004. 234 p. Revised and Updated by Tom Badgett and Todd M.Thomas with Corey Sandler. 31

NATIONAL INSTRUMENTS CORPORATION (NI). **LabVIEW user's manual**: Ni publication. Austin, Texas, 1998. 476 p. Part number 320999B-01. 69

PARK, H.; RYU, H.; BAIK, J. Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In: INTERNATIONAL CONFERENCE ON SECURE SYSTEM INTEGRATION AND RELIABILITY IMPROVEMENT (SSIRI), 2., 2008, Yokohama, Japan. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2008. DOI 10.1109/SSIRI.2008.52. 38

PRESSMAN, R. S. **Engenharia de software**. 2. ed. Sção Paulo, SP: Makron Books, 1995. 1056 p. 25

_____. **Software engineering: a practitioner's approach**. 5. ed. New York, NY: McGraw Hill, 2001. 860 p. 29, 33

PROBERT, R. L.; YU, H.; SALEH, K. Relative-clock-based and test result analysis of distributed systems. In: ANNUAL INTERNATIONAL PHOENIX CONFERENCE ON

COMPUTERS AND COMMUNICATIONS (IPCCC), 11., 1992, Scottsdale, AZ, USA. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 1992. 37

REACTIVE-SYSTEMS. **Model-based Testing and Validation with Reatics(R)**. 2008. Web site. Disponível em: <`http://www.reactive-systems.com/`>. Acesso em: 01 out. 2008. 36

SANTIAGO, V.; VIJAYKUMAR, N. L.; GUIMARãES, D.; AMARAL, A. S.; FERREIRA, E. An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION (ICST 2008), 4., 2008, Lillehammer, Norway. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2008. p. 63–72. 36

SANTIAGO, V. A.; MATTIELLO-FRANCISCO, M. de F. **Protocolo de comunicação PDC-OBDH**. Ago 2006. Projeto QSEE - Qualidade do Software Embarcado em Aplicações Espaciais, FINEP. Q00-PPDOB-v06. 59, 70

SANTIAGO, V. A.; MATTIELO-FRANCISCO, F.; COSTA, R.; SILVA, W. P.; AMBRÓSIO, A. M. QSEE project: an experience in outsourcing software development for space. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING & KNOWLEDGE ENGINEERING (SEKE'07), 9., 2007, Boston, USA. **Proceedings...** Washington, DC, USA: SEKE, 2007. p. 51–56. 25, 26, 43, 44, 45, 59, 60, 67, 79, 85

SANTIAGO, V. A.; SILVA, W. P.; VIJAYKUMAR, N. L. Shortening test cases execution time for embedded software. In: INTERNATIONAL CONFERENCE ON SECURE SYSTEM INTEGRATION AND RELIABILITY IMPROVEMENT (ISSRI'08), 2., 2008, Yokohama, Japan. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2008. p. 81–88. 46, 79

SILVA, W. P.; SANTIAGO, V. A.; MATTIELO-FRANCISCO, F.; PASSOS, D. QSEE-TAS: Uma ferramenta para execução e relato automatizados de teste de software para aplicações espaciais. In: SESSÃO DE FERRAMENTAS DO SBES, 13., 2006, Florianópolis, SC, BRA. **Anais...** Porto Alegre, RS: Sociedade Brasileira de Computação (SBC), 2006. p. 43–48. 25, 27, 48

SILVA, W. P.; SANTIAGO, V. A.; VIJAYKUMAR, N. L.; MATTIELO-FRANCISCO, F. SPAC: Ferramenta para processamento e análise de dados científicos no processo de validação de software em aplicações espaciais. In: SESSÃO DE FERRAMENTAS DO SBES, 14., 2007, João Pessoa, PB, BRA. **Anais...** Porto Alegre, RS: Sociedade Brasileira de Computação (SBC), 2007. p. 32–39. 26, 27, 67, 72

SOMMERVILLE, I. **Engenharia de software**. 6. ed. Sção Paulo, SP: Prentice-Hall, 2003. 606 p. 38

SRIKANTH, H.; WILLIAMS, L. On the economics of requirements-based test case prioritization. **SIGSOFT Softw. Eng. Notes**, ACM, New York, NY, USA, v. 30, n. 4, p. 1–3, 2005. ISSN 0163-5948. 38

STOREY, N. **Safety-critical computer systems**. 1. ed. England: Addison Wesley Longman Limited, 1996. 1056 p. 33, 35

TAKAHASHI, K.; GOTOH, K.; NITANAI, S.; ISHIHATA, Y. Design and implementation of an OSI conformance test system considering extensions for interconnectability testing. In: IEEE SINGAPORE INTERNATIONAL CONFERENCE ON NETWORKS, 1993, Singapore. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 1993. v. 2, p. 665–669. 39

TANTIPRASUT, D.; NEIL, J.; FARRELL, C. ASN.1 protocol specification for use with arbitrary encoding schemes. **IEEE/ACM Transactions On Networking**, v. 5, n. 4, p. 502–513, Aug. 1997. 39

TELELOGIC. Bluetooth development with SDL and TTCN-2. **Telelogic White Papers**, v. 1, p. 1–10, 2007. 36

WALKIN, L. **The asn1c**: a free and open source compiler of ASN.1. Lionet, Oct 2007. Web Site. Disponível em: <`http://lionet.info/asn1c/`>. Acesso em: 07 nov. 2007. 39

WISSINK, T.; AMARO, C. Successful test automation for software maintenance. In: INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 2006, Pennsylvania USA. **Proceedings...** IEEE, 2006. p. 265–266. Disponível em: <`http://icsm2006.cs.drexel.edu/`>. Acesso em: 01 ago. 2007. 26, 40

YAMADA, S.; OHTERA, H.; NARIHISA, H. Software reliability growth models with testing-effort. **IEEE Transactions on Reliability**, IEEE Computer Society, New York, NY, USA, p. 19–23, 1986. 88

ZIMMERMANN, H. **OSI reference model**: the ISO model of architecture for open systems interconnection. Norwood, MA, USA: Artech House, Inc., 1988. 2–9 p. 50

A APÊNDICE A - DESENVOLVIMENTO DA QSEE-TAS/SPAC

Este apêndice descreve os princpais artefatos obtidos durante o desenvolvimento da ferramenta QSEE-TAS/SPAC.

A.1 Hierarquia dos Instrumentos Virtuais (VI's) - Módulos da QSEE-TAS

A Figura A.1 ilustra de forma simplificada o relacionamento de dependência entre os principais módulos da QSEE-TAS. Isso demostra como a ferramenta está organizada internamente.

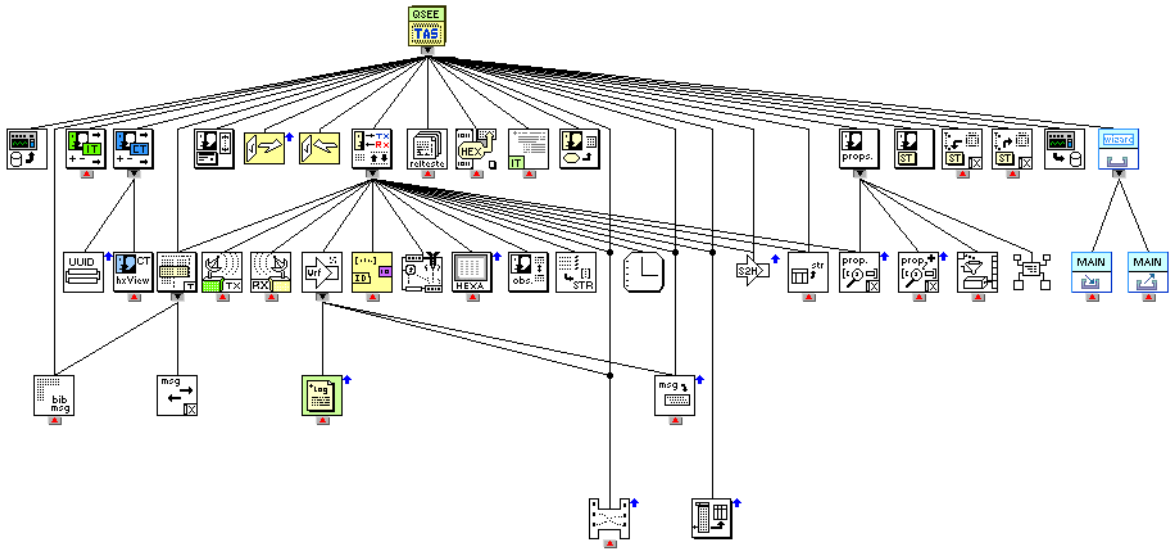


Figura A.1 - Hierarquia dos módulos da QSEE-TAS.

A.2 A Linguagem de Descrição de Mensagens

Esta sessão descreve a Linguagem de Descrição de Mensagens (LDM) que permite que mensagens sejam descritas e manipuladas pela QSEE-TAS tanto para enviar mensagem à IST quanto para processar as respostas da IST. A gramática da LDM esta apresentada na Figura A.2.

```
LDMMessage := [FieldDef]+ ;
FieldDef := IDENTIFIER FieldSizeSpec FieldValueSpec ;
FieldSizeSpec := FixedSizeDef
               |  VariableSizeDef
               ;
FixedSizeDef  := [0-9]+
```

```
VariableSizeDef := '[' MinimumSizeDef '..' MaximumSizeDef ']' ;
FieldValueSpec := HexadecimalValue
                | ScriptDefinedValue
                ;
HexadecimalValue := [HEXDIGIT]+ h ;
ScriptDefinedValue := '{' ScriptInvocation '}' ;
ScriptInvocation := IDENTIFIER '(' Arguments ')' ;
Arguments := Expression ArgumentList ;
ArgumentList := ',' Expression ArgumentList
              |
              ;
HEXDIGIT := [0-9a-fA-F] ;
IDENTIFIER := [A-Za-z][A-Za-z0-9]+ ;
```

Figura A.2 - Gramática LDM.

A.3 Arquivo XML para Importação e Exportação

A QSEE-TAS pode intereoperar com outras ferrametas por meio de importação e exportação de dados de projeto de teste via arquivo XML cujo DTD (*Document Type Definition*) está apresentado na Figura A.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : qsee-tas-tdx.dtd
    Created on : 20 de Novembro de 2006, 09:44
    Author     : Wendell Pereira da Silva
    Description:
        The purpose of this document is to describe the syntax
        of the XML file to be used in test data interchange
        defined in the context of QSEE-TAS tool kit.
    PUBLIC ID  : -//INPE-DAS//vocabulary//EN
    SYSTEM ID  :
    http://www.cea.inpe.br/qsee/tas/pub/qsee-tas-test-data.dtd
-->
<!ELEMENT test-data (meta*, test-spec, test-report?)>
```

```
<!ATTLIST test-data version CDATA #REQUIRED>
<!ELEMENT meta EMPTY>
<!ATTLIST meta
    name  CDATA #REQUIRED
    value CDATA #REQUIRED
>
<!ELEMENT test-spec (environment?, test-item*, test-case*)>
<!ELEMENT environment (sut?, designation?, test-team?, channels?)>
<!ELEMENT sut (#PCDATA)>
<!ATTLIST sut
    version CDATA #IMPLIED
    spec    CDATA #IMPLIED
>
<!ELEMENT designation (#PCDATA)>
<!ELEMENT test-team (#PCDATA)>
<!ELEMENT channels (channel+)>
<!ELEMENT channel (description, parameters)>
<!ATTLIST channel
    id CDATA #REQUIRED
    type (RS-232|Parallel|TCP-IP|UDP-IP|RawSocket) "RS-232"
>
<!ELEMENT parameters (param*)>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param
    name CDATA #REQUIRED
>
<!ELEMENT test-item (description, purpose?, related-requirements?,
  related-test-items?, related-test-cases?)>
<!ATTLIST test-item
    id       CDATA #REQUIRED
    category (functional|performance|dependability) "functional"
>
<!ELEMENT related-requirements (#PCDATA)>
<!ELEMENT related-test-items (id*)>
<!ELEMENT related-test-cases (id*)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT test-case (description, purpose?, related-requirements?,
  txt-verdict?, interaction*)>
```

```
<!ATTLIST test-case
    id CDATA #REQUIRED
    executed (true|false) "false"
    last-exec-date CDATA #IMPLIED
    last-verdict
     (null|pass|pass-restrict|fail|inconclusive|error) "null"
>
<!ELEMENT description (#PCDATA)>
<!ELEMENT purpose (#PCDATA)>
<!ELEMENT txt-verdict (#PCDATA)>
<!ELEMENT interaction ((action, expected-reaction) | observation)>
<!ATTLIST interaction
type (action-reaction|observation) "action-reaction"
>
<!ELEMENT action (message)>
<!ELEMENT expected-reaction (timeout|message|dont-care)>
<!ELEMENT message (description?, field*)>
<!ATTLIST message
    channel-id CDATA "C01"
    iterations CDATA "1"
    delay CDATA "500"
    retries CDATA "0"
>
<!ELEMENT field (#PCDATA)>
<!ATTLIST field
    name CDATA #REQUIRED
    size CDATA #REQUIRED
    content-type
      (hexa-string|text-ascii|decimal-string|script) "hexa-string"
>
<!ELEMENT observation (description?, point+)>
<!ELEMENT dont-care EMPTY>
<!ELEMENT timeout EMPTY>
<!ELEMENT point (local, expected-result)>
<!ELEMENT local (#PCDATA)>
<!ELEMENT expected-result (#PCDATA)>
<!ELEMENT test-report (test-session*)>
<!ELEMENT test-session (environment, test-leader, sut-leader, log*)>
```

```
<!ATTLIST test-session
    id CDATA #REQUIRED
    status (started|conclued) "started"
    start-date CDATA #REQUIRED
    end-date CDATA #IMPLIED
>
<!ELEMENT test-leader (#PCDATA)>
<!ELEMENT sut-leader (#PCDATA)>
<!ELEMENT log (test-case, appraisal, trace*)>
<!ELEMENT appraisal (#PCDATA)>
<!ATTLIST appraisal
    verdict (null|pass|pass-restrict|fail|inconclusive|error) "null"
>
<!ELEMENT trace (timestamp, (obs|msg))>
<!ATTLIST trace
    type (obs|tx|rx) #REQUIRED
>
<!ELEMENT obs (description, expected-result, observed-result)>
<!ELEMENT observed-result (#PCDATA)>
<!ELEMENT msg (#PCDATA)>
<!ATTLIST msg
    direction (tx|rx) #REQUIRED
>
```

Figura A.3 - DTD validador do XML para intercâmbio de dados com a QSEE-TAS.

A.4 Modelo do Banco de Dados Relacional

A Figura A.4 apresenta o modelo ER que foi concebido para estruturar os dados de um Projeto de Teste da QSEE-TAS.

O modelo ER foi traduzido para o modelo de implementação (físico) usando-se os tipos de dados ANSI/SQL do MySQL, ilustrado na Figura A.5.

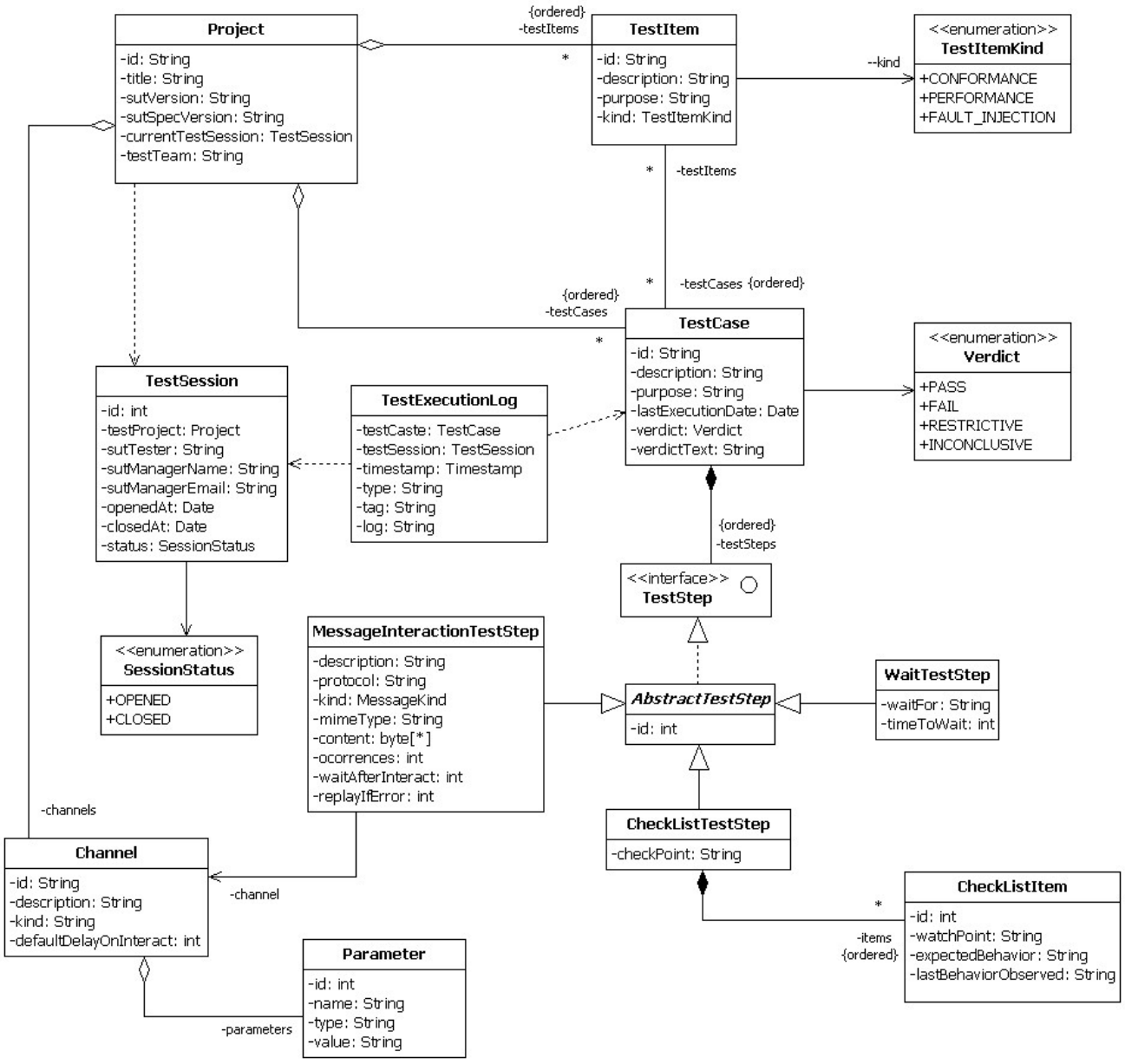


Figura A.4 - Modelo lógico do banco de dados da QSEE-TAS.

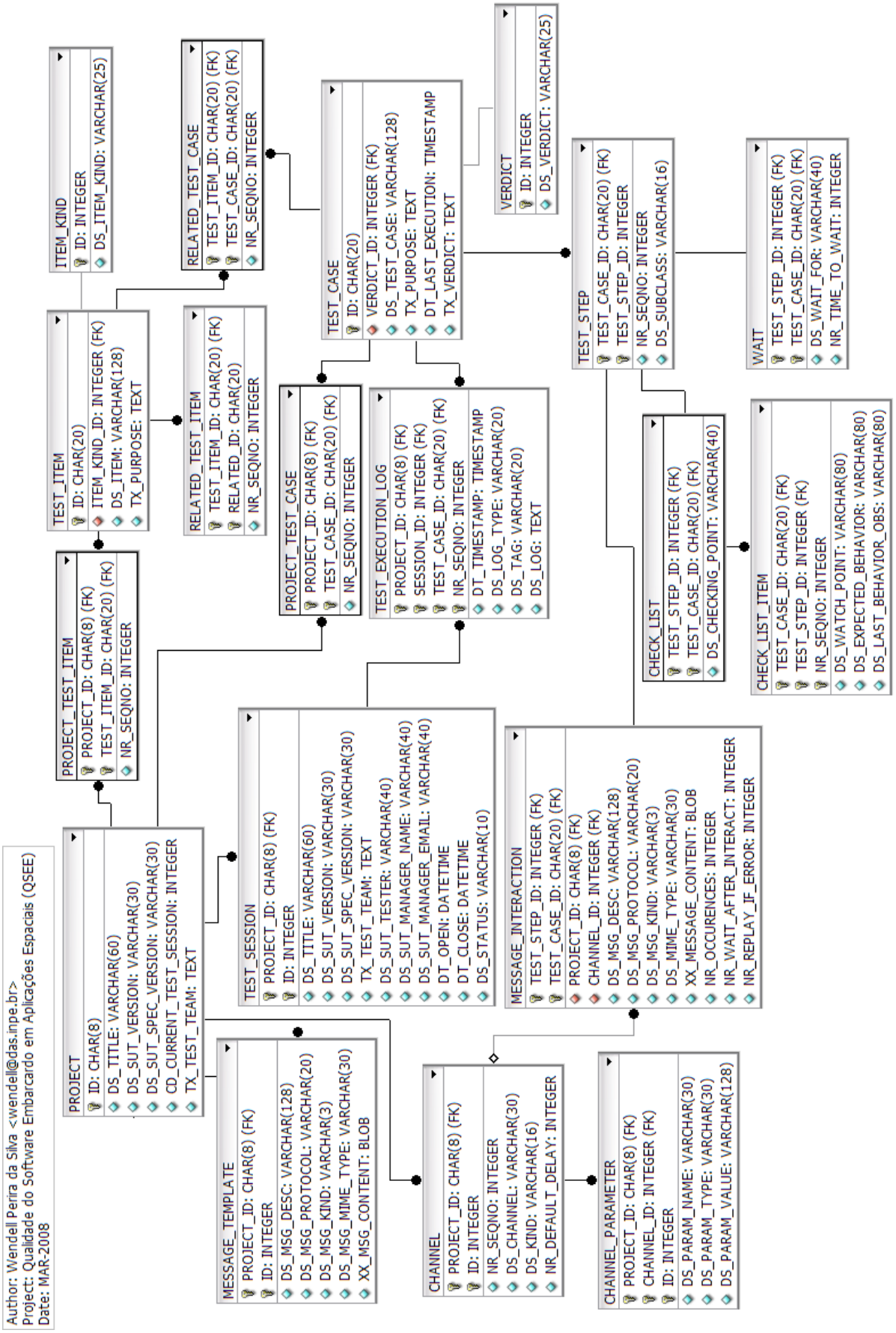


Figura A.5 - Modelo físico do banco de dados da QSEE-TAS.

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

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

Manuais Técnicos (MAN)

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

Notas Técnico-Científicas (NTC)

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

Relatórios de Pesquisa (RPQ)

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

Propostas e Relatórios de Projetos (PRP)

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

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

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

Programas de Computador (PDC)

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

Pré-publicações (PRE)

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