sid.inpe.br/mtc-m19/2012/03.29.14.24-TDI

UMA ARQUITETURA FLEXÍVEL PARA O MODELO DO SUBSISTEMA DE COMPUTAÇÃO DE BORDO DE UM SIMULADOR DE SATÉLITES

Joaquim Pedro Barreto

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pela Dra. Ana Maria Ambrósio, aprovada em 12 de abril de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BKC7A2>

INPE
São José dos Campos
2012

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) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

CONSELHO DE EDITORAÇÃO E PRESERVAÇÃO DA PRODUÇÃO INTELECTUAL DO INPE (RE/DIR-204):
Presidente:
Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)
Membros:
Dr. Antonio Fernando Bertachini de Almeida Prado - Coordenação Engenharia e Tecnologia Espacial (ETE)
Drª Inez Staciarini Batista - Coordenação Ciências Espaciais e Atmosféricas (CEA)
Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)
Dr. Germano de Souza Kienbaum - Centro de Tecnologias Especiais (CTE)
Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos (CPT)
Drª Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação
Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)
BIBLIOTECA DIGITAL:
Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)
REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:
Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)
Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)
EDITORAÇÃO ELETRÔNICA:
Vivéca Sant´Ana Lemos - Serviço de Informação e Documentação (SID)

sid.inpe.br/mtc-m19/2012/03.29.14.24-TDI

UMA ARQUITETURA FLEXÍVEL PARA O MODELO DO SUBSISTEMA DE COMPUTAÇÃO DE BORDO DE UM SIMULADOR DE SATÉLITES

Joaquim Pedro Barreto

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais, orientada pela Dra. Ana Maria Ambrósio, aprovada em 12 de abril de 2012.

URL do documento original:
<http://urlib.net/8JMKD3MGP7W/3BKC7A2>

INPE
São José dos Campos
2012

Dados Internacionais de Catalogação na Publicação (CIP)

Barreto, Joaquim Pedro.
B275u Uma arquitetura flexível para o modelo do subsistema de computação de bordo de um simulador de satélites / Joaquim Pedro Barreto. – São José dos Campos : INPE, 2012.
xx + 116 p. ; (sid.inpe.br/mtc-m19/2012/03.29.14.24-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012.
Orientadora : Dra. Ana Maria Ambrosio.

1. simulador satélite. 2. SMP. 3. OBDH. 4. arquitetura software. 5. modelagem. I.Título.

CDU 629.783

Copyright © 2012 do MCT/INPE. Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação, ou transmitida sob qualquer forma ou por qualquer meio, eletrônico, mecânico, fotográfico, reprográfico, de microfilmagem ou outros, sem a permissão escrita do INPE, com exceção de qualquer material fornecido especificamente com o propósito de ser entrado e executado num sistema computacional, para o uso exclusivo do leitor da obra.

Copyright © 2012 by MCT/INPE. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, microfilming, or otherwise, without written permission from INPE, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use of the reader of the work.

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

Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais

Dr. Mauricio Gonçalves Vieira Ferreira

Presidente / INPE / SJCampos - SP

Dra. Ana Maria Ambrosio

Orientador(a) / INPE / São José dos Campos - SP

Dr. Walter Abrahão dos Santos

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

Dra. Selma Shin Shimizu Melnikoff

Convidado(a) / USP / São Paulo - SP

Este trabalho foi aprovado por:

( ) maioria simples

( ) unanimidade

Aluno (a): Joaquim Pedro Barreto

São José dos Campos, 12 de abril de 2012

“Cada um de nós compõe a sua história e cada ser em si carrega o dom de ser capaz”.

Almir Sater e Renato Teixeira

AGRADECIMENTOS

Agradeço a Deus esta oportunidade.

Também agradeço a minha esposa Júlia, o amor e compreensão.

E a minha orientadora, Dra. Ana, o apoio e paciência, o incentivo nos momentos de incerteza e dificuldades, a fim de que este trabalho fosse concretizado.

Aos amigos, que, direta ou indiretamente, colaboraram para a realização deste trabalho.

RESUMO

Simuladores de satélites são utilizados durante o ciclo de vida de uma missão espacial para diferentes finalidades e por isso a reutilização dos mesmos em diferentes missões permite reduzir prazos e custos. Dentre os subsistemas simulados de um satélite, o subsistema de computação de bordo é um dos mais complexos e reutilizados. Este trabalho propõe uma arquitetura para o modelo do subsistema de computação de bordo para um simulador de satélites, baseado nos padrões SMP e PUS, existentes na área espacial. A utilização de padrões no desenvolvimento de simuladores implica maior interoperabilidade e promove o reuso levando a sistemas desenvolvidos com maior qualidade e menor quantidade de falhas pela herança de código previamente validado e verificado. Uma prova de conceito da flexibilidade da arquitetura proposta foi realizada através da implementação e teste de um modelo de computador de bordo simplificado. A arquitetura é baseada na decomposição do modelo em serviços e no conceito de componentes de software. Um protótipo simplificado baseado no padrão europeu para simuladores de satélite SMP foi desenvolvido para verificação da arquitetura proposta. Uma análise sobre as dificuldades de uso dos padrões, a necessidade de ferramentas especializadas para aumentar a produtividade de um ambiente de desenvolvimento e as vantagens da solução proposta também são apresentados nesta dissertação.

A FLEXIBLE ARCHITECTURE FOR A SATELLITE SIMULATOR ONBOARD COMPUTER SUBSYSTEM MODEL

ABSTRACT

Satellite simulators are used throughout a mission lifecycle for several purposes and reusing them in different missions can help mitigating developing time and costs. Among all satellite subsystems simulated, the onboard computer subsystem is one of the most complexes and reused. This work proposes an architecture to this subsystem for a satellite simulator, based SMP and PUS space standards. The use of standards to develop satellite simulators implies higher interoperability and promotes reuse leading to applications developed with higher quality and lower failures due to previously validated and verified code heritage. A concept proof of the architecture flexibility was done by implementing and testing a simplified on board computer model. The model was decomposed in services and its implementation followed the software concept approach. A simplified simulator prototype based on the SMP satellite Simulator standard was developed for testing purposes. An analysis about the difficulties to use the standards, the need of specialized tools to increase productivity in a development environment and the advantages of the proposed solution are also presented in this document.

LISTA DE FIGURAS

Pág.

Figura 1.1 - Comunalidade entre requisitos de software das missões Rosetta/VenusExpress/Mars Express. 1
Figura 1.2 - Metodologia do trabalho. 6
Figura 2.1 - Arquitetura do padrão SMP. 14
Figura 2.2 - Arquitetura típica de um simulador SMP. 16
Figura 2.3 - Diagrama hierárquico de componentes do SMP. 17
Figura 2.4 - Relacionamentos entre componentes. 18
Figura 2.5 - Interfaces de serviços do SMP. 19
Figura 2.6 - Interfaces de Eventos de modelos. 21
Figura 2.7 - Interfaces IEntryPoint e ITask. 22
Figura 2.8 - Mecanismo de Invocação dinâmica. 22
Figura 2.9 - Diagrama de sequência do mecanismo de Invocação Dinâmica. 23
Figura 2.10 - Interface de publicação. 24
Figura 2.11 – Mecanismo de Persistência de dados. 25
Figura 2.12 - Padrão TelemetryStream. 29
Figura 2.13 - Gerador de Meta-componentes. 30
Figura 2.14 - Processo de criação de aplicações. 30
Figura 2.15 - Geração automática de códigos e casos de teste. 31
Figura 3.1 – Principais funções de um OBC. 34
Figura 3.2 - Lista de serviços do padrão PUS - Parte 1. 36
Figura 3.3 - Lista de serviços do padrão PUS - Parte 2. 37
Figura 3.4 - Estrutura do pacote de telemetria do padrão PUS. 38
Figura 4.1 - Interfaces interna e externa de um OBDH. 42
Figura 4.2 - Diagrama de contexto da arquitetura flexível para um Modelo de OBDH. 43
Figura 4.3 - Diagrama de atividades para utilização dos mecanismos de comunicação. 46
Figura 4.4 - Arquitetura genérica baseada em componentes. 47
Figura 4.5 - Especialização de um modelo de OBDH. 48
Figura 5.1 - Arquitetura do protótipo proposto. 53
Figura 5.2 - Máquina de estados de um simulador SMP. 54
Figura 5.3 - Classe principal do protótipo do simulador. 56
Figura 5.4 - Classe de *Log*. 58
Figura 5.5 - Classe de escalonamento. 59
Figura 5.6 - Classe de gerenciamento de eventos do sistema. 59
Figura 5.7 - Classe de implementação do modelo SOLO. 61
Figura 5.8 - Classe de implementação do subsistema TT&C. 62
Figura 5.9 - Classe de implementação do modelo Térmica. 63
Figura 5.10 - Classe de implementação do modelo PSS 64
Figura 5.11 - Fluxo de dados entre os modelos dos subsistemas. 66
Figura 5.12 - Hierarquia para criação de um modelo específico no padrão SMP. 70

Figura 5.13 - Diagrama de classes do MDK com as classes necessárias ao protótipo... 71
Figura 5.14 - Diagrama de classe dos subsistemas..... 73
Figura 5.15 - Mecanismos de comunicação utilizados no protótipo. 75
Figura 5.16 - Diagrama de sequência de comunicação. 77
Figura 5.17 - OBDH especializado em diferentes componentes de software. 80
Figura 6.1 – Configuração do modelo do OBDH do cenário 3. 84
Figura 6.2 – Interface gráfica do protótipo do simulador. 85
Figura 6.3 - Evolução dos valores dos parâmetros. 86
Figura 6.4 - Execução do serviço Evento-Ação no limite inferior 87
Figura 6.5 - Execução do serviço Evento-Ação no limite superior 88
Figura 6.6 - Envio de comando para requisição do relatório de monitoração de parâmetros 89
Figura 6.7 – Relatório de monitoração de parâmetros..... 89
Figura A.1 - Relação das interfaces do SMP 104
Figura B.1 – Configuração de serviços do OBDH do cenário 1 105
Figura B.2 – Componentes utilizados na execução do cenário 1 106
Figura B.3 – Agendamento e execução de telecomando 107
Figura B.4 – Configuração de serviços do OBDH do cenário 2 108
Figura B.5 – Componentes utilizados na execução do cenário 2 109
Figura B.6 – Agendamento e execução de telecomando 110
Figura B.7 – Relatório de estatística dos parâmetros 111
Figura B.8 – Configuração de serviços do OBDH do cenário 4 112
Figura B.9 – Componentes utilizados na execução do cenário 4 113
Figura B.10 – Agendamento e execução de telecomando 114
Figura B.11 – Relatório de estatística dos parâmetros 115

LISTA DE TABELAS

Pág.

Tabela 4.1 - Cenários para avaliação da arquitetura..... 52
Tabela 5.1 - Fluxo de dados entre os modelos TT&C, OBDH, PSS, Térmica e SOLO. 66
Tabela 6.1 – Valores máximo e mínimo dos parâmetros. 83

LISTA DE SIGLAS E ABREVIATURAS

AOCS	*Attitude and Orbital Control System*
APID	*Application Process IDentification*
INPE	Instituto Nacional de Pesquisas Espaciais
CBERS	*China and Brazil Earth Resource Satellite*
DLL	Biblioteca de vínculo dinâmico *(Dynamic Link Library)*
ECSS	*European Cooperation for Space Standardization*
ESA	*European Space Agency*
GS	Estação Terrena (*Ground Station*)
MDA	*Model Driven Architecture*
OBC	*On Board Computer*
OBDH	*On Board Data Handling*
MDK	*Model Development Kit*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
PSS	*Power Supply System*
PUS	*Packet Utilization Standard*
SIMC3	Simulador para o Satélites CBERS3&4
SIMSAT	*Software Infrastructure for Modeling Satellites*
SMDL	*Simulation Model Definition Language*
SMP	*Simulation Modeling Platform*
TC	Telecomando
TM	Telemetria
TT&C	*Telemetry Tracking and Control*
UML	*Unified Modelling Language*
USA	*United Space Alliance*
UTC	*Coordinated Universal Time*
XML	*Extensible Markup Language*
XSL	*EXtensible Stylesheet Language*

SUMÁRIO

Pág.

1 INTRODUÇÃO 1
1.1 Objetivo 4
1.2 Metodologia 5
1.3 Organização do documento 7
2 SIMULADORES DE SATÉLITES 9
2.1 Arquiteturas de simuladores de satélites 9
2.2 Simulation Modelling Platform (SMP) 13
2.2.1 Visão Geral da Simulation Modelling Platform (SMP) 14
2.2.2 Mecanismos para inter-relacionamento entre componentes do SMP 20
2.2.3 Criação do ambiente de simulação 25
2.3 O Framework OBS 27
2.3.1 Visão Geral do framework OBS 27
2.3.2. Criação de aplicações no ambiente do *framework* 30
2.4 Comparação entre SMP e o *Framework* OBS 32
3 SUBSISTEMAS DE COMPUTAÇÃO DE BORDO PARA SIMULADORES 33
3.1 Subsistema de computação de Bordo 33
3.2 Packet Utilization Standard (PUS) 35
4 ARQUITETURA FLEXÍVEL PARA UM MODELO DE OBDH 41
4.1 Descrição geral da arquitetura 41
4.2 Definição das interfaces e dos mecanismos de inter-relacionamento 44
4.3 Exemplo de aplicação da arquitetura 48
4.4 Verificação da arquitetura 49
5 DESCRIÇÃO DO PROTÓTIPO 53
5.1 Visão geral do protótipo simplificado 53
5.2 Os modelos do simulador 60
5.3 Fluxo de dados entre os modelos 65
5.4 *Model Development Kit* (MDK) 69

5.5 Mecanismos de comunicação implementados no protótipo 74
5.6 Os vários modelos de ODBH 79
6 RESULTADOS.....83
6.1 Configuração inicial do protótipo do simulador 83
6.2 Execução dos cenários 84
7 CONCLUSÃO.....91
7.1 Vantagens da arquitetura proposta.....91
7.2 Dificuldades encontradas no uso do padrão SMP.....93
7.3 Limitações 94
7.4 Trabalhos Futuros 95
REFERÊNCIAS BIBLIOGRÁFICAS 97
APÊNDICE A.....103
APÊNDICE B.....105
B.1 Cenário 1.....105
B.2 Cenário 2.....108
B.3 Cenário 4.....112
B.4 Cenário 5.....116

1 INTRODUÇÃO

Toda missão espacial requer a construção de simuladores de satélite. Portanto, a reutilização dos mesmos é de fundamental importância para reduzir os custos da missão. A reutilização de um simulador de satélites pode dar-se entre missões, levando-se em conta a similaridade de um satélite para outro. Nas missões Rosetta, Mars Express e Vênus Express, constatou-se que aproximadamente 50% dos requisitos eram comuns às três missões (REGGESTAD et al., 2004). A Figura 1.1 mostra os detalhes numéricos desta comunalidade.

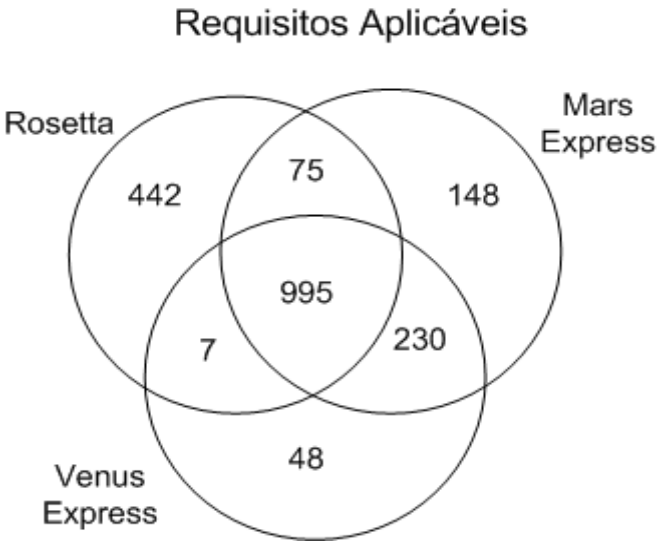


Figura 1.1 - Comunalidade entre requisitos de software das missões Rosetta/VenusExpress/Mars Express.
Fonte: Adaptada de Reggestad et al (2004)

Dado que um simulador de satélite consta de uma forma simplificada de representar o satélite, conclui-se que a proporção de requisitos comuns entre satélites seja a mesma proporção de similaridade entre os seus respectivos simuladores. Além do aspecto da comunalidade do simulador de um satélite para outro satélite, um simulador pode ainda ser reutilizado dentro das várias fases de uma mesma missão, ao longo das várias etapas de verificação e validação (EICKHOFF, 2009).

Considerando que o desenvolvimento de simuladores de satélite é uma tarefa dispendiosa tanto em tempo quanto em recursos financeiros, a mesma só é justificável se o custo de desenvolvimento for inferior ao de criação de um equipamento físico ou então se o simulador pudesse ser reutilizado em diversas fases ou missões.

Um dos subsistemas que está presente em praticamente qualquer satélite, com maior ou menor complexidade e que pode sofrer variações em tamanho e complexidade em um simulador de satélites é o subsistema de computação de bordo, também chamado de subsistema para manipulação de dados a bordo, ou simplesmente OBDH (*On-Board Data Handling*). Em um simulador, o OBDH pode ser um modelo que contenha o computador de bordo físico, tal como ele estaria no satélite, incluindo o hardware e o software. Nesse caso, o simulador é classificado como um simulador com "*hardware-in-the-loop*". Alternativamente, o modelo OBDH pode ter o hardware emulado e o software de bordo real. Em uma forma mais simplificada, o modelo OBDH pode ser composto de um software que implementa o conjunto completo ou parcial das funções do OBDH, sejam elas implementadas em hardware ou software no computador de bordo real. Todas estas variações de modelos são possíveis e, em geral, são implementadas em uma missão. Esta variação de um modelo de OBDH em um simulador depende do objetivo da verificação, em cada fase do desenvolvimento da missão espacial.

A abordagem de desenvolvimento de um simulador de satélites, a forma como seus elementos (subsistemas representados por modelos) interagem, o grau de fidelidade na representação, a inclusão ou não de "*hardware-in-the-loop*", a definição do acoplamento entre os elementos a serem simulados, bem como a possibilidade de inserção e retirada destes elementos a qualquer momento, de forma prática e rápida, são itens merecedores de estudos aprofundados para melhor reutilização e, consequentemente, redução de custos e prazos da missão.

Com relação à reutilização no panorama de simuladores de satélites, o simulador SIMC3 em desenvolvimento no INPE, no âmbito da missão CBERS (*China-Brazil Earth Resource Satellite*), teve seu projeto inicial incluindo soluções para permitir reutilização, se não total, pelo menos parcial do software. A arquitetura geral do SIMC3 e uma análise comparativa com outros simuladores operacionais do INPE são apresentadas em (AMBROSIO et al., 2006). Entretanto, o projeto do SIMC3 não contemplou soluções do padrão *Simulation Modeling Platform* (SMP) estabelecido pela Agência Espacial Européia. Um estudo foi realizado para avaliar o custo em se adotar a infraestrutura do SMP no simulador SIMC3 (BARRETO et al., 2010). O uso de uma arquitetura que satisfaça a necessidade exclusiva de um simulador de satélite específico implica a readequação da mesma para uso em uma nova missão, resultando em maiores custos, prazos e complexidade. Portanto, a busca por uma arquitetura não vinculada a uma determinada configuração de satélite representa um ganho para futuros projetos de simuladores.

Um estudo sobre simuladores desenvolvidos para missões espaciais no âmbito da ESA (*European Space Agency*), realizado durante esta dissertação (AMBROSIO; BARRETO, 2012), mostrou que o padrão SMP (*Simulation Modelling Platform*) para desenvolvimento de simuladores de satélites evidenciou-se como uma solução em franca implementação por membros daquela agência e que também já está sendo estudado e analisado fora do continente europeu pela USA (*United Space Alliance*) (NEMETH; DEMAREST, 2010), por promover fortemente o reuso, a portabilidade e a interoperabilidade. Este padrão (ECSS, 2011) define um conjunto de interfaces que se constituem em um arcabouço de software para desenvolvimento de simuladores de satélites. O arcabouço provido pelo SMP pretende garantir que qualquer simulador nele baseado implemente modelos, serviços e mecanismos de comunicação compatíveis entre si.

O padrão SMP apresenta características que vão ao encontro de uma proposta de decomposição do modelo de um OBDH em componentes que contemplem

as especificidades de reuso e interoperabilidade, bem como a comunicação entre os componentes e o ambiente de simulação.

O SMP padroniza o desenvolvimento de simuladores de satélites, pois utiliza técnicas e mecanismos para elaboração de uma arquitetura de software baseada em componentes, visando a independência de plataforma (portabilidade), a interoperabilidade e o reuso dos componentes do simulador. Entre as técnicas de software utilizadas pelo padrão estão a Orientação a Objeto (SOMMERVILE, 2010; PRESSMAN, 2005), a UML (*Unified Modelling Language*) (UML, 2011), a arquitetura orientada a modelos (*Model Driven Architecture*) (MDA, 2003) e Componentes de Software (SOMMERVILLE, 2010).

Além do padrão SMP, a ECSS (*European Cooperation for Space Standardization*) definiu o padrão PUS (*Packet Utilization Standard*) que padroniza as funcionalidades comuns aos subsistemas de computação de bordo e sua comunicação com os sistemas do Segmento Solo (ECSS, 2003). O padrão PUS define um conjunto de serviços comuns que, apesar de não focarem necessidades de simuladores de satélites, podem ser implementados em diferentes níveis de fidelidade em um simulador de satélites. Em (AMBROSIO et al., 2007) é apresentado um estudo sobre o uso de serviços padronizados em simuladores de satélites.

1.1 Objetivo

O objetivo deste trabalho é propor uma arquitetura de software flexível, baseada em componentes de software, para o modelo de subsistema de computação de bordo de um simulador de satélite, levando-se em conta a reutilização destes componentes no desenvolvimento de diferentes simuladores.

A arquitetura flexível é conveniente no cenário em que múltiplas configurações de OBDH são possíveis para um simulador, dependendo do objetivo do

simulador. As múltiplas configurações do modelo OBDH, por sua vez, vão depender dos serviços incluídos e dos níveis de fidelidade exigidos para o modelo.

Ainda como objetivo desta dissertação está o conhecimento com exploração e domínio dos padrões SMP e PUS como estado-da-arte em simuladores e OBDHs na área espacial.

O padrão SMP está diretamente relacionado ao projeto e ao desenvolvimento de simuladores de satélite de forma geral, enquanto que o padrão PUS apresenta uma organização dos serviços comuns de um OBDH, embora não seja comumente utilizado nos simuladores pesquisados.

1.2 Metodologia

O desenvolvimento deste trabalho envolveu: (i) o estudo de arquiteturas de simuladores de satélites existentes, de sistemas de computação de bordo e de padrões relacionados ao tema: SMP, PUS, *DesignPatern*s; (ii) a proposição de uma arquitetura para o modelo de OBDH; (iii) o desenvolvimento de um protótipo de simulador incluindo, entre outros, o modelo de OBDH na arquitetura proposta; (iv) a execução de um conjunto de cenários para verificação da arquitetura proposta e, finalmente, (v) uma análise dos resultados obtidos e conclusões.

A Figura 1.2 sintetiza a metodologia utilizada para a realização deste trabalho.

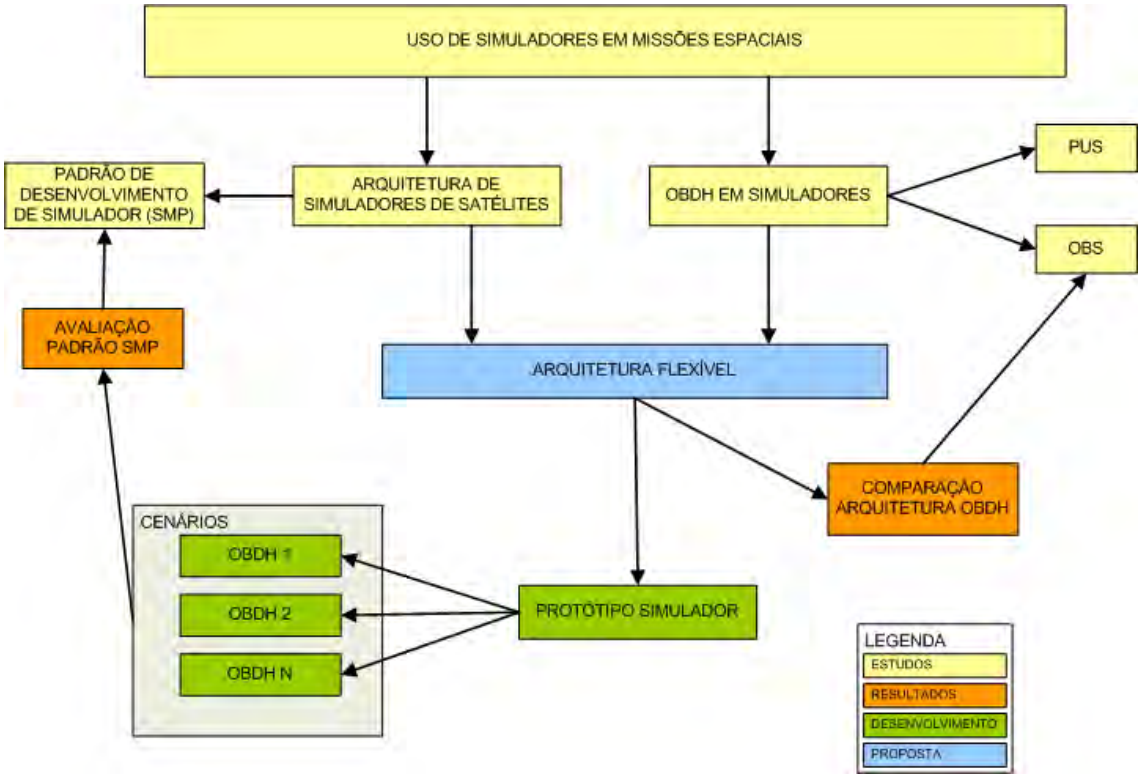


Figura 1.2 - Metodologia do trabalho.

Inicialmente, um estudo sobre arquiteturas e uso de simuladores em missões espaciais foi realizado, mostrando a utilização dos mesmos nas diferentes etapas de missões. Pode-se observar a importância e a preocupação com o reuso, bem como a tendência da utilização de padrões no desenvolvimento de simuladores, para mitigação de custos e prazos. Como continuidade do estudo, uma análise de arquiteturas de simuladores foi feita, no âmbito da ESA. E este estudo de arquiteturas mostrou que o SMP vem ganhando destaque no desenvolvimento dos simuladores (CÔME; IRVINE, 1998; DELHAISE; BRU, 2006; HOMEM et al., 2006; LINDMAN et al., 2006; FRITZEN et al., 2010; PIGNÈDE et al, 2010).

O estudo de arquiteturas de simuladores permitiu observar também que soluções visando o reuso de modelos de subsistemas de OBDH em simuladores de satélites não foram encontradas. Nos simuladores analisados, normalmente o OBDH é um modelo fechado. Em função disso e da

necessidade identificada de se buscar uma solução para o subsistema de OBDH do simulador SIMC3 (**SIM**ulador para o satélite **C**BERS**3**), o estudo foi direcionado para aplicação neste subsistema. Um padrão e um estudo similar foram considerados na metodologia. O padrão PUS (ECSS, 2003) para OBDHs define um conjunto de serviços típicos de um OBDH.

O trabalho de (P&P SOFTWARE GMBH, 2003) define uma arquitetura para um subsistema de OBDH baseada em *Design Patterns* e é chamado de *Framework OBS* (*On Board Software*). A solução apresentada não é voltada para um simulador de satélites mas sim para o software a bordo de um satélite real. Uma breve comparação entre o *framework* OBS e a solução proposta nesta dissertação foi realizada.

Uma arquitetura foi proposta, baseada nos padrões SMP e PUS, com seus respectivos serviços para OBDH. Um protótipo de simulador de satélites simplificado foi desenvolvido, para a verificação da arquitetura. Um conjunto de cenários foi então elaborado, visando verificar a flexibilidade e reúso da arquitetura.

Após a execução dos cenários foi feita uma análise geral da arquitetura. Além disso, uma avaliação do padrão SMP foi produzida, considerando-se o grau de atendimento às necessidades, a dificuldade/facilidade no uso e eventuais sugestões de melhoria.

1.3 Organização do documento

Esta dissertação está estruturada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica mostrando um estudo sobre arquiteturas de simuladores e o padrão SMP para desenvolvimento de simuladores de satélites. O Capítulo 3 mostra o subsistema de computação de bordo e suas características bem como o padrão PUS e seus respectivos serviços. O Capítulo 4 detalha a arquitetura proposta e os cenários de uso para verificação enquanto que o Capítulo 5 apresenta os detalhes da implementação da

arquitetura e do protótipo. O Capítulo 6 mostra os testes realizados, os experimentos aplicados à arquitetura e os resultados obtidos na implementação e testes e o Capítulo 7 apresenta as conclusões finais. Adicionalmente, o Apêndice A mostra todas as interfaces do padrão SMP e o Apêndice B mostra os resultados complementares ao Capítulo 6.

2 SIMULADORES DE SATÉLITES

Este capítulo apresenta um resumo do estudo sobre as arquiteturas encontradas na literatura no âmbito da Agência Espacial Européia (ESA); um detalhamento do padrão para simuladores de satélites, o SMP *(Simulation Modelling Platform*); e uma descrição do *framework* OBS (*On Board Software*) para software a bordo de satélites reais.

2.1 Arquiteturas de simuladores de satélites

O estudo sobre arquiteturas de simuladores de satélites permitiu o levantamento de evidências da importância dos simuladores nas missões espaciais, suas aplicações em diferentes momentos do desenvolvimento de um satélite e o seu reuso para diminuir custos e prazos. O estudo também mostrou como a ESA direciona o desenvolvimento de simuladores, fazendo uso de padronização e recomendando que as empresas desenvolvam dentro destes padrões.

Buscando atingir os objetivos de redução de custos e prazos através do reuso, a ESA, primeiramente definiu um núcleo de simulação chamado SIMSAT (WILLIAMS, 1992; HOMEM et al., 2006), composto de: (i) uma estrutura de simulação, contendo uma ou mais interface(s) com o usuário, (ii) um núcleo de serviços de simulação e (iii) uma biblioteca de modelos genéricos. Entre os serviços de simulação estão: núcleo de tempo real, escalonamento de eventos, gerenciamento de dados, *log* de mensagens, comandos, modelagem de equipamentos de estações terrenas como codificadores de telecomandos e decodificadores de telemetrias. A biblioteca de modelos e componentes genéricos, que podem ser utilizados no desenvolvimento de um simulador operacional específico, contém modelos de solo, modelos dinâmicos de órbita e veículos espaciais e um conjunto de emuladores para processadores, visando emular o hardware dos computadores de bordo. O SIMSAT é a base para criação de simuladores no âmbito da ESA e, em sua versão mais recente, é compatível com o padrão SMP. A própria arquitetura do SMP mostra que o

conhecimento adquirido no projeto e desenvolvimento do SIMSAT foi aplicado na estruturação do SMP.

Outra arquitetura analisada foi a do simulador CSIM (HOMEM et al., 2006; PIDGEON et al., 2008) vinculado à constelação de satélites GALILEO (ESA, 2012a). Trata-se de um simulador bastante complexo, podendo chegar a 51 modelos de satélites em simulação, incluindo ainda diversas estações terrenas. O simulador faz uso do SIMSAT para o núcleo da simulação e incorpora modelos específicos de subsistemas dos satélites da missão GALILEO. Além disso, implementa interfaces externas com a estação real de controle, com um simulador de interface do segmento de missão e com estações simuladas. No que tange à simulação do satélite, a arquitetura reaproveita os modelos genéricos do SIMSAT de dinâmica orbital e ambiente espacial, do comportamento térmico e elétrico. Já os subsistemas de controle de órbita e atitude, controle e gravação de dados, cargas úteis bem como aspectos específicos de controle térmico e suprimento de energia, foram desenvolvidos especificamente para o CSIM.

O simulador para a missão XMM (*X-ray Multi Mirror*) (ESA, 2012b) também fez uso do SIMSAT, embora adotasse uma abordagem diferente do CSIM. Sua arquitetura (CÔME; IRVINE, 1998) separa os subsistemas do satélite em módulos com conexões em três barramentos: OBDH, elétrico e térmico. O mesmo ocorre com as cargas úteis a serem inseridas no simulador. Estas podem ainda serem simuladas em modo emulado ou funcional, sendo que a escolha da configuração é feita pela interface visual do SIMSAT. As interfaces entre os subsistemas foram simplificadas para facilitar o projeto do software e simplificar a integração. A previsão inicial era desenvolver dois simuladores, um para operações de missão e outro para operações científicas, porém dado o curto prazo, as especificações foram mescladas e apenas um simulador com as duas funcionalidades foi desenvolvido.

A arquitetura SimGen (ROLET; CROENNE, 2006) pode ser usada no desenvolvimento de simuladores para sistemas espaciais e treinamento de operadores. Trata-se de um *framework* para criação de simuladores parametrizados que não faz uso do SIMSAT e apresenta uma arquitetura baseada em blocos escritos na linguagem Java, objetivando flexibilidade. Uma interface gráfica permite a geração de um simulador através da seleção dos blocos desejados. Permite, também, a geração de scripts para a execução da simulação. A arquitetura apresenta flexibilidade tanto na criação do simulador quanto na independência de hardware. A portabilidade é garantida, uma vez que é desenvolvido em Java.

O simulador LISA (*Laser Interferometer Space Antenna*) PathFinder (LPF) (DELHAISE; BRU, 2006) está sendo desenvolvido para a missão LISA (MCNAMARA; RACCA, 2009; ESA, 2012c), de detecção de ondas gravitacionais no espaço, cujo lançamento está previsto para 2014. Este simulador também utilizou o SIMSAT como núcleo básico. Da mesma forma que o CSIM, o simulador LPF faz uso dos modelos genéricos do comportamento térmico e elétrico e de dinâmica orbital, além da interface visual e de comunicação. Inclui interface para a estação terrena, para telemetrias e telecomandos. Como desenvolvimento específico inclui o gerenciamento das cargas úteis, o sistema de controle de rádio frequência e os modelos para transmissão de telemetrias e recepção de telecomandos.

Destoando dos outros simuladores analisados, o simulador para o Segmento Solo (STELLATO; ROMANI, 2005) tem como objetivo simular diferentes configurações de segmento solo, incluindo definição de cenários, execução do plano de voo e análise dos dados. Através da simulação, é possível avaliar questões como tamanho e crescimento da rede de dados, robustez do segmento solo para gerenciamento de falhas e mudanças na configuração, cenários de atuação (nominal, contingência, crise), desempenho em termos de propagação de órbita, levando-se em conta recursos do sistema e tempo de execução das atividades, treinamento de operadores para procedimentos de

manutenção. A arquitetura desse simulador é baseada em três camadas: (i) interface visual, (ii) definição e aplicação dos cenários e (iii) análise dos dados da simulação e biblioteca auxiliar. A interface apresenta ferramentas gráficas para definição dos cenários. Esta, por sua vez, obtém os parâmetros necessários à simulação, executa a simulação e disponibiliza os dados para análise posterior. A biblioteca auxiliar oferece ferramentas utilitárias utilizadas na preparação e execução da simulação bem como na análise dos dados da simulação.

Finalmente, o SIMC3, um simulador em desenvolvimento no INPE, para atender a missão CBERS3 (*China and Brazil Earth Resource Satellite 3*) (AMBROSIO et al., 2006), tem por objetivo treinar os operadores dos satélites CBERS 3&4, validar procedimentos de voo durante o ciclo de vida do satélite e validar o *software* de controle de satélites a ser utilizado no Centro de Controle de Satélites do INPE. Sua arquitetura inicial é dividida em três camadas: interface com o usuário, controle da simulação e modelos dos subsistemas. A interface com o usuário é responsável pelo provimento de recursos para início, pausa, reinício e fim de uma simulação, escolha da configuração inicial que o simulador irá executar, interação em tempo real com os valores dos parâmetros da simulação e visualização da evolução dos valores dos parâmetros simulados. O controle da simulação provê os mecanismos necessários no ambiente de simulação para execução do simulador, como controle de *threads*, temporização, controle de eventos, armazenamento das mensagens de *log*, controle de criação e comunicação entre os modelos, entre outros. Os modelos englobam os subsistemas do satélite a serem simulados, o ambiente espacial e as estações terrenas. Visando mitigar o custo de acesso a discos e banco de dados, todos os parâmetros da simulação são carregados para a memória no início da simulação. Desta forma, a arquitetura centraliza a comunicação entre modelos e o controle da simulação do tipo *blackboard,* onde uma área de memória centralizada comum é disponibilizada a todos os modelos e o controle de leitura e escrita é realizado pela camada de controle da simulação. Uma

análise preliminar de compatibilidade entre os serviços do simulador SIMC3 e o padrão SMP foi feita em (BARRETO et al., 2010).

O estudo mostrou que praticamente todos os simuladores no âmbito da ESA utilizam o software SIMSAT e, como este foi adaptado ao padrão SMP, também os simuladores tornaram-se compatíveis com o padrão. Considerando também que o SMP explicitamente promove reuso, portabilidade e interoperabilidade, os simuladores desenvolvidos sob o padrão beneficiar-se-ão destas características. Ressalte-se também que o estudo não encontrou solução específica para modelagem de OBDH para simuladores de satélite. O SIMSAT já possui um conjunto de emuladores que é utilizado pelos simuladores. Da mesma forma, nenhuma arquitetura mostrou uma solução envolvendo o padrão PUS.

Uma descrição mais completa destas arquiteturas encontra-se em (AMBROSIO; BARRETO, 2012).

2.2 Simulation Modelling Platform (SMP)

O SMP é um padrão proposto pela ESA para desenvolvimento de simuladores de satélite, visando promover a portabilidade, a interoperabilidade e reuso de modelos entre diversos ambientes e sistemas operacionais (ARGÜELLO et al., 2000). Inicialmente o padrão foi chamado de SMP1 (*Simulation Model Portability 1*). Todavia, uma série de limitações foi encontrada nesta versão, como a não utilização de orientação a objetos, falta de suporte a configuração dinâmica, mecanismos primitivos de escalonamento, etc; quando uma nova versão foi proposta para suprir as deficiências apontadas. Esta segunda versão passou a ser denominada de SMP2 e foi submetida à ECSS (*European Cooperation for Space Standardization*) como proposta para se tornar uma norma. Em janeiro de 2011, um Memorando Técnico foi disponibilizado pela ECSS sob a denominação de *Simulation Modelling Platform* (SMP) (ECSS, 2011). Este Memorando está baseado na norma ECSS-E-ST-40 que aplica os conceitos de engenharia de software (definição de requisitos, projeto,

produção, verificação e validação, transferência, operação e manutenção) nos segmentos de uma missão espacial (ECSS, 2009).

2.2.1 Visão Geral da Simulation Modelling Platform (SMP)

A arquitetura do padrão é decomposta em três camadas, sendo que a primeira camada representa o mundo real a ser modelado (*reality*), a segunda camada apresenta a especificação dos modelos independente de plataforma (PIM - *Platform Independent Model*), não levando em conta como os modelos serão desenvolvidos e executados, e a terceira camada apresenta os objetos já em plataforma específica (PSM - *Platform Specific Model*) para execução. A Figura 2.1 apresenta a visão de alto nível do SMP.

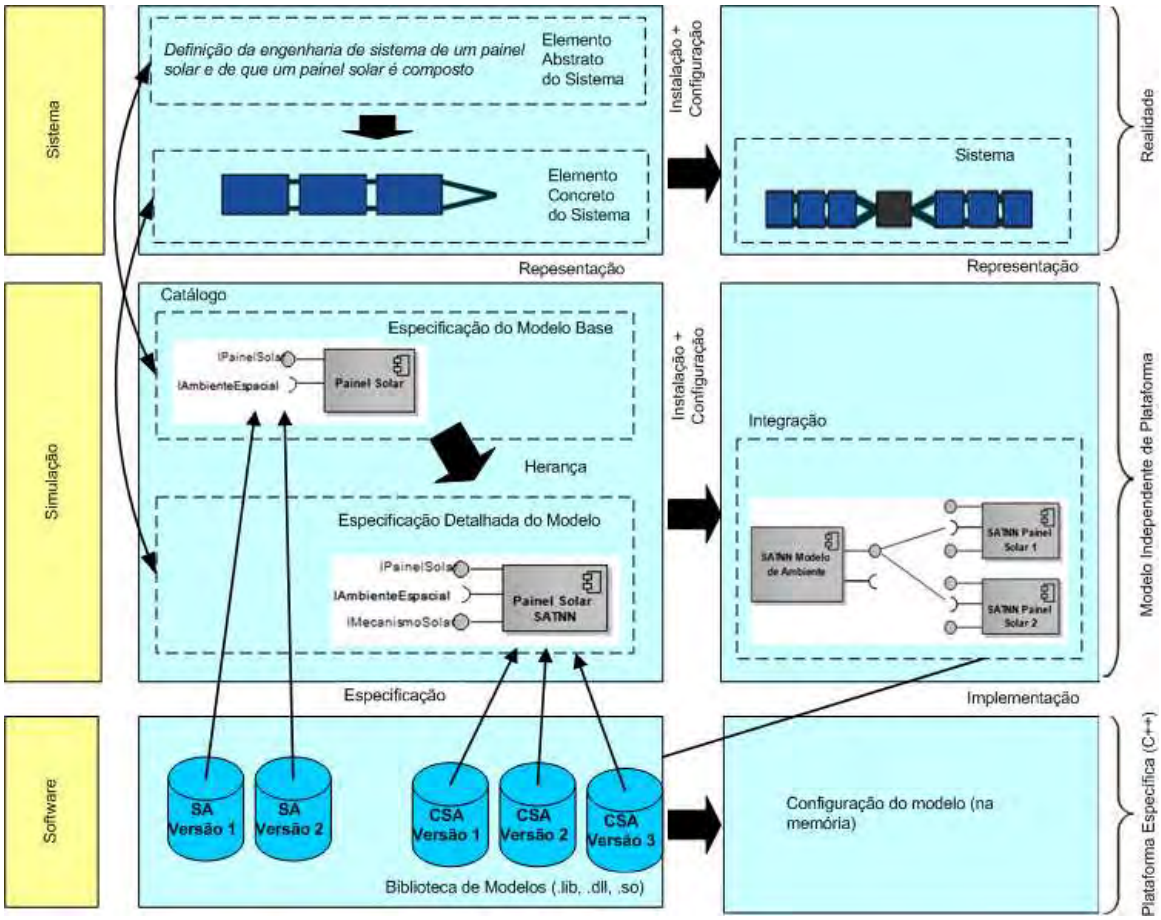


Figura 2.1 - Arquitetura do padrão SMP.
Fonte: Adaptada de ECSS (2011).

As duas últimas camadas utilizam o conceito de **modelo**. Um modelo é uma visão abstrata que ignora alguns detalhes de um sistema. A palavra **modelo** vem do latim *modulus*, que significa medida, regra, padrão, exemplo a ser seguido. Modelos estão baseados na idéia de abstração: um modelo não está relacionado a um objeto ou fenômeno particular, mas sim a muitos, ou seja, a um grupo ou uma classe. Para se diferenciar um modelo de um artefato são necessários três critérios (LUDEWIG, 2003; KÜHNE, 2005):

- Mapeamento: um objeto ou fenômeno original é mapeado para um modelo e passa a ser referenciado como o original. Este critério não implica a existência física do original, já que este pode ser planejado ou fictício (estimativa de custo de um software, por exemplo);

- Redução: nem todas as propriedades do original são mapeadas para o modelo. Uma vez que o modelo é reduzido ele deve conter apenas algumas propriedades do original;

- Pragmatismo: o modelo pode substituir o original para algum propósito, ou seja, o modelo é útil. O modelo é a informação de algo (conteúdo, significado), criado por alguém, destinado a alguém, para algum objetivo.

Na camada PIM, o painel solar exemplificado na camada real pode ser modelado tanto de forma genérica (contendo as funcionalidades básicas) quando especializado (herança) para uma determinada missão espacial (adicionando-se funcionalidades específicas). Na terceira camada se pode ter os componentes de software tanto para os modelos genéricos do painel quanto para os modelos especializados.

A Figura 2.1 apresenta também duas colunas sendo que a primeira corresponde às definições dos artefatos e a segunda às instâncias dos artefatos, tomando-se como analogia a idéia de Orientação a Objetos.

O objetivo principal do SMP é a promoção da independência de plataforma para os modelos. Esta independência é obtida pela definição de um conjunto de modelos, independentes de plataforma, que depois possam ser mapeados para uma plataforma específica.

Outros fundamentos usados no SMP, que reforçam o reuso e a portabilidade, são: (i) a separação entre projeto e execução (apresentado nas colunas da figura anterior); (ii) a configuração dinâmica; (iii) o uso extensivo de interfaces, componentes e herança (recurso de orientação a objeto).

O padrão SMP possibilita a criação de objetos através de uma configuração dinâmica, o que permite a troca de modelos em tempo de execução. Uma vez que as interfaces sejam idênticas, é possível substituir um modelo por outro de maior fidelidade ou até mesmo trocar a execução de um modelo de software que simula um hardware por um modelo físico de um equipamento real (*hardware in the loop*).

A arquitetura típica de um simulador baseado no padrão SMP possui uma camada para os modelos do satélite que serão simulados, uma camada de serviços obrigatórios e uma camada de controle da simulação (Ambiente Nativo de Simulação). A Figura 2.2 mostra esta arquitetura.

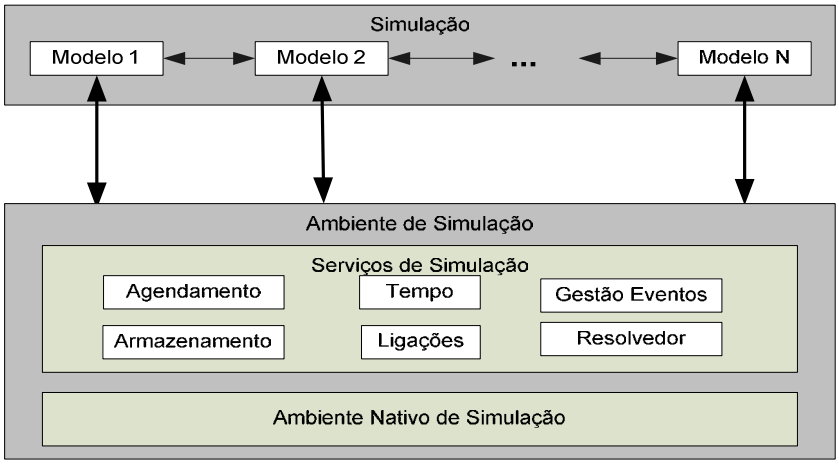


Figura 2.2 - Arquitetura típica de um simulador SMP.
Fonte: Adaptada de ECSS (2011).

A essência da simulação é representada pelos modelos, que representam os subsistemas ou equipamentos de um satélite. Os Serviços de Simulação fazem parte da camada Ambiente de Simulação, provendo facilidades para os modelos e para o ambiente, como temporização, agendamento, eventos, armazenamento, ligações entre componentes e serviço para resolução de nomes de diretórios. Ainda nesta camada de Ambiente, eventuais recursos nativos de simulação já existentes podem ser reaproveitados, através do encapsulamento dos mesmos, para que se adaptem ao padrão de interfaces do SMP.

A nomenclatura do SMP trata serviços de simulação e modelos de forma igualitária. Ambos derivam da interface IComponent. Também a interface de simulação é tratada como um componente. Todas as conexões entre componentes e entre serviços e componentes são realizadas através de um conjunto predefinido de interfaces. Os componentes estão vinculados entre si em uma árvore hierárquica, formada por composições de componentes. A interface de composição provê métodos para navegação através da hierarquia construída, o que permite recuperar todas as informações dos componentes das composições. A Figura 2.3 mostra um diagrama hierárquico dos componentes do SMP.

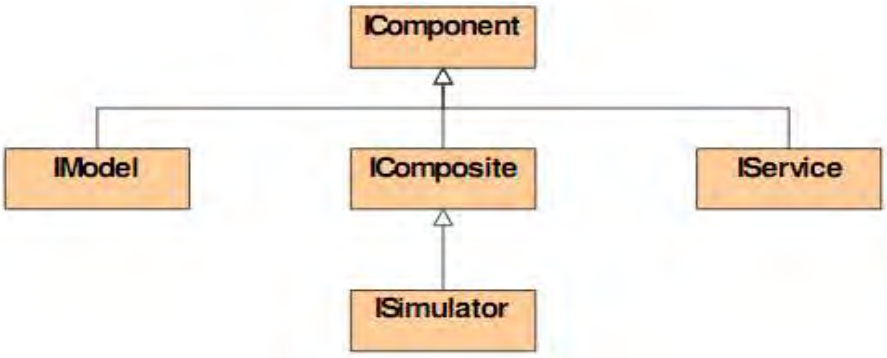


Figura 2.3 - Diagrama hierárquico de componentes do SMP.
Fonte: ECSS (2011).

Sob o ponto de vista de componentes, os modelos (interface IModel), os serviços (interface IService) e as composições de componentes (interface IComposite) se equivalem. A instância do simulador (interface ISimulator) é a base da hierarquia.

Os modelos, além da composição, podem se relacionar entre si também por referências (interface IReference). Uma composição é baseada em um container (interface IContainer) ao passo que uma referência é baseada em uma agregação (interface IAggregate). A Figura 2.4 mostra estas interfaces e seus relacionamentos.

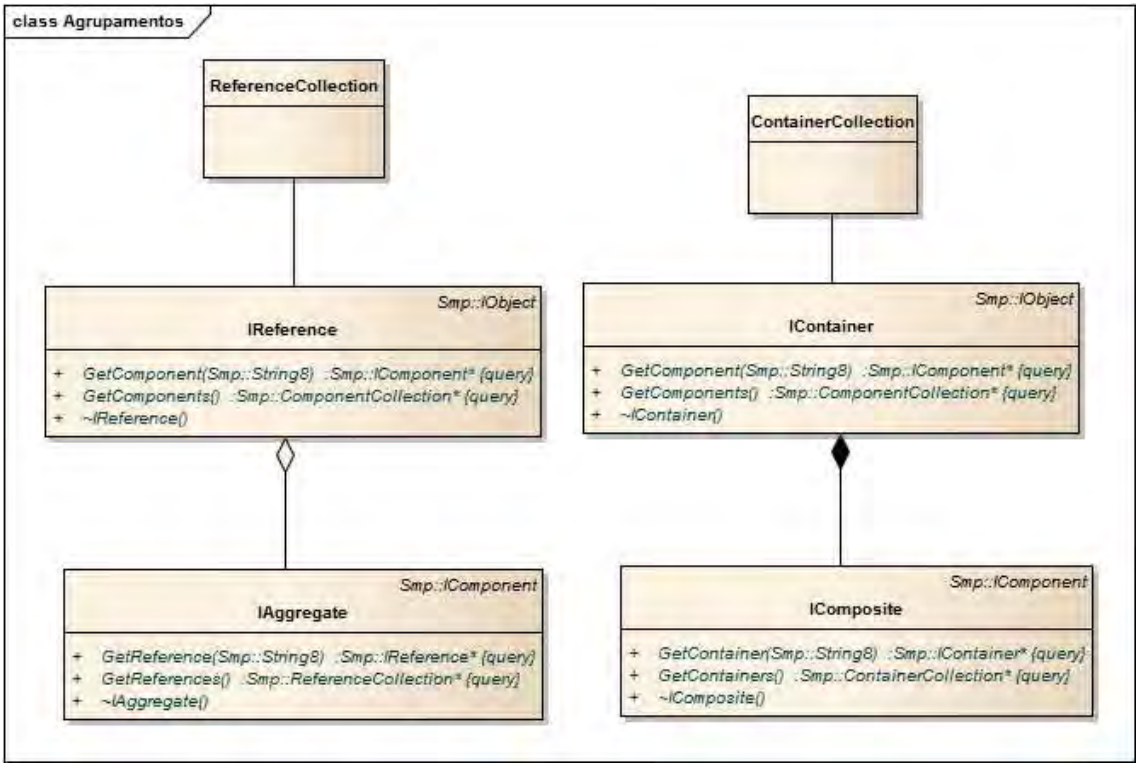


Figura 2.4 - Relacionamentos entre componentes.

O padrão também define seis serviços obrigatórios, derivados da interface IService.

A Figura 2.5 mostra o diagrama de classes dos serviços SMP.

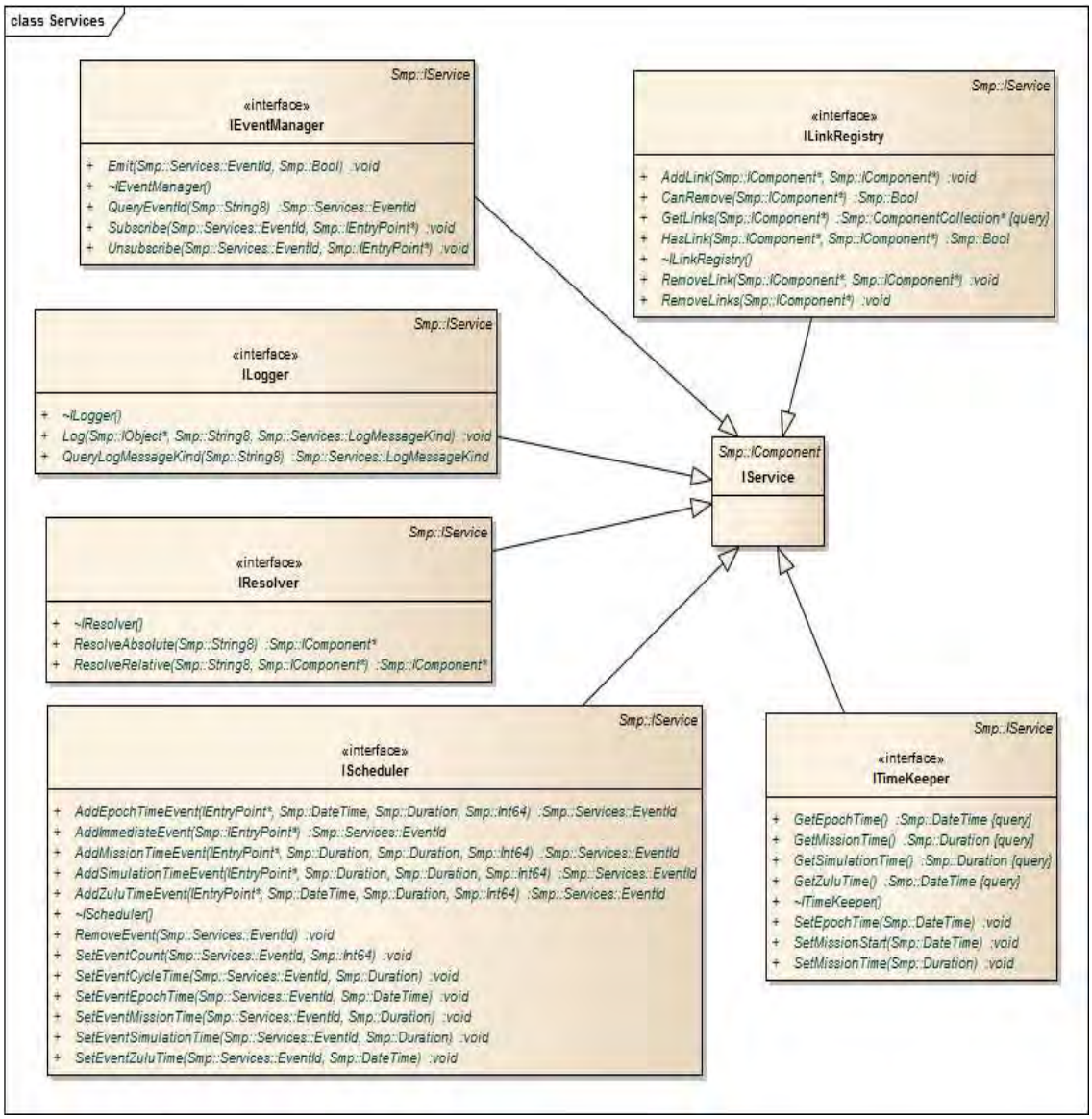


Figura 2.5 - Interfaces de serviços do SMP.

É possível a criação de serviços definidos pelo usuário, desde que implemente a interface padrão para serviços. Os serviços mandatórios são:

a) Temporização (ITimeKeeper): provê quatro tipos de tempo para a simulação - tempo da simulação, tempo da missão, tempo da época (*epoch time*) e tempo UTC (computador).

b) Gerência de Eventos (IEventManager): provê um mecanismo para disparo de eventos globais assíncronos.

c) Armazenamento (ILogger): armazena todas as mensagens geradas por modelos ou serviços.

d) Escalonamento (IScheduler): executa eventos de forma cíclica ou em intervalos de tempos específicos.

e) Resolver (IResolver): permite resolver as referências a outros componentes pelo nome. A referência pode ser tanto absoluta quanto relativa a outro componente.

f) Ligações (ILinkRegistry): mantém a lista de todos os vínculos entre os componentes da simulação, permitindo que a remoção dos mesmos seja feita de forma segura.

2.2.2 Mecanismos para inter-relacionamento entre componentes do SMP

O padrão SMP dispõe de mecanismos para inter-relacionamento entre componentes. São eles: Evento, *Entry-points*, Invocação Dinâmica, Publicação e Persistência.

O mecanismo de Evento permite a um dado modelo notificar outros modelos inscritos sempre que um evento interno for ativado durante sua execução. A Figura 2.6 mostra as interfaces envolvidas na manipulação de eventos.

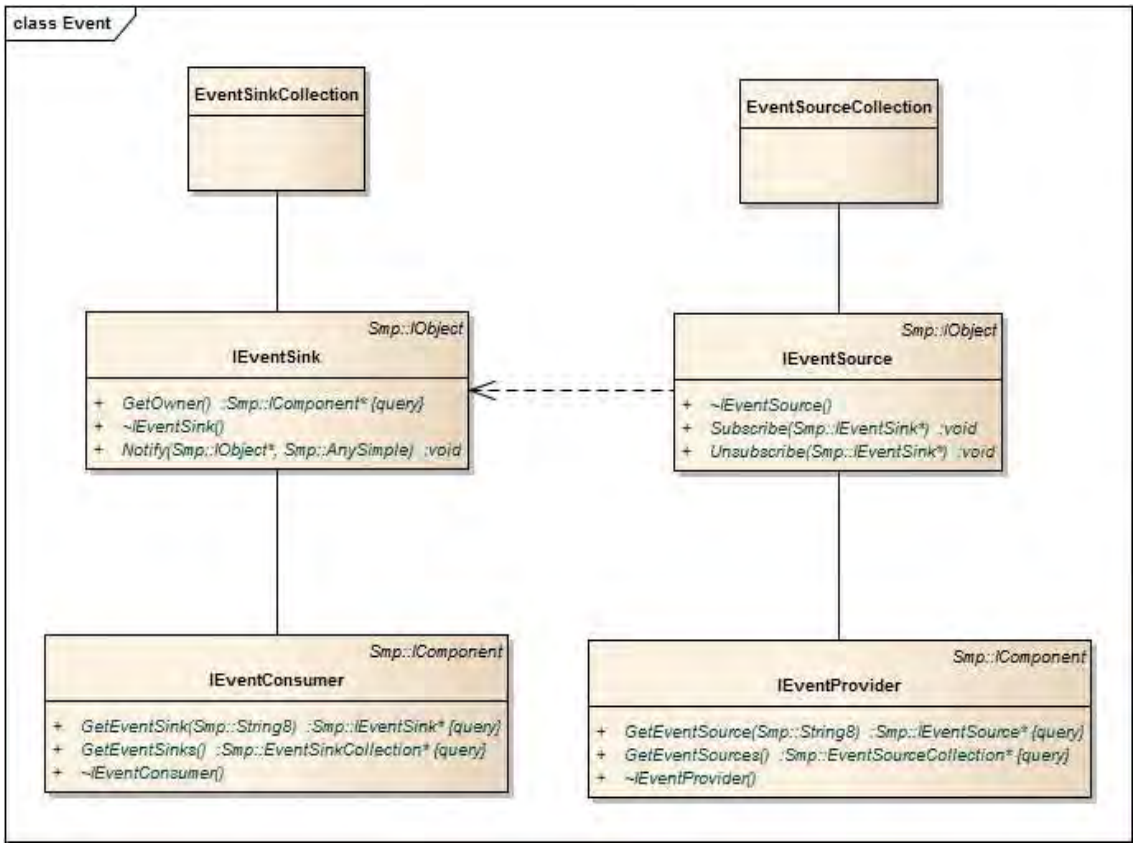


Figura 2.6 - Interfaces de Eventos de modelos.

Cada objeto EventSink poderá se inscrever em um objeto EventSource e será notificado toda vez que um evento for emitido por este. As interfaces IEventConsumer e IEventProvider são opcionais e servem para estender as funcionalidades das interfaces de eventos, permitindo consultar eventos pelo nome bem como obter as coleções de eventos.

O mecanismo de *Entry Point* (interface IEntryPoint) consiste em um método que não retorna nenhum valor e é utilizado pelo escalonador ou pelo gerenciador de eventos, quando um evento é emitido. A interface ITask permite agrupar uma coleção de *entry points* para serem executados conjuntamente. A Figura 2.7 apresenta as interfaces IEntryPoint e ITask.

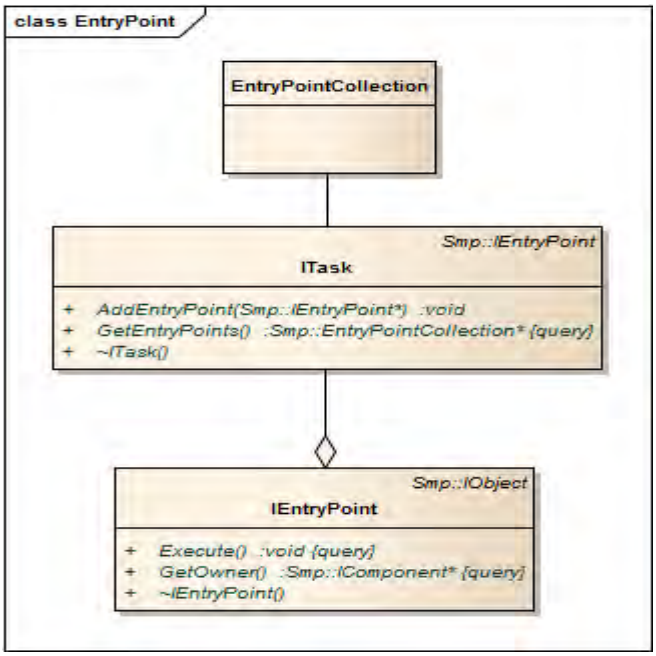


Figura 2.7 - Interfaces IEntryPoint e ITask.

O método Execute é disparado quando o *entry point* é acionado.

O mecanismo de Invocação Dinâmica disponibiliza os métodos de um modelo através de uma interface padrão (IRequest). Esta interface contém todas as informações necessárias à invocação de um determinado método bem como à obtenção do retorno da operação. A Figura 2.8 mostra as interfaces envolvidas.

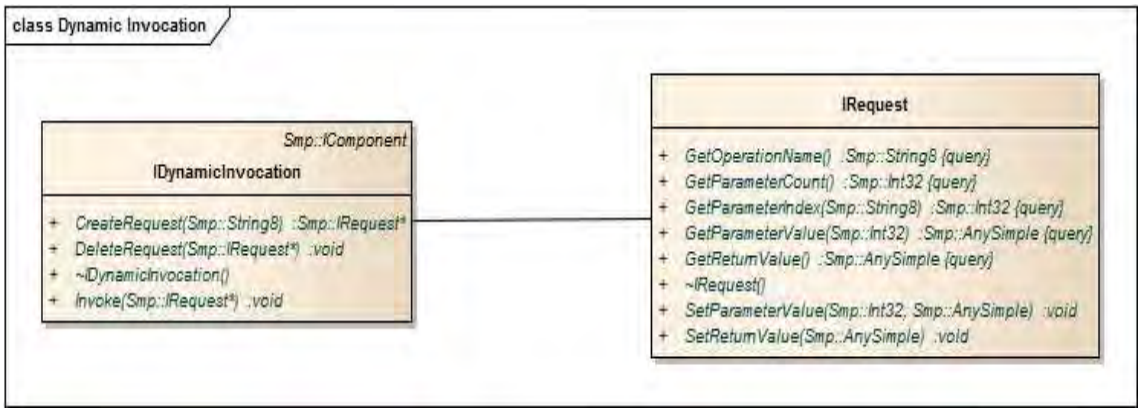


Figura 2.8 - Mecanismo de Invocação dinâmica.

Para utilização do objeto *Request* após o mesmo ter sido criado, é necessária uma chamada ao método *Invoke* da interface IDynamicInvocation, que, por sua vez, invoca a operação contida no objeto, permitindo a recuperação do valor da operação.

A Figura 2.9 mostra um diagrama de sequência do funcionamento do mecanismo de Invocação dinâmica.

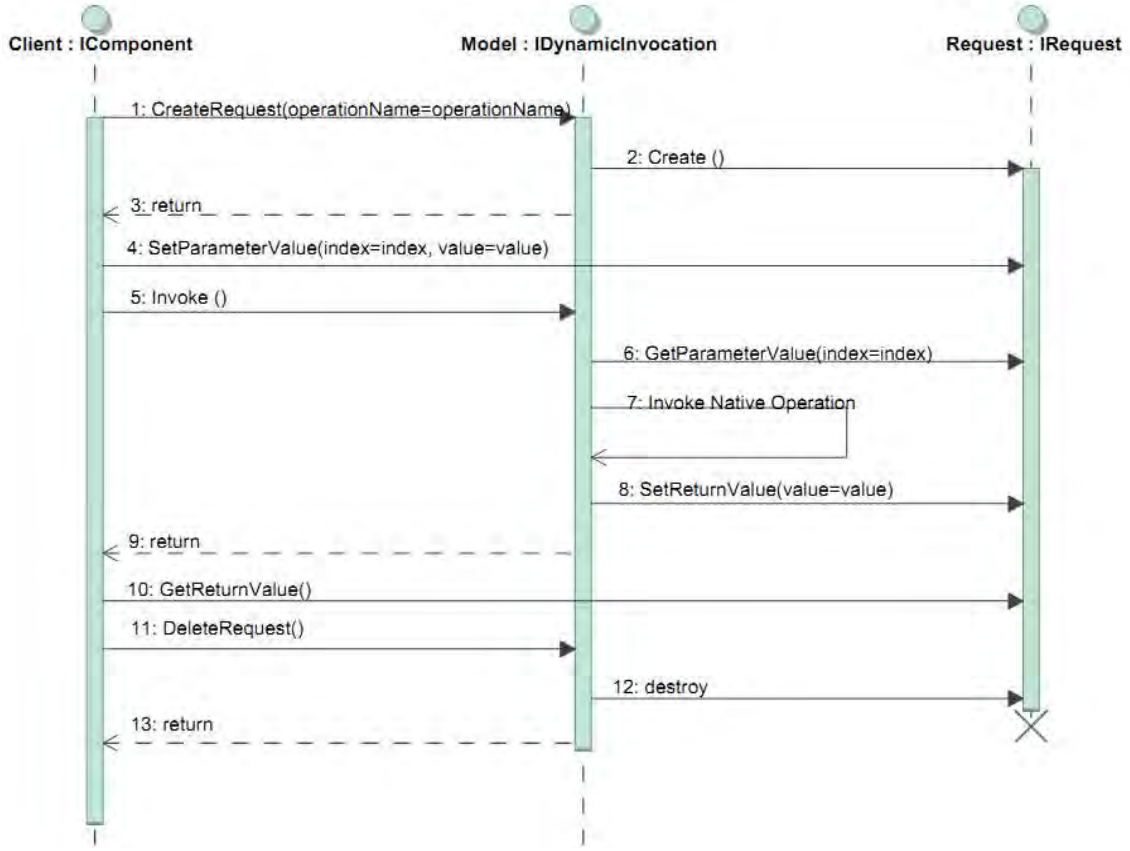


Figura 2.9 - Diagrama de sequência do mecanismo de Invocação Dinâmica.
Fonte: Adaptada de ECSS (2011).

Basicamente, um componente solicita a criação de uma requisição. A mesma é criada pela invocação dinâmica e retornada ao componente. Este aciona a invocação (método *Invoke*), que passa a requisição ao objeto da invocação. A requisição aciona o método do modelo, passando os parâmetros e recebendo o

retorno. O componente obtém o valor de retorno da operação da requisição e a destrói.

O ambiente de simulação também permite que os modelos publiquem seus métodos, propriedades ou membro de dados. O mecanismo de Publicação consiste em tornar disponível para os demais modelos e para o ambiente de simulação os referidos elementos internos de um determinado modelo. É possível definir o grau de visibilidade do elemento sendo disponibilizado. A interface IPublication define a forma de publicação de cada um destes elementos. A Figura 2.10 mostra a interface de publicação.

Figura 2.10 - Interface de publicação.

Com relação ao mecanismo de Persistência dos componentes, o padrão define duas formas: (i) externa, na qual o ambiente de simulação armazena e recupera os dados acessando diretamente os campos que são publicados pelos modelos, e (ii) auto persistência, na qual o próprio componente implementa a interface. Esta última opção é aplicada a modelos especializados, que devem ter o mecanismo independente do ambiente. A Figura 2.11 mostra as interfaces IPersist, IStorageReader e IStorageWriter, responsáveis pelo mecanismo de Persistência.

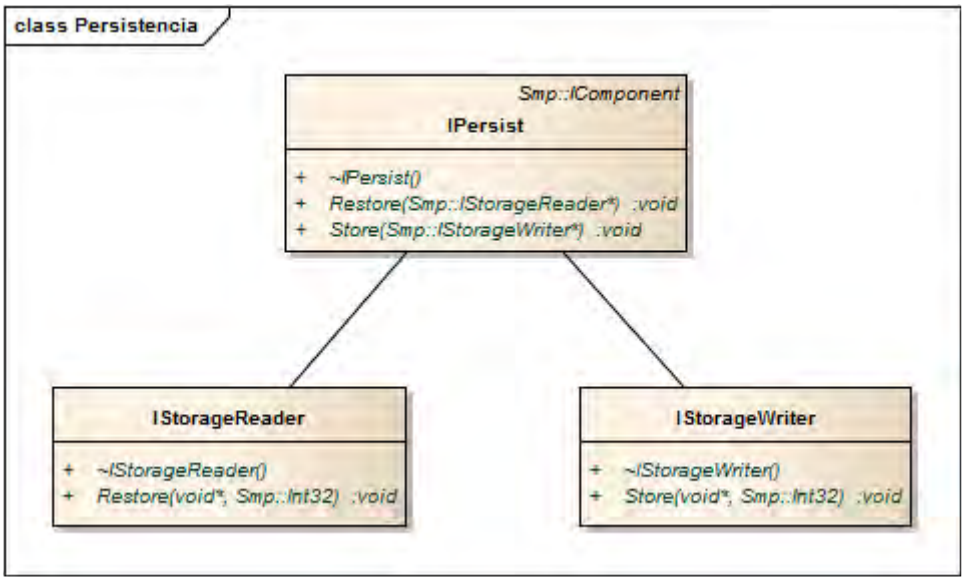


Figura 2.11 – Mecanismo de Persistência de dados.

A persistência externa (através dos métodos *Store* e *Restore* da interface ISimulator) é mandatória para todo simulador no padrão SMP. As interfaces IStorageReader e IStorageWriter são específicas da plataforma e disponibilizadas pelo ambiente de simulação para os modelos.

2.2.3 Criação do ambiente de simulação

A criação do ambiente de simulação, incluindo os componentes e seus relacionamentos para execução da simulação, pode ser feita estática ou dinamicamente.

Na criação estática do simulador, todos os passos para montagem do ambiente de simulação são feitos com base em código fixo. Ou seja, qualquer modificação que venha a ser feita posteriormente implicará em modificação no código de criação e em recompilação.

A criação dinâmica permite que os passos para a criação do ambiente de simulação sejam definidos em arquivos externos, escritos em uma linguagem própria, baseada em XML (XML, 2006). Desta forma, os passos da criação podem ser modificados em tempo de execução. Essa facilidade permite substituir, por exemplo, um dado componente, em tempo de execução. O SMP usa a linguagem de definição de modelos para simulação SMDL (*Simulation Model Definition Language*) que permite criar catálogos de modelos, relações de dependências entre modelos, escalonamentos, integração, empacotamento e documentação.

Cada um destes catálogos contém as seguintes informações:

a) Modelos: contém as definições dos modelos, seus tipos e atributos. Os modelos podem estar aninhados, formando uma hierarquia;

b) Dependências: as instâncias dos modelos estão conectadas, usando interligações baseadas em regras e restrições definidas para os modelos;

c) Escalonamentos: define como as instâncias dos modelos serão escalonadas para execução, tanto com base em tempos pré definidos quanto em eventos cíclicos. Define também as tarefas que serão disparadas quando das execuções;

d) Integração: define quais instâncias ou tipos serão integrados em uma biblioteca ou um arquivo binário, por exemplo. Embora normalmente haja apenas uma instância de um mesmo modelo, outras poderão ser criadas, permitindo, assim, diferentes versões de um mesmo modelo;

e) Documentação: refere-se ao ambiente a ser criado por estes catálogos. Seu objetivo é interligar todos os catálogos para criação do ambiente de simulação.

2.3 O Framework OBS

O *framework* OBS (*On Board Software*) foi proposto por (P&P SOFTWARE GMBH, 2003) para desenvolvimento de aplicações voltadas para computadores de bordo de satélites, mais precisamente para suporte às funções de controle de órbita e atitude (CECHTICKY et al., 2002). Na área espacial, este subsistema é conhecido como AOCS (*Attitude and Orbit Control System*).

Este framework não foi desenvolvido visando especificamente simuladores de satélites, como o padrão SMP. Todavia, uma série de similaridades pode ser observada entre este *framework* e o SMP. Ambos fazem uso de interfaces abstratas visando padronizar o desenvolvimento de artefatos e incrementar tanto o reuso como a interoperabilidade. O *framework* OBS possui um conjunto de interfaces na forma de padrões de projeto. Enquanto que o SMP procura cobrir todos os tópicos necessários ao desenvolvimento de simuladores de satélites, o OBS foca em aplicações para computação de bordo.

2.3.1 Visão Geral do framework OBS

O *framework* OBS é composto por:

a) um catálogo de Padrões de Projeto (GAMMA et al., 1997) que oferece soluções otimizadas para problemas recorrentes em *software* de bordo, formando um arquitetura configurável;

b) um conjunto de interfaces abstratas que permite adaptar pontos da arquitetura configurável de acordo com necessidades específicas;

c) um conjunto de componentes concretos para suporte à implementação da arquitetura configurável;

d) um Gerador de Meta-Componentes que permite que implementações específicas das interfaces abstratas sejam automaticamente geradas de uma especificação de auto nível;

e) um conversor de meta-componentes que permite a modificação automática de componentes concretos para sejam compatibilizados com propriedades específicas da aplicação;

f) um conjunto de componentes que garantem a compatibilidade do *framework* com o padrão PUS;

g) ferramentas de teste que executam testes unitários em todo o *framework*, gerando relatórios automaticamente;

h) uma interface entre o *framework* OBS e os modelos gerados pelo Simulink (MATHWORKS, 2011);

i) um dicionário de dados que define um conjunto de termos para facilitar a descrição das aplicações;

j) um conjunto de dados para dar suporte à qualificação das aplicações instanciadas a partir do *framework* OBS.

O Catálogo de Padrões de Projeto contém os elementos de construção disponibilizados pelo *framework* OBS. Este catálogo disponibiliza padrões de projeto para solução de problemas típicos no desenvolvimento de um sistema de software para computadores de bordo, contando, atualmente, com vinte padrões.

A Figura 2.12 mostra o exemplo do padrão TelemetryStream, do catálogo.

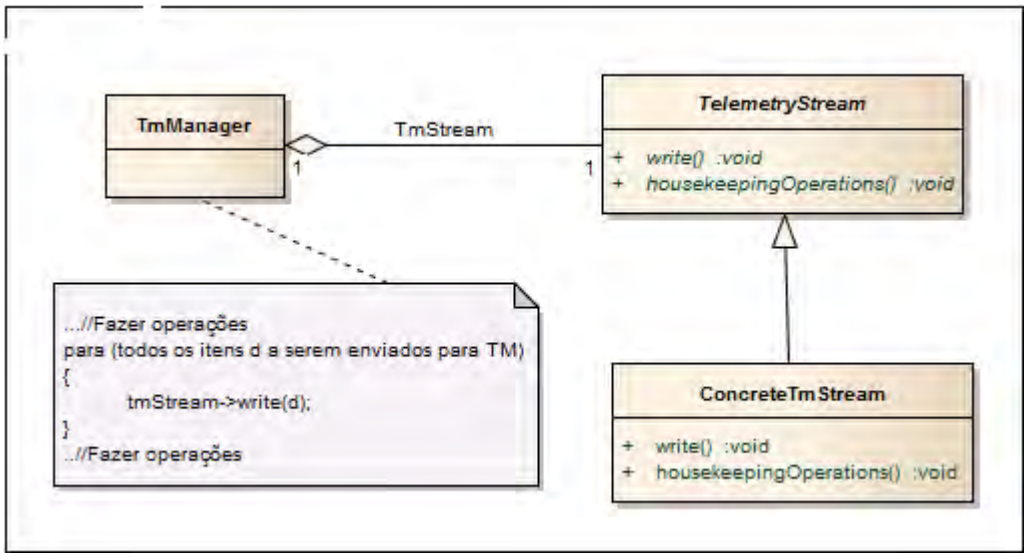


Figura 2.12 - Padrão TelemetryStream.
Fonte: Adaptada de P&P Software GMBH (2003).

O padrão é composto de uma interface TelemetryStream que define as operações que um componente TmManager pode executar. Este, por sua vez, contém uma referência a um componente ConcreteTmStream, através da interface abstrata. Isto faz com que o componente concreto atue como um *plugin*, podendo ser substituído sem impacto no código do TmManager, já que a interface se mantém inalterada.

As interfaces abstratas definem um conjunto de serviços que devem ser previstos pelo *framework*, porém a forma de implementar é livre. Algumas interfaces possuem uma implementação através dos componentes concretos que podem ser de dois tipos: componente do núcleo e componente padrão. Os componentes do núcleo encapsulam comportamentos que não se modificam para as aplicações no domínio do *framework*. Já os componentes padrões, cuja implementação sugere um comportamento padrão, podem ser modificados pelas aplicações que os utilizarem.

O gerador de Meta-Componentes adapta os elementos disponibilizados pelo *framework* aos requisitos da aplicação a ser desenvolvida. A Figura 2.13 mostra a geração dos Meta-componentes.

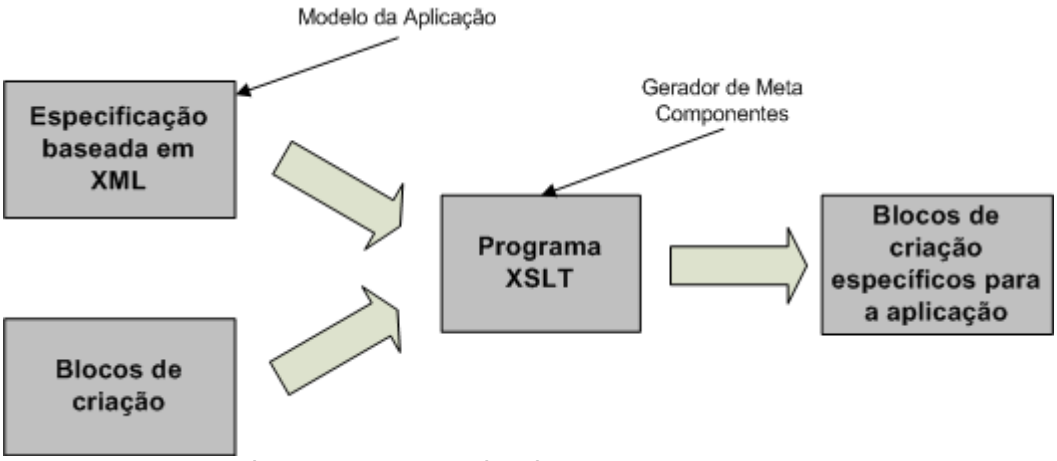


Figura 2.13 - Gerador de Meta-componentes.
Fonte: Adaptada de P&P Software GMBH (2003).

O Gerador de Meta-Componentes é formado por programas escritos na linguagem XSL (XSL, 2006), que processam especificações escritas em XML e geram código para um componente específico de uma aplicação ou para configuração de um cluster de componentes.

2.3.2. Criação de aplicações no ambiente do *framework*

O trabalho de (P&P SOFTWARE GMBH, 2003) associa ao *framework* um desenvolvimento baseado em dois processos: o de criação e o de instanciação. A Figura 2.14 mostra as etapas dos dois processos.

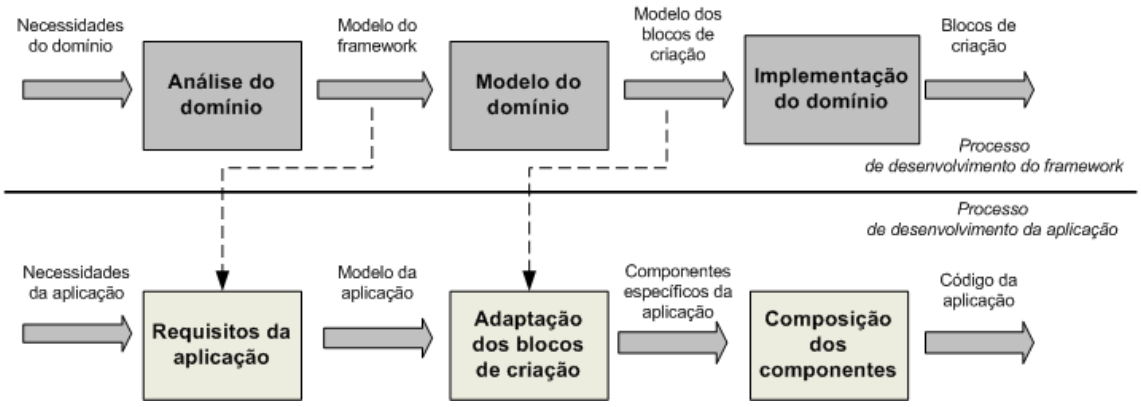


Figura 2.14 - Processo de criação de aplicações.
Fonte: Adaptada de P&P Software GMBH (2003).

O processo de criação pode ser dividido em três fases:

a) Análise do domínio - o domínio de objetivo é caracterizado e um modelo formal do domínio é produzido;

b) Projeto - os elementos reusáveis para os quais as instâncias do *framework* serão construídas são identificados e modelados;

c) Implementação - os modelos dos elementos reusáveis são codificados.

No processo de instanciação, os elementos de criação são adaptados às necessidades da aplicação dentro do domínio do *framework* e utilizados na instanciação da aplicação, pela composição dos componentes desenvolvidos.

Um gerador de meta-componentes é usado no processo de adaptação dos elementos disponibilizados pelo *framework* para compatibilização com os requisitos da aplicação a ser desenvolvida.

Juntamente com o código gerado, também casos de testes podem ser gerados automaticamente. A Figura 2.15 ilustra o processo.

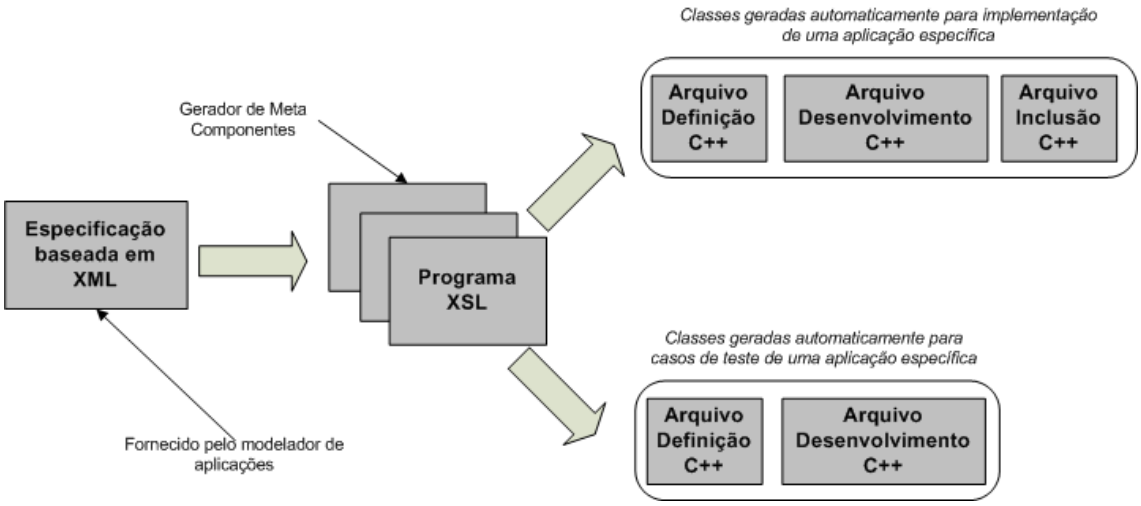


Figura 2.15 - Geração automática de códigos e casos de teste.
Fonte: Adaptada de P&P Software GMBH (2003).

Da mesma forma que na Figura 2.13 (gerador de meta-componentes), partindo-se de uma especificação em linguagem XML, um gerador XSL gera automaticamente os arquivos da implementação e os arquivos de casos de teste para a aplicação, a ser incluída na ferramenta de testes para a aplicação.

2.4 Comparação entre SMP e o *Framework* OBS

Há semelhanças consideráveis entre o padrão SMP e o *framework* OBS, como o uso de interfaces abstratas, de classes concretas e de uma linguagem de script para manipulação dos artefatos.

No *framework* OBS alguns componentes concretos são implementados enquanto que no SMP não. Entretanto, nota-se a necessidade dos componentes concretos ao se implementar o padrão SMP. A solução dada no SMP para esta carência foi a inclusão de um *kit* de desenvolvimento responsável pela criação dos componentes concretos, portados para uma determinada linguagem. Este *kit* torna o desenvolvimento mais ágil, pois já contém a implementação das interfaces, incluindo eventuais dependências hierárquicas. Desta forma, a criação de novas classes baseadas em uma determinada interface permite que se herde da classe concreta da mesma, ao invés de implementar todos os métodos requeridos.

Ambas as soluções, cada uma à sua maneira, dispõem de linguagens de script estruturadas em XML, para geração e criação dos artefatos. O SMP, diferentemente do OBS, não possui mecanismos para geração de casos de teste nem solução adaptada para o padrão PUS. Por outro lado, apresenta um arsenal de soluções bastante amplo e muito mais completo para desenvolvimento de simuladores de satélites.

3 SUBSISTEMAS DE COMPUTAÇÃO DE BORDO PARA SIMULADORES

Este capítulo apresenta as principais características de um subsistema de computação de bordo de um satélite e detalha o padrão denominado *Packet Utilization Standard (*PUS).

3.1 Subsistema de computação de Bordo

Os computadores a bordo de satélites (*On Board Computer* - OBC) são responsáveis por diferentes funções, entre elas a supervisão de bordo (FRANCISCO, 2003). As atividades englobadas pela supervisão incluem gerenciamento de subsistemas, processamento de comandos, monitoração dos equipamentos, gerência de cargas úteis e de outros subsistemas, além dos protocolos para a correta comunicação com cada equipamento ou subsistema.

O subsistema de computação de bordo de um satélite executa a supervisão de bordo e é conhecido como OBDH (*On Board Data Handling*). Contém uma complexa arquitetura de hardware e software para comunicação com a maioria dos subsistemas de bordo e com os subsistemas do Segmento Solo, como as estações terrenas e o Centro de Controle de Satélites. Quando o satélite não está comunicável com os subsistemas em solo, cabe ao OBDH tomar decisões a bordo, baseadas na análise dos dados obtidos dos subsistemas.

Basicamente, duas funções são fundamentais em um computador de bordo: Controle e Monitoração.

A função de Controle, a qual pode também ser encontrada na literatura como Comando e Controle, permite a configuração remota dos subsistemas tanto da plataforma (satélite propriamente dito) quanto das cargas úteis. O controle dos subsistemas é realizado pela execução de telecomandos. Mesmo quando o satélite não está em comunicação com o Segmento Solo, o OBDH pode atuar, através dos comandos temporizados, corrigindo alguma falha, ligando ou desligando equipamentos, etc.

A função de Monitoração é responsável pela coleta dos dados gerados pelos equipamentos a bordo. Os dados coletados podem ser de status de um equipamento (nesse casso são chamados dados de *housekeeping*) ou dados obtidos pelas cargas úteis (por exemplo, dados científicos, imagens, etc...). Quando o satélite está na região de visibilidade da antena de uma estação terrena de recepção, tanto os dados de *housekeeping* como os dados das cargas úteis são recebidos em solo. Os dados coletados, que são transmitidos aos sistemas de solo, são chamados telemetrias. Estes dados podem ser armazenados temporariamente durante a órbita do satélite até que o mesmo torne-se visível à estação terrena de recepção.

A descrição das principais funções de um computador de bordo pode ser encontrada em (AYYAZ et al, 2008; dos SANTOS, 2008). Segundo o autor, as principais funções de um subsistema de computação de bordo podem ser divididas conforme ilustrado na Figura 3.1.

Figura 3.1 – Principais funções de um OBC.
Fonte: Adaptada de Ayyaz et al (2008).

As funções de um subsistema de computação de bordo mostradas na figura 3.1. podem variar em complexidade. Pode-se ter desde um subconjunto de funções básicas como obtenção de telemetria e execução de telecomando, até um conjunto mais complexo que inclua a interação com o subsistema de controle de atitude e órbita ou incorpore suas funções.

3.2 Packet Utilization Standard (PUS)

Levando em consideração que o subsistema OBDH (*On Board Data Handling)* de todo satélite artificial terá um conjunto semelhante de funções, a Agência Espacial Européia (ESA), no contexto da ECSS (*European Cooperation for Space Standardization*), elaborou um padrão para um conjunto possível de funções comuns de um OBDH, denominado *Packet Utilization Standard* (PUS).

O PUS descrito no documento ECSS-E-70-41-A (ECSS, 2003) estabelece uma interface, em nível de aplicação, entre o Segmento Solo e os processos executados no computador de bordo. Esta interface é baseada em um conjunto padronizado de serviços cuja implementação é definida na fase de projeto da missão, de acordo com os requisitos de interface elétrica, testes e operações de voo. O padrão define dezesseis serviços responsáveis pelas funcionalidades do software do computador de bordo de um satélite. Normalmente apenas um subconjunto de serviços é selecionado para uma dada missão. A implementação destes serviços pelo computador de bordo requer aplicações em solo para monitoração e controle dos dados, capazes de enviar e receber dados formatados no padrão PUS.

As Figuras 3.2 e 3.3 apresentam todos os serviços do padrão PUS.

Verificação de telecomando
Verificação do telecomando em cada estágio da sua execução, desde a aceitação até a sua completa execução. Deve emitir no mínimo dois relatórios sobre a aceitação do telecomando: sucesso ou falha. Emite relatórios adicionais durante fase de aceitação e execução

Distribuição de comandos a dispositivos
Responsável pelo tratamento de comandos On-Off, Carga de Registrador e CPDU (Command Pulse Distribution Unit). Para os dois primeiros tipos, o serviço deverá redistribuir os mesmo para os respectivos dispositivos. Para o tipo CPDU, o comando deverá ser enviado à respectiva unidade, que emitirá pulsos para reconfiguração de funções vitais do satélite

Relatório de manutenção e diagnóstico
Responsável por fornecer informações relativas a housekeeping e diagnóstico. Juntamente com o relatório de estatística de parâmetros e eventos, fornecem todas as informações operacionais

Relatório estatístico de parâmetros
Responsável pela geração de relatórios contendo os valores máximo, mínimo e médio do desvio padrão dos parâmetros on board , durante um intervalo de tempo, notadamente quando o satélite encontra-se fora de visibilidade pela estação terrena

Relatório de eventos
Responsável por prover relatórios com dados não cobertos por outros serviços. Entre as informações, pode-se encontrar falhas ou anomalias detectadas on board, ações autônomas executadas à bordo e progresso normal de operações e atividades

Gerenciamento de memória
Responsável por prover funcionalidades de gerenciamento de memória, como carga, descarga e checagem do conteúdo da memória, seja em blocos contíguos ou não contíguos

Gerenciamento de funções
Provê suporte a funções de software que não são implementadas como padrão de missões mas que podem ser controladas a partir do solo, como o controle da operação de uma carga útil

Gerenciamento do tempo
Provê meios para o controle da taxa de geração de relatórios do tempo de referência do satélite

Figura 3.2 - Lista de serviços do padrão PUS - Parte 1.
Fonte: Adaptada de ECSS (2003).

Escalonamento de bordo
Controlar a fila de comandos temporizados em bordo e garante a execução dos mesmos nos tempos para os quais foram escalonados. Entre as atividades do serviço estão a inclusão e exclusão de telecomandos da fila e o deslocamento da execução dos mesmos

Monitoração de bordo
Serviço responsável pelo controle da lista de monitoração de parâmetros onboard. A lista de monitoração é definida pelo solo e o serviço pode gerar um relatório informando quais parâmetros saíram dos limites previstos

Transferência de grandes dados
Responsável pelo suporte a transferência de grande quantidade de dados. O serviço deve prover um mecanismo de quebra dos grandes pacotes de dados, permitindo seu envio

Controle do envio de pacotes
Encaminhamento dos pacotes de telemetrias para o solo

Armazenamento e recuperação de bordo
Responsável por oferecer suporte à seleção de pacotes armazenados, para serem recuperados. O serviço permite a escolha dos pacotes armazenados a serem baixados e, em seguida, executa a recuperação dos dados destes pacotes selecionados

Serviço de Testes
Responsável por ativar testes implementados em bordo e gerar relatórios dos mesmos. À exceção do teste de conectividade solo bordo, os outros testes são específicos de determinada missão

Procedimento das operações a bordo
Responsável por prover chamadas padrão aos procedimentos definidos em solo, monitorar os estados e gerar relatórios da execução destes procedimentos

Ação e eventos
Responsável por definir uma ação que é executada autonomamente quando um evento é detectado

Figura 3.3 - Lista de serviços do padrão PUS - Parte 2.
Fonte: Adaptada de ECSS (2003).

Cada um dos serviços apresentados possui um identificador único, caracterizado como tipo do serviço, podendo ser especializado em atividades

específicas identificadas igualmente de forma única por um subtipo. Um conjunto de aplicações identificadas individualmente por um APID (*Application Process Identification*) é executado no computador de bordo e pode fornecer dados (telemetria) mediante requisições (telecomando). O padrão define a estrutura destes pacotes (telemetria e telecomando).

A título de exemplificação, a Figura 3.4 mostra a estrutura do pacote de telemetria para um serviço do PUS.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Source Data	Spare (Optional)	Packet Error Control (Optional)
Version Number (=0)	Type (=0)	Data Field Header Flag	Applica-tion Process ID	Grouping Flags	Source Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	(see Note 2)

Figura 3.4 - Estrutura do pacote de telemetria do padrão PUS.
Fonte: Adaptada de ECSS (2003).

O cabeçalho do pacote (*Packet Header*) contém os dados de identificação (*Packet ID*), da sequência de controle (*Packet Sequence Control*) e o comprimento do mesmo (*Packet Length*). Esta seção é fixa. O segmento de dados do pacote (*Packet Data Field*) é variável e contém os dados capturados em bordo e formatados para o segmento.

O cabeçalho do segmento de dados (*Data Field Header*) contém as informações de tipo e subtipo dos serviços, que devem ser únicos para uma determinada missão. No pacote de telemetria, estes campos indicam a origem dos dados sendo enviados (tipo e subtipo responsáveis pelos dados). Já em um telecomando, os referidos campos indicam o destino (serviço e atividade) que o telecomando deseja acionar.

O OBDH foi, portanto, o subsistema escolhido para aplicação do trabalho. Assim sendo, os serviços do padrão PUS orientaram a proposta de decomposição do modelo de OBDH para um simulador de satélites. A arquitetura proposta nesta dissertação, a ser apresentada, considerou a forma de comunicação interna e externa do OBDH, através dos serviços PUS integrantes de um OBDH.

4 ARQUITETURA FLEXÍVEL PARA UM MODELO DE OBDH

Este capítulo apresenta a arquitetura proposta nesta dissertação para um Modelo de OBDH para Simuladores de Satélites. A arquitetura proposta é baseada no padrão SMP (ver seção 2.2), no padrão PUS (ver seção 3.2) e no conceito de Componentes de Software.

O termo **flexível** é usado aqui com o significado de que a arquitetura deve permitir qualquer número de componentes para a composição de um Modelo de OBDH.

4.1 Descrição geral da arquitetura

Um Modelo de Subsistema de OBDH para um simulador de satélites deve executar todos os serviços do Subsistema de OBDH de um satélite real.

Considerando que um modelo é uma visão abstrata do elemento real (ver definição na seção 2.2.1), um modelo de OBDH representará uma visão em alto nível das funções desempenhadas por um OBDH real. Se a modelagem é feita com base em serviços comuns e reusáveis, então o modelo pode representar OBDHs que poderão ser usados em diferentes simuladores de satélites.

Nesta proposta o Modelo do subsistema de OBDH é dividido em **componentes**, sendo que cada componente representa um serviço do PUS ou um outro serviço específico, não previsto no PUS. Assim, pode-se criar Modelos de OBDHs combinando-se quaisquer conjuntos de serviços. Os componentes que implementam os serviços (ou especializações dos mesmos) podem também possuir diferentes versões, o que pode modificar sua interação com os demais componentes durante a troca de dados. Além disso, cada componente pode ser decomposto em outros componentes, dependendo da complexidade do mesmo. Por exemplo, o serviço de Relatório e Estatística de Parâmetros do PUS (ver seção 3.2), pode incluir todos os tipos de estatísticas

previstos (valor médio, desvio padrão, valor máximo e mínimo) ou apenas alguns deles.

Tais serviços incluem ou exigem comunicação interna entre os mesmos e comunicação externa entre o OBDH e os demais subsistemas do satélite e comunicação com os sistemas em solo.

A fim de ilustrar a idéia apresentada, a Figura 4.1 mostra exemplos de serviços básicos como verificação de telecomando, monitoração de bordo, de serviços específicos, como serviço específico 1, e indica interfaces internas e externas com os outros subsistemas do satélite e com os subsistemas do segmento de solo.

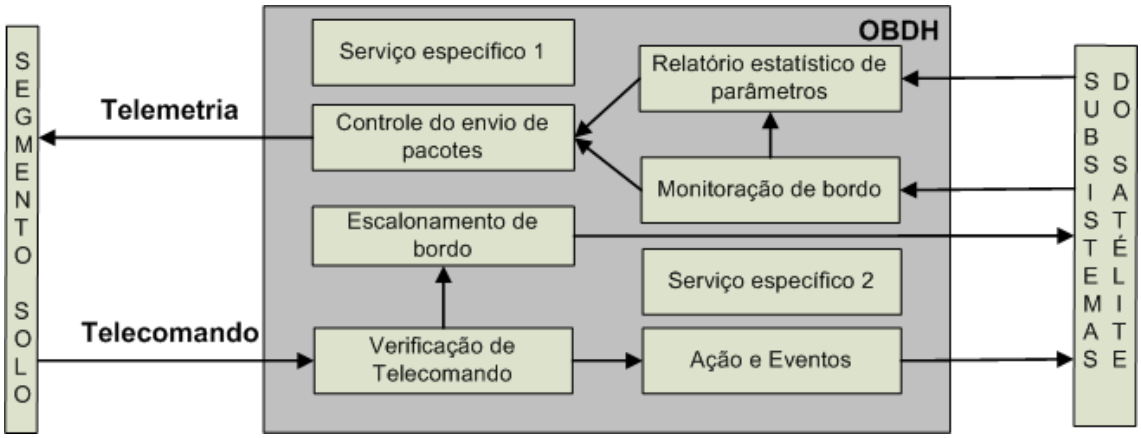


Figura 4.1 - Interfaces interna e externa de um OBDH.

A comunicação entre os componentes de serviço do Modelo OBDH, na arquitetura proposta faz uso da solução apresentada no SMP, ou seja, usa as interfaces de troca de dados e comunicação: IEventSource, IEventSink, IAnySimple, IPublication, IRequest (ver seção 2.2), além dos mecanismos de inter-relacionamento entre os componentes.

Desta forma, um Modelo de OBDH consta de um conjunto de componentes de software (implementando os serviços) que pode ser aumentado ou modificado

de acordo com o Subsistema de OBDH real, mesmo que este não implemente o padrão PUS.

As possíveis combinações de serviços compõem uma **Configuração de OBDH**. Desta forma, um modelo de OBDH consta de uma configuração de OBDH.

Diferentes simuladores podem usar diferentes modelos de OBDHs (um modelo em cada simulador).

A Figura 4.2 ilustra o contexto de aplicação da arquitetura flexível. Em 4.2(a) encontram-se os serviços disponíveis, incluindo aqueles previstos no padrão PUS; em 4.2(b) pode-se observar diferentes configurações de OBDHs. Cada OBDH é composto de um ou mais serviços, encapsulados em componentes. Em 4.2(c) observam-se exemplos de dois simuladores com destaque ao modelo OBDH que eles incorporaram. Cada simulador tem apenas um modelo OBDH.

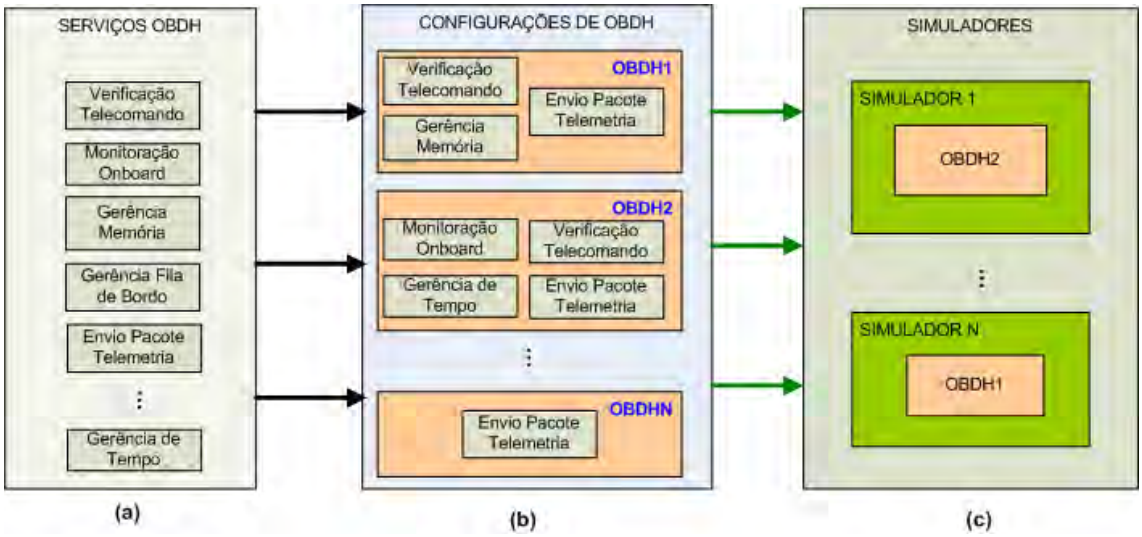


Figura 4.2 - Diagrama de contexto da arquitetura flexível para um Modelo de OBDH.

A interoperabilidade e flexibilidade são materializadas na letra (b) da Figura 4.2, pois é onde ocorrem a inclusão, exclusão ou troca dos serviços em uma determinada configuração de OBDH.

4.2 Definição das interfaces e dos mecanismos de inter-relacionamento

Cada componente pode ter associado a ele um conjunto de entradas e outro de saídas de dados. Cada entrada ou saída é mapeada para os mecanismos de comunicação do SMP.

Ao incluir um novo serviço em uma Configuração de um OBDH há que se definir seu conjunto de entradas e de saídas e, em seguida, mapear tais conjuntos aos mecanismos que as implementarão.

Uma vez definida a Configuração do OBDH, tem-se o novo modelo OBDH e, então, deve-se definir os mecanismos necessários para incluir o novo Modelo ao Simulador. Estes mecanismos **para inter-relacionamento entre componentes** do SMP são aqueles definimos na seção 2.2.2. Para definição desses mecanismos a seguinte sequência de questões deve ser respondida:

a) O modelo necessita de inicialização?

Se sim, ao menos um *Entry-point* será utilizado.

b) O modelo usará o serviço de escalonamento do simulador?

Se sim, outro *Entry-point* será utilizado.

c) O modelo possui saídas de dados?

Em caso positivo, o simulador deverá publicar estes dados. Para tanto, deve-se informar exatamente quais tipos de dados serão publicados. Há duas formas de disponibilizar os dados de saída: por publicação, quando o modelo permite o acesso ao dado; e por requisição, quando outro modelo solicitará a execução do método que retorna o dado.

d) O modelo deverá avisar outros modelos em caso de mudança de estado ou geração de evento assíncrono?

Se sim, o mecanismo de geração de eventos, permitindo que outros eventos se registrem para serem notificados, deverá ser implementado pelo modelo.

e) O modelo deverá ser informado de algum evento ocorrido em outro(s) modelo(s)?

Se sim, deverá ser implementado o mecanismo do SMP que permita ao mesmo se registrar junto ao modelo cujo evento deva ser notificado em caso de ocorrência.

f) O modelo deverá prover uma interface de requisição a métodos internos, ou seja, disponibilizar a interface IRequest permitindo que outros modelos possam executar seus métodos através de uma invocação dinâmica?

Se sim, deverá ser implementado o mecanismo do SMP que permite uma requisição, por outro modelo, de um método interno, onde o modelo requisitante fornece os parâmetros de entrada da método a ser invocado e recebe o valor de retorno.

A Figura 4.3 mostra um diagrama de atividades representando os passos descritos para definição dos mecanismos **para inter-relacionamento entre componentes** do SMP..

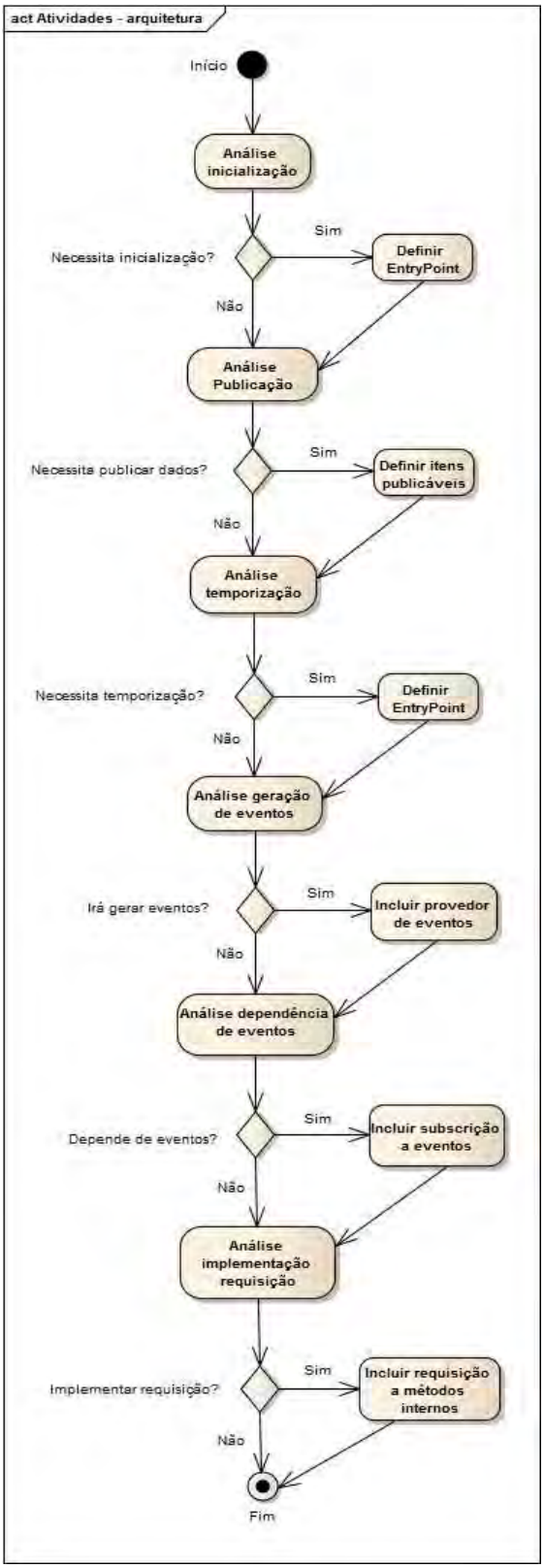


Figura 4.3 - Diagrama de atividades para utilização dos mecanismos de comunicação.

Após esta análise, é possível saber quais mecanismos o modelo deverá conter para que seja compatível com o padrão SMP e possa ser inserido em um simulador compatível com o padrão.

A Figura 4.4 mostra, de forma genérica, a interação entre dois componentes da arquitetura proposta. No primeiro plano, o Componente 1 é integrado ao Componente 2. No segundo plano, o Componente 1 é divido em dois outros componentes: 1A e 1B, os quais são integrados ao Componente 2.

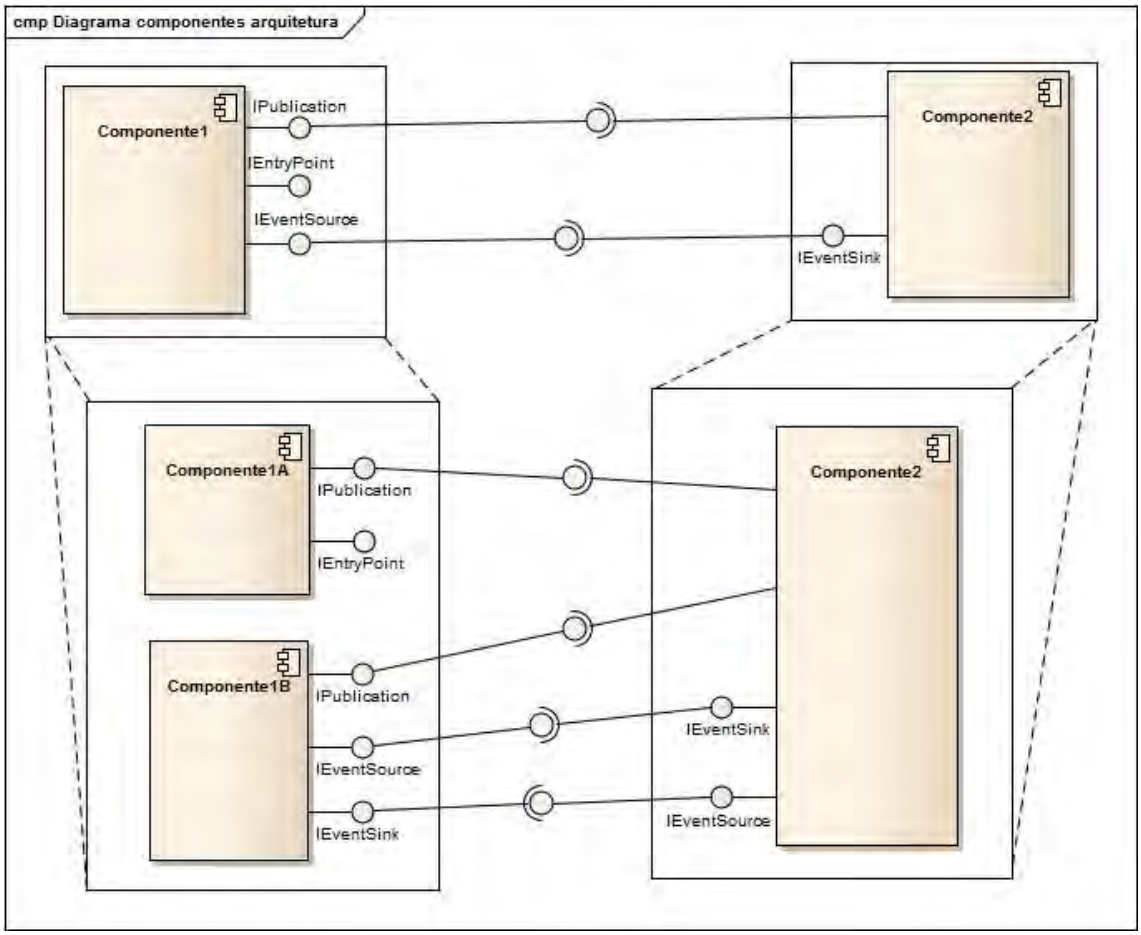


Figura 4.4 - Arquitetura genérica baseada em componentes.

Verifica-se que os mecanismos implementadas pelo Componente 1 continuam nos Componentes 1A e 1B, apenas que localizados no novo componente responsável pela sua funcionalidade. Esta divisão em um componente pode

implicar a necessidade de se incluir mecanismos adicionais para que a comunicação entre os modelos seja mantida.

4.3 Exemplo de aplicação da arquitetura

Neste item, a arquitetura genérica é mapeada em um exemplo ilustrativo. No exemplo, o modelo de OBDH é composto de dois serviços: Monitoração de Parâmetros e Estatística de Parâmetros. Os serviços são implementados inicialmente por dois componentes (cada um responsável por uma funcionalidade). Posteriormente, um dos componentes é novamente especializado em dois outros componentes, devido a uma especialização ou ao aumento do grau de fidelidade do serviço.

A Figura 4.5 ilustra este caso.

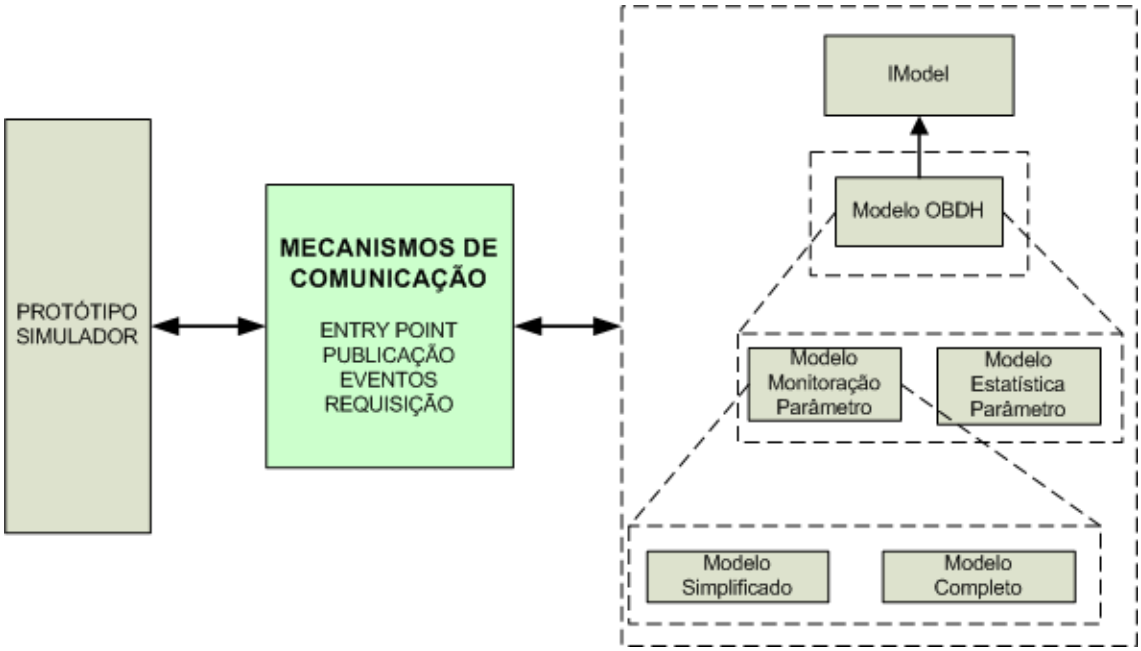


Figura 4.5 - Especialização de um modelo de OBDH.

A interface IModel, obrigatória para os modelos desenvolvidos dentro do padrão SMP, deve ser implementada em uma classe concreta genérica para modelos. Um subsistema de OBDH simplificado poderia ser implementado em

um único componente, representado pelo modelo OBDH. No primeiro nível de especialização, OBDH é decomposto em dois serviços, Monitoração de Parâmetros e Estatística de Parâmetros. Em seguida, o serviço de Monitoração de Parâmetro é especializado em dois modelos, o Modelo Simplificado, que apenas registra quais parâmetros de uma lista predefinida atingiram valores fora dos limites, e o Modelo Completo, que não só registra a informação do modelo simplificado como também permite registrar o valor atingido, identificar qual limite foi ultrapassado (superior ou inferior) e gerir a lista de parâmetros (inclusão e exclusão).

Comparando-se a Figura 4.5 com a Figura 4.4, observa-se que os serviços de Monitoração de Parâmetro e Estatística de Parâmetro correspondem ao Componente1 e Componente2, respectivamente. Havendo a decomposição do serviço Monitoração de Parâmetros, os novos componentes Modelo Simplificado e Modelo Completo correspondem, respectivamente, aos Componente1A e Componente 1B, que mantém os vínculos ao item Componente2 (ainda que haja modificações na quantidade de conexões).

A avaliação a ser feita, neste caso, é a forma de comunicação entre os componentes, ou seja, qual o fluxo de dados existentes e quais mecanismos de inter-relacionamento devem ser utilizados.

A aglutinação dos componentes traduz-se em um conjunto de serviços, compondo o OBDH a ser utilizado por um simulador. Os componentes devem possuir interfaces bem definidas entre si e o conjunto, o ODBH propriamente dito, também deve possuir interfaces bem definidas para serem utilizados pelo simulador. A concatenação e comunicação entre os serviços do OBDH e o ambiente de simulação são feitas através dos mecanismos previstos no SMP. Através da utilização dos mesmos, a substituição de um componente por dois ou mais componentes pode ser feita adaptando-se os mecanismos para que seja mantida a comunicação, dentro da padronização preconizada pelo SMP.

4.4 Verificação da arquitetura

Para verificar a flexibilidade da arquitetura foram criados diferentes modelos de OBDH e um protótipo de simulador baseado no SMP. Cada modelo de OBDH pode ser composto de um conjunto de serviços de OBDH implementados como componentes de software, de tal forma que, a união de vários componentes possa compor um modelo de OBDH específico. Os cenários visaram abranger diferentes aspectos da avaliação.

Entre os fatores considerados na avaliação da arquitetura estão:

(i) a complexidade para se modelar, implementar e gerenciar um ambiente de componentes para composição de diferentes simuladores;

(ii) o tempo utilizado na montagem de um OBDH, a partir dos componentes implementados e;

(iii) a viabilidade de se ter este processo dentro dos conceitos de portabilidade, reuso e interoperabilidade preconizados pelo SMP.

Cinco cenários foram definidos para que fosse possível colocar em prática a troca dos componentes de software para compor diferentes configurações de OBDH criados a partir da arquitetura. Estes cenários exploram o reuso pretendido bem como a flexibilidade de incremento e decremento da complexidade de um Modelo de OBDH dentro de um simulador.

Os seguintes cenários foram definidos para execução de diferentes configurações de OBDH:

Cenário 1 – agendamento de telecomandos a bordo

O modelo de OBDH deve receber telecomandos temporizados que devam ser agendados para execução posterior. O OBDH deve possuir uma configuração que inclua o serviço que implemente esta funcionalidade. Neste cenário o

simulador é usado para testar um caso de uso de recepção de telecomando cuja execução deva ser posterior ao recebimento. Uma vez agendado, o telecomando deverá ser executado na hora prevista. O modelo de OBDH neste cenário deve implementar apenas a funcionalidade de agendamento..

Cenário 2 – interação dos serviços de agenda e estatística

O modelo de OBDH deve incluir o serviço de agendamento do Cenário 1 e também o serviço de geração de relatório de estatísticas dos parâmetros. Neste cenário, o serviço de geração de relatório estatístico coletará os dados a serem computados posteriormente. Durante a execução do Cenário 2, o envio de telecomandos temporizados como definido no cenário 1 possibilitará a mudança nos valores estatísticos dos parâmetros. Assim, pode-se confirmar a integração entre os serviços nesta Configuração de OBDH.

Cenário 3 – modelo de OBDH com os serviços de monitoração e evento-ação

O objetivo do Simulador deste cenário é avaliar a autonomia do modelo de OBDH, com ações geradas de forma automática pelo serviço de Evento-Ação, em função da detecção de anomalia, pelo serviço de Monitoração de Parâmetros. Esta configuração do OBDH conterá dois serviços PUS interligados: Monitoração de Parâmetros e Evento-Ação. Sempre que o serviço de monitoração detectar que o valor de um parâmetro monitorado de um subsistema ultrapassou os limites definidos para o mesmo, um comando interno será gerado pelo serviço de evento-ação, visando corrigir o valor do parâmetro.

Cenário 4 – modelo de OBDH com os serviços de monitoração e agendamento

O simulador deve receber telecomandos temporizados, agendar a execução e ativar a monitoração dos parâmetros para posterior envio de relatórios de parâmetros com valores fora dos limites definidos. Equivale à configuração do cenário 1 com a inclusão do serviço que monitora a lista de parâmetros quanto

aos valores limites definidos. Da mesma forma que no cenário 2, as estatísticas podem ser modificadas pelos comandos temporizados. Neste cenário os valores podem ser modificados pelos comandos, fazendo com que os mesmos ultrapassem os limites.

Cenário 5 – substituição de componentes do modelo de OBDH existente

O simulador deve reutilizar a configuração do cenário 3, alterando alguns serviços por uma versão com maior fidelidade. O objetivo é mostrar a flexibilidade de se retirar componentes colocando outros com maior fidelidade, sem que a interface de comunicação seja modificada.

A execução destes cenários permite uma análise de como diversas combinações de componentes ou versões de um mesmo componente podem ser agrupadas e reutilizadas em diversas configurações de um modelo de OBDH de um simulador. A Tabela 4.1 mostra a correlação entre cenários e serviços.

Tabela 4.1 - Cenários para avaliação da arquitetura.

Serviços do OBDH	**Cenário 1**	**Cenário 2**	**Cenário 3**	**Cenário 4**	**Cenário 5**
Estatísticas de parâmetros					
Agendamento de comandos					
Monitoração de parâmetros					
Serviço evento - ação					

Os modelos de OBDH foram compostos para atender as necessidades de cada um dos cinco cenários distintos.

O passo seguinte à definição da arquitetura e dos cenários de verificação foi o desenvolvimento do protótipo de simulador para a execução dos cenários. A implementação real da arquitetura, através dos componentes de software representando os serviços PUS, previstos para comporem as configurações de OBDH desejadas, permitiu concretizar a proposta.

5 DESCRIÇÃO DO PROTÓTIPO

Este capítulo apresenta o protótipo de um simulador simplificado, desenvolvido para verificar a flexibilidade da arquitetura proposta para o modelo do subsistema de computação de bordo para um simulador de satélites.

5.1 Visão geral do protótipo simplificado

O protótipo consta de um simulador que segue o padrão SMP, contêm cinco modelos de subsistemas de satélite, quais sejam, OBDH, Térmica, Suprimento de energia (PSS) e Comunicação (TT&C), mais um modelo do segmento solo (SOLO).

A Figura 5.1 mostra a arquitetura do protótipo do simulador baseada no SMP. O modelo SOLO encontra-se junto aos demais modelos do simulador para efeitos didáticos, já que em um simulador real, este modelo não faz parte do conjunto de subsistemas internos ao satélite.

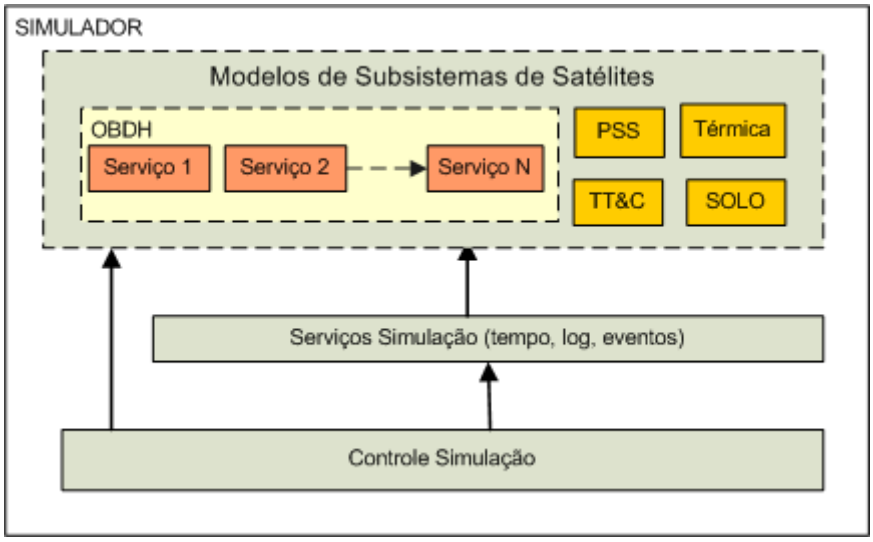


Figura 5.1 - Arquitetura do protótipo proposto.

O modelo OBDH é formado por um conjunto de serviços que implementam algumas funcionalidades definidas pelo PUS. Esta configuração varia dependendo da complexidade do OBDH que se queira simular.

Os estados do simulador são controlados por um núcleo de Controle da Simulação, que cria o ambiente de simulação e é composto do controle e mecanismos de comunicação.

O simulador exercita a máquina de estados ilustrada na Figura 5.2, de um simulador em conformidade com o padrão SMP.

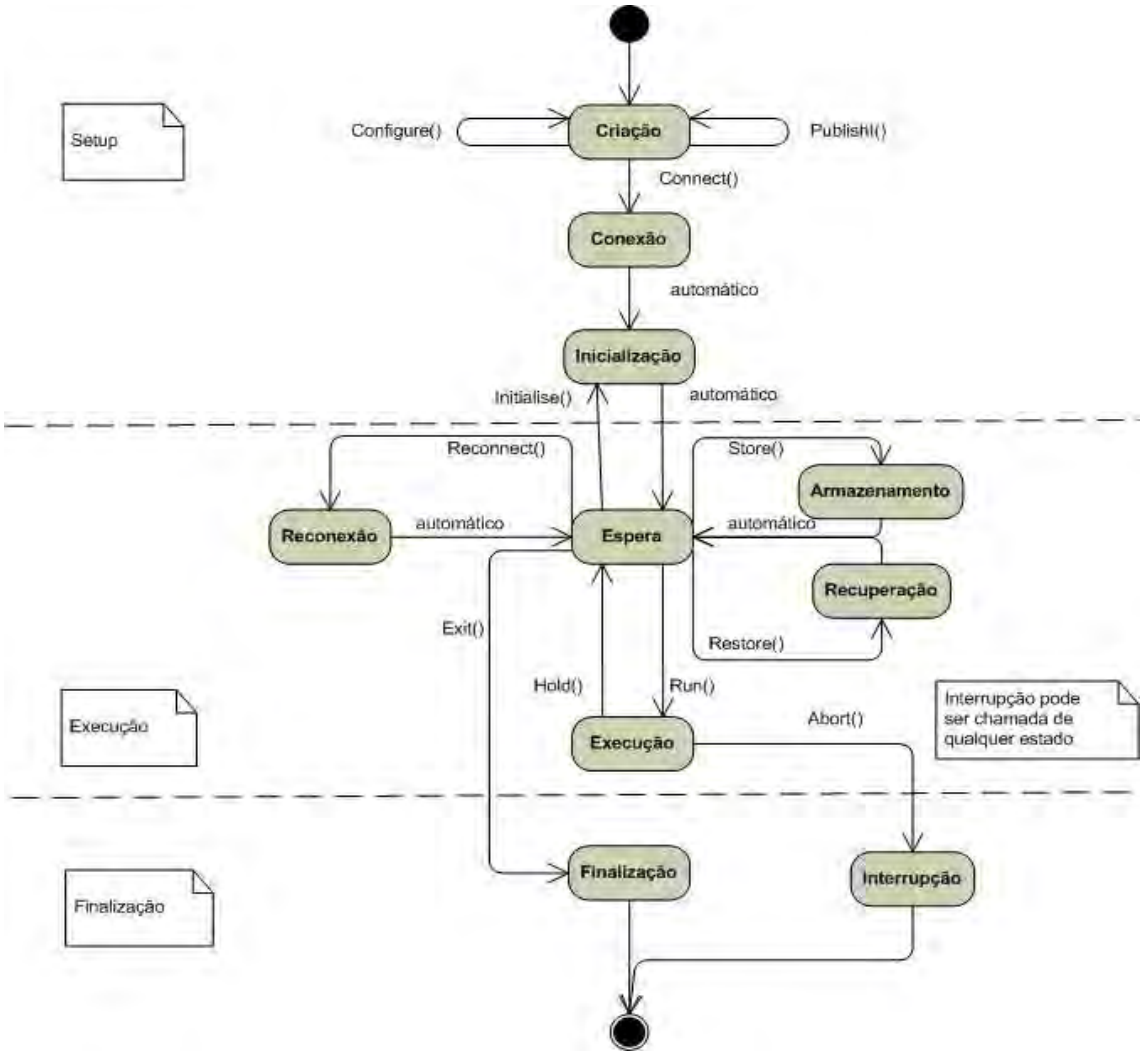


Figura 5.2 - Máquina de estados de um simulador SMP.
Fonte: Adaptada de ECSS (2011).

A máquina de estados define todos os estados pelos quais o simulador deve passar. Durante a etapa de *setup*, o simulador passa pelos estados de Criação, Conexão e Inicialização. No estado de Criação, todos os objetos do simulador são criados e os métodos *Configure* e *Publish* da interface ISimulator são executados. Estes métodos, por sua vez, chamam os mesmos métodos de cada modelo, para que os mesmos se autoconfigurem e publiquem seus métodos e/ou membros de dados. Em seguida, o simulador aciona seu método *Connect*, mudando o simulador para o estado de Conexão, fazendo com que todos os modelos também tenham seus métodos *Connect* acionados. Após a conexão, o simulador passa para o estado de Inicialização, quando eventualmente *Entry points* de inicialização são executados. Na sequência, o simulador entra no estado de Espera, aguardando o início da execução da simulação.

Na etapa de Execução, o simulador alterna entre os estados de Espera, Execução, Armazenamento, Recuperação e Reconexão. A mudança para o estado de Execução é feita através do método *Run*, ao passo que o retorno ao estado de Espera é feita pelo método *Hold*. No estado de Espera, com a execução paralisada, é possível armazenar-se a configuração do momento do simulador ou restaurar uma configuração anterior, previamente armazenada. Também do estado de Espera, é possível que sejam adicionados novos componentes ao simulador, fazendo-se necessária a conexão destes componentes ao simulador. Neste caso, o simulador irá ao estado de Reconexão através do método *Reconnect*.

Um mecanismo de escalonamento temporizado controla a execução dos modelos, enquanto o simulador encontra-se no estado de execução, produzindo as saídas esperadas.

Finalmente, a etapa de Finalização é atingida tanto pelo método *Exit*, que leva o simulador ao estado de finalização, quanto pelo método *Abort*, que leva o simulador ao estado de Interrupção, encerrando a simulação.

Os estados de armazenamento, recuperação, reconexão e interrupção não foram necessários dentro do escopo do trabalho.

O simulador foi baseado no Controle da Simulação definido na interface ISimulator do SMP. Esta interface é responsável pela implementação da máquina de estados do simulador mostrada na Figura 5.2 e por todo o controle da simulação. A Figura 5.3 mostra a classe concreta da referida interface.

Figura 5.3 - Classe principal do protótipo do simulador.

Os métodos *AddModel* e *AddService* são usados para inserir os respectivos objetos (criados a partir dos modelos e serviços) e preparar a estrutura hierárquica do simulador. Os métodos *Publish*, *Configure* e *Connect* são chamados para, internamente, invocarem os mesmos serviços de cada modelo da hierarquia, a fim de integrá-los ao ambiente da simulação. Os métodos *Run* e *Exit* iniciam e finalizam a simulação.

A título de ilustração, são mostrados, a seguir, trechos de código relacionados a alguns métodos listados.

Criação do simulador:

```
...  sim = new SIMSimulator("SIMPrototype","Prototipo do Simulador SMP"); ...
```

Adição de um modelo:

```
... sim->AddModel(objOBDH); ...
```

O método *Publish*, por sua vez, invoca, para cada modelo, o mesmo método, que publica membros de dados ou métodos de cada modelo da hierarquia:

```
...  sim->Publish(); ...
```

Conteúdo do método *Publish*.

```
... for (itModel = models->begin(); itModel != models->end(); itModel++)
        {
         Smp::IModel *model = dynamic_cast<Smp::IModel*>(*itModel);
         if (model->GetState() == Smp::MSK_Created)
                model->Publish(m_publication);
        }  ...
```

Neste código, para cada modelo da hierarquia, é verificado se o mesmo está no estado "Criado" (conforme a máquina de estados da Figura 5.2) e, em caso positivo, o método *Publish* do modelo é invocado, passando-se o objeto de

publicação (implementado a partir da interface IPublish). Cada um dos outros dois métodos (*Connect* e *Configure*) possuem comportamento semelhante ao *Publish*: executam um *loop* através da hierarquia e invocam o método de mesmo nome, para cada modelo.

O protótipo implementa três serviços de simulação previstos no SMP: Armazenamento (interface ILogger), Escalonamento (interface IScheduler) e Gerenciamento de Eventos (interface IEventManager).

A Figura 5.4 mostra a classe SIMLogger, que implementa o serviço de log.

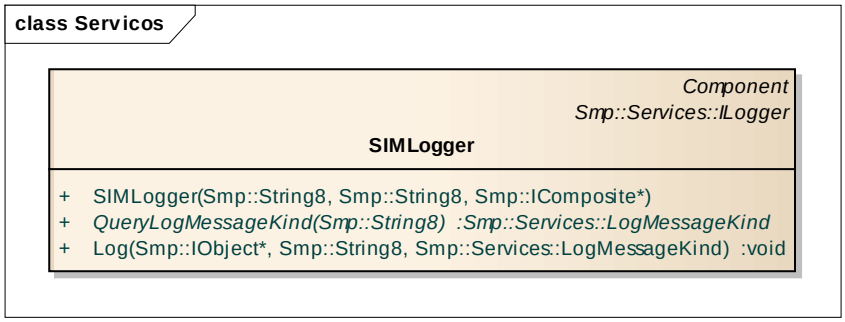


Figura 5.4 - Classe de *Log.*

No presente protótipo, o método apenas recebe a mensagem e a exibe. Já em uma aplicação real, o armazenamento poderia ser feito em um banco de dados, por exemplo.

A Figura 5.5 mostra a classe SIMScheduler, responsável pelo escalonamento das atividades temporizadas do simulador.

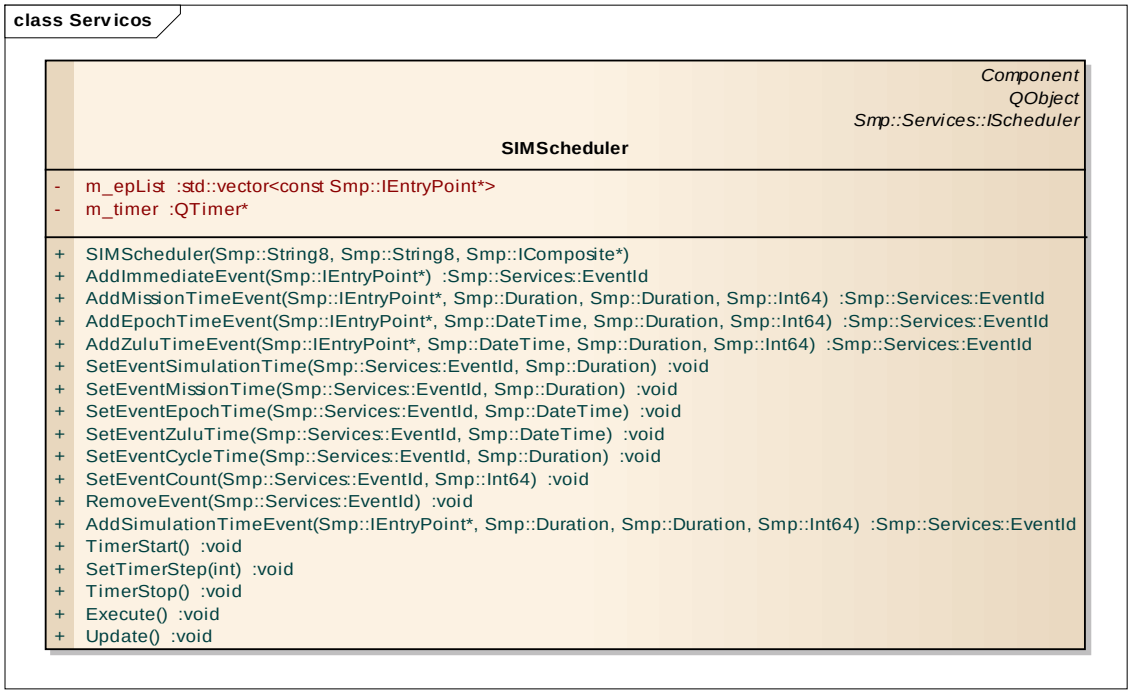


Figura 5.5 - Classe de escalonamento.

Apenas os métodos *AddImediateEvent* e *Execute* são implementados pelo protótipo.

Finalmente, a Figura 5.6 mostra a classe SIMEventManager, que gerencia os eventos globais da simulação.

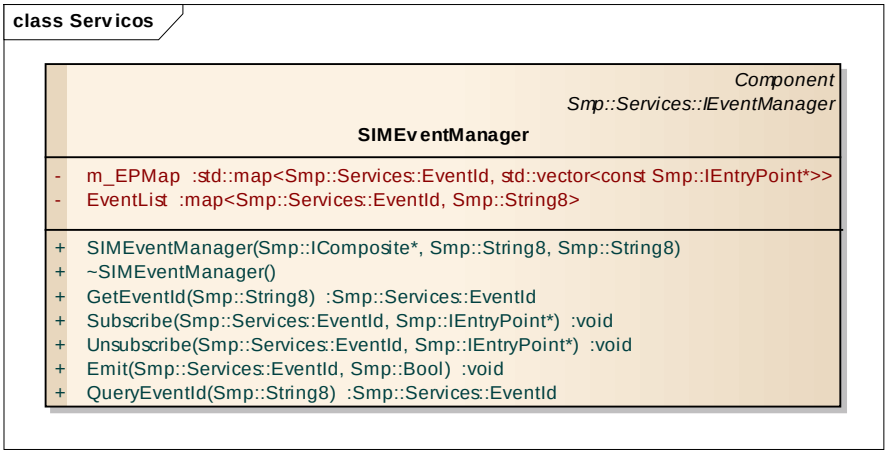


Figura 5.6 - Classe de gerenciamento de eventos do sistema.

Os métodos *Subscribe* e *Emit* são os principais responsáveis pela gestão dos eventos. Enquanto o primeiro permite controlar quem será notificado em caso de ocorrência do evento, o segundo emite o evento propriamente para a lista de objetos cadastrados. Um objeto poderá inscrever *entry points* vinculados a determinados eventos. Toda vez que o referido evento ocorrer, os *entry points* inscritos serão executados automaticamente.

A seguir, é mostrada a atuação do mecanismo. Para o método *Subscribe*, o código a seguir insere o *entry point* associado ao evento:

```
... m_EPMap[a_event] = EPvector; ...
```

Em caso de ocorrência do evento:

```
... for (std::vector<const Smp::IEntryPoint*>::iterator it = EPvector.begin();
        it != EPvector.end(); it++)
        {     if(a_sync)
              (*it)->Execute();
        }  ...
```

Os *entry points* que se registraram para o evento "a_sync" são executados.

Todos os *entry points* são executados na ordem em que se registraram.

5.2 Os modelos do simulador

Os modelos do simulador movimentam o fluxo de dados simulando a geração e recepção de dados entre os subsistemas que eles implementam. Estes dados são também trocados com o ambiente de simulação.

As funcionalidades dos modelos Solo, PSS, TT&C e Térmica são bastante simplificadas, objetivando basicamente implementar uma comunicação entre os mesmos e o OBDH.

O **modelo SOLO** (implementado na classe chamada GS) foi definido neste protótipo porque um OBDH real tem forte interação com o segmento solo, que executa as funções de envio de telecomando e recepção de telemetrias. Os telecomandos também permitem enviar solicitações de relatórios definidos pelo PUS e implementados no modelo do OBDH em uso pelo simulador.

A Figura 5.7 mostra a classe que implementa o modelo SOLO.

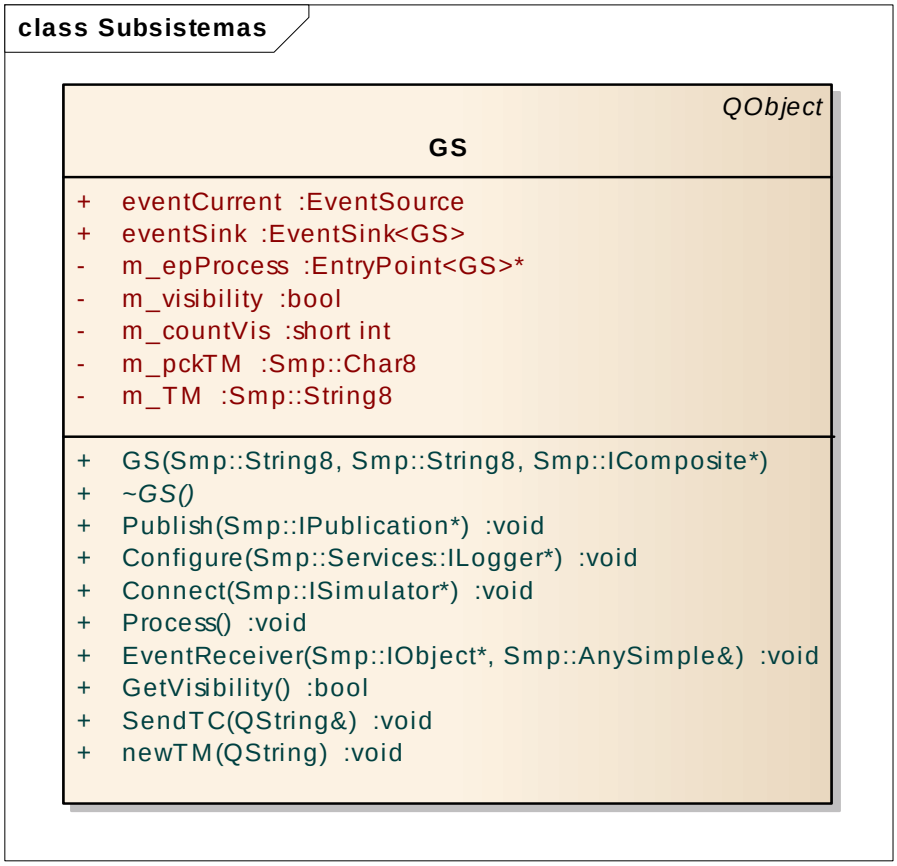


Figura 5.7 - Classe de implementação do modelo SOLO.

O método *Process* verifica se o satélite está em visibilidade pela estação. O objeto Solo está subscrito ao modelo de TT&C, que envia uma notificação sempre que o mesmo esteja em visibilidade pelo solo e haja telemetrias a serem recebidas. A inscrição é feita pelo código:

```
... objGS->eventCurrent.Subscribe(&objTTC->eventSink); ...
```

Esta inscrição é vinculada diretamente ao método *EventReceiver*, através do construtor da classe, como mostrado:

```
... eventSink("EventGS", "Event GS", this, &GS::EventReceiver) ...
```

O **modelo TT&C** executa duas funções: receber um telecomando e repassá-lo ao OBDH e receber a telemetria do OBDH e enviá-lo ao segmento solo. A Figura 5.8 mostra a referida classe.

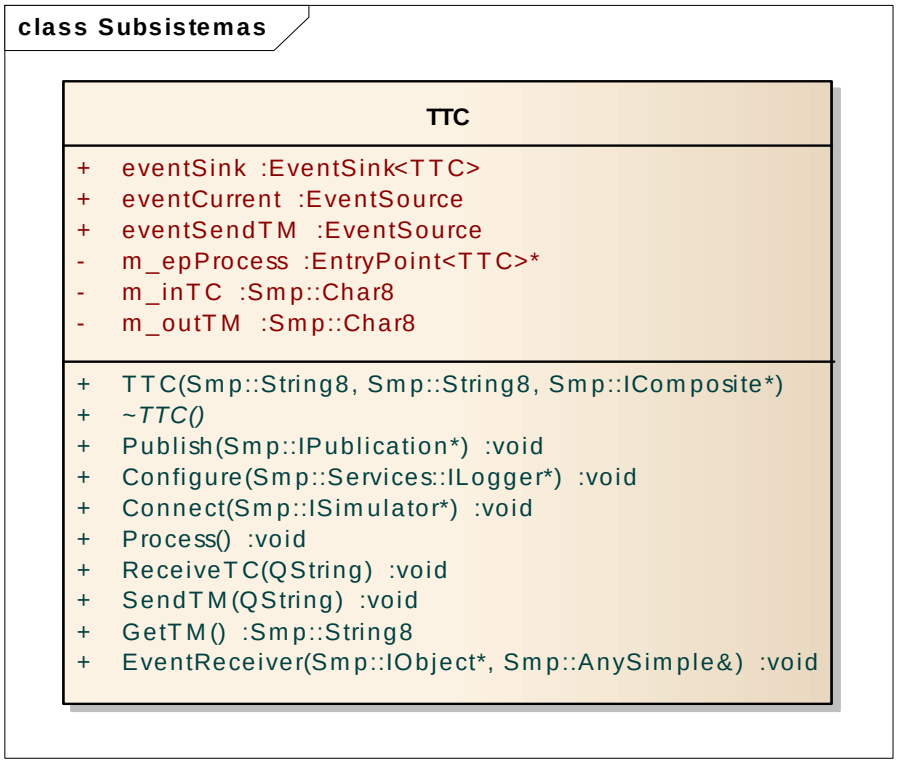


Figura 5.8 - Classe de implementação do subsistema TT&C.

Os procedimentos de recepção de encaminhamento de telemetria ou telecomando são feitos pelos mecanismos de inscrição e notificação, como mostrado na classe da estação. A diferença é que este subsistema comunica-se com outros dois subsistemas, recebendo dados do OBDH e repassando à estação e vice-versa.

O **modelo Térmica** apenas executa uma função senoidal simples, elevando ou diminuindo a temperatura do satélite ciclicamente, repassando os valores ao OBDH. A Figura 5.9 mostra a classe responsável pelo modelo Térmica.

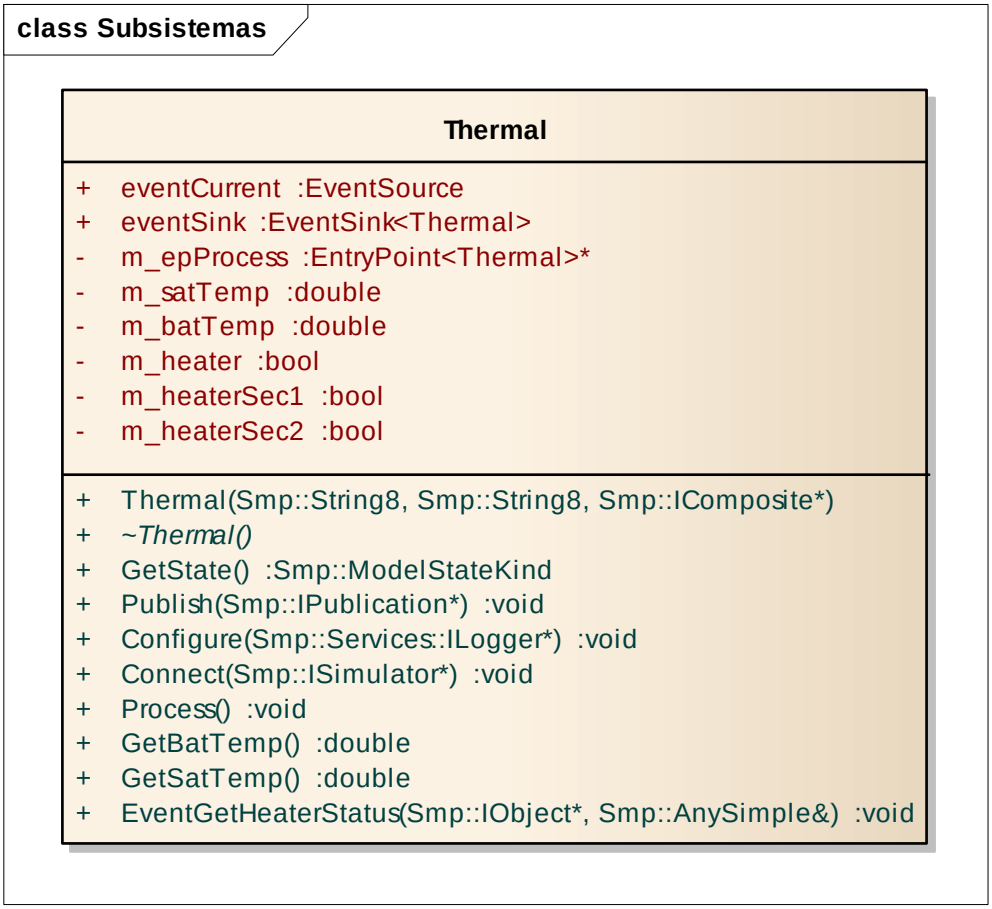


Figura 5.9 - Classe de implementação do modelo Térmica.

Foram criados três *heaters* que são ligados ou desligados via telecomandos, para que se possa influenciar na curva dos valores. A térmica opera dentro de

uma malha fechada de processamento, cujos valores são atualizados a cada ciclo de processamento do simulador. Portanto, a cada ciclo os novos valores são disponibilizados ao OBDH, que os recupera para formatação da telemetria.

O **modelo Suprimento de Energia (PSS)** simula a carga e descarga da bateria do satélite e repassa os dados de corrente e potência ao modelo OBDH. A Figura 5.10 mostra a classe responsável pelo PSS.

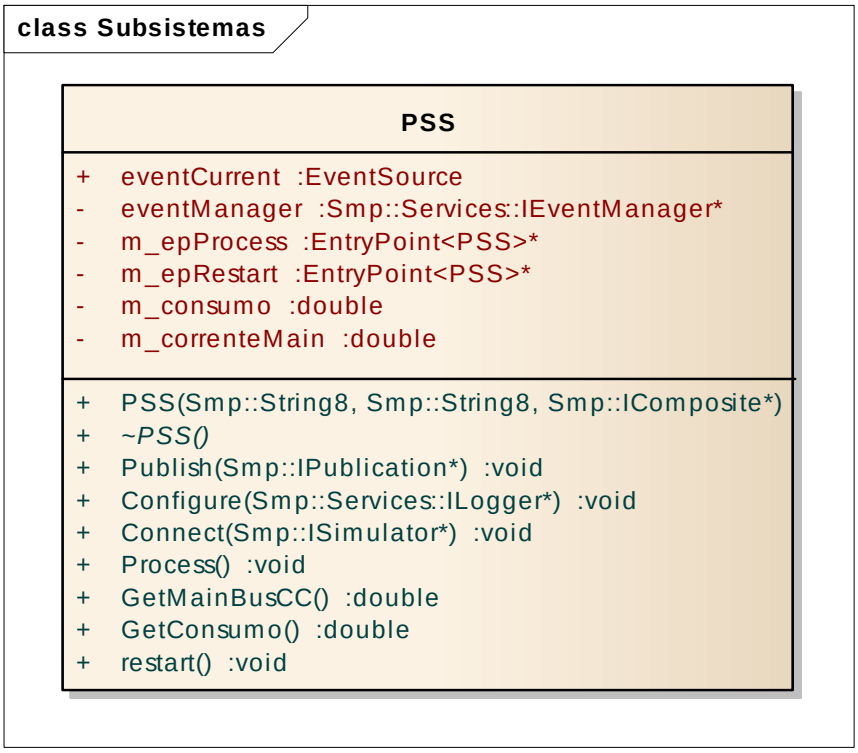


Figura 5.10 - Classe de implementação do modelo PSS

A geração de valores também obedece ao ciclo de processamento do simulador, com dados sendo disponibilizados ao OBDH em cada ciclo, tal qual no modelo de térmica.

Os modelos foram derivados da interface IModel, de acordo com descrição apresentada na seção 2.2 .

Além da modelagem e implementação dos subsistemas, são utilizados mecanismos para comunicação, seja entre modelos ou modelos e serviços ou, ainda, modelo e controle da simulação. As interfaces que definem os mecanismos são as de comunicação de eventos modelo-modelo (IEventSource, IEventSink), invocação dinâmica entre modelos(IEventDynamic, IReference e IRequest) e comunicação direta modelo-modelo (IPublication).

5.3 Fluxo de dados entre os modelos

A fim de estimular as interfaces de comunicação entre os modelos (da mesma forma que ocorrem as comunicações entre os subsistemas de satélite e do segmento solo), foi definido um conjunto de dados que perfazem uma malha dentro do simulador. Apenas o modelo de solo possui comunicação com a interface gráfica, uma vez que é através do segmento solo que as informações do satélite chegam ao Centro de Operações. Portanto, este modelo recebe dados da interface gráfica e os retransmite aos demais modelos via comandos. Os demais modelos operam em malha fechada, processando dados que são disponibilizados à interface gráfica através do modelo SOLO.

No âmbito da malha fechada, os modelos PSS e Térmica produzem dados dentro de uma taxa de tempo (ciclo de simulação). Estes dados são então disponibilizados ao OBDH, que por sua vez formata e envia ao modelo TT&C, de onde é lido pelo SOLO. Durante o processamento, o OBDH pode receber comandos que interferem nos subsistemas de PSS e Térmica, alterando seus valores. A Tabela 5.1 define o fluxo de dados entre os modelos dos subsistemas do satélite e entre estes e o modelo SOLO.

Tabela 5.1 - Fluxo de dados entre os modelos TT&C, OBDH, PSS, Térmica e SOLO.

	SOLO	**TT&C**	**OBDH**	**PSS**	**Térmica**
SOLO		Telecomando			
TT&C	Telemetria		Telecomando		
OBDH		Telemetria		Comandos direto e temporizado	Comandos direto e temporizado
PSS			Valores Corrente e Potência		
Térmica			Temperaturas bateria e satélite		

A Figura 5.11 sumariza os modelos e os dados trocados entre eles no simulador. Cada linha mostra a direção e que tipo de dado está sendo enviado/recebido.

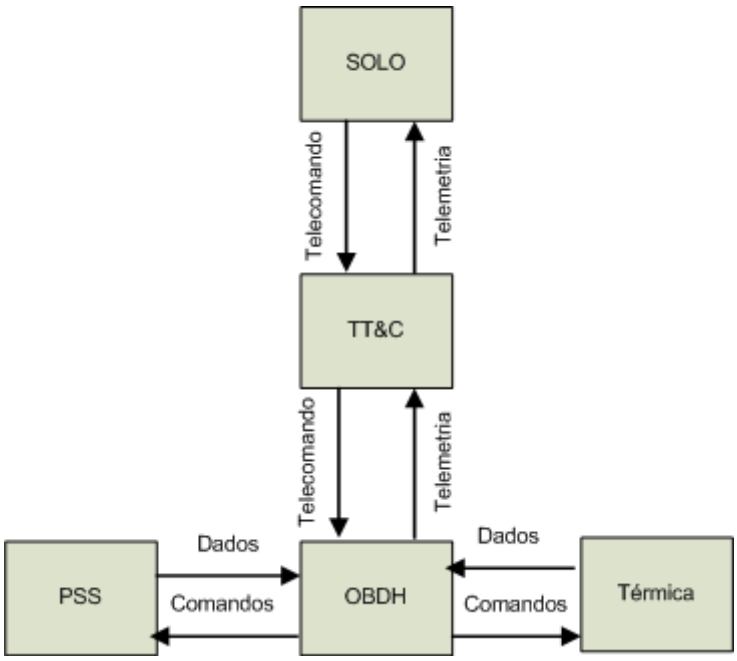


Figura 5.11 - Fluxo de dados entre os modelos dos subsistemas.

O fluxo Telecomando pode ser de dois tipos: direto ou temporizado. Um telecomando direto é executado imediatamente pelo OBDH, ao passo que um comando temporizado é agendado para execução posterior, de acordo com a temporização definida.

Os telecomandos foram estruturados tentando buscar uma proximidade com a estrutura de dados definida na seção 3.2 (ver Figura 3.4.). Da referida estrutura foram considerados os campos de APID, tipo e subtipo do serviço (conforme *header* do campo de dados) e os dados úteis do telecomando a serem processados pelo OBDH.

Para este protótipo foram criados quatro telecomandos: TC01, TC02, TC03 e TC04. Os três primeiros são diretos ao passo que o último é temporizado.

O telecomando TC01 possui a seguinte estrutura:

- Posição 0: Indica o APID da aplicação;
- Posição 1: Indica o tipo do serviço;
- Posição 2: Indica o subtipo do serviço;
- Posição 3: Indica o valor associado ao primeiro *heater* do subsistema de térmica.
- Posição 4: Indica o valor associado ao segundo *heater* do subsistema de térmica.

O telecomando TC02 possui a seguinte estrutura:

- Posição 0: Indica o APID da aplicação;
- Posição 1: Indica o tipo do serviço;
- Posição 2: Indica o subtipo do serviço;

- Posição 3: Possui um valor fixo que indica a requisição do relatório de estatísticas dos parâmetros de bordo;

O telecomando TC03 possui a seguinte estrutura:

- Posição 0: Indica o APID da aplicação;
- Posição 1: Indica o tipo do serviço;
- Posição 2: Indica o subtipo do serviço;
- Posição 3: Possui um valor fixo que indica a requisição do relatório de monitoração dos parâmetros de bordo;

O telecomando TC04 possui a seguinte estrutura:

- Posição 0: Indica o APID da aplicação;
- Posição 1: Indica o tipo do serviço;
- Posição 2: Indica o subtipo do serviço;
- Posição 3: Indica o valor associado ao terceiro *heater* do subsistema de térmica.
- Posição 4: Indica a temporização do comando

Para os telecomandos que possuem função de ligar ou desligar um equipamento, o valor 1 significa ligar enquanto que o valor 0 significa desligar. Já o telecomando para requisição do relatório possui o valor 1. Cada telecomando também inclui o APID da aplicação para a qual o mesmo está endereçado.

Exemplos:

TC01 = “11510” -> envia o telecomando direto ligando o primeiro *heater*

TC02 = “1171” -> solicita o relatório de estatística de parâmetros

TC03 = “1181” -> solicita o relatório de monitoração de parâmetros

TC04 = “11215” -> envia o telecomando temporizado, ligando o terceiro *heater* em 5 segundos.

5.4 *Model Development Kit* (MDK)

O Model Development Kit (MDK) é um conjunto de classes concretas implementadas a partir das interfaces abstratas do SMP. Desta forma, garantem-se os conceitos de interoperabilidade, portabilidade e reuso. Cada objeto de um simulador a ser desenvolvido herda diretamente das classes concretas implementadas pelo MDK, sem a necessidade de replicar toda a hierarquia de classes previstas pelo SMP. Por exemplo, para se criar um novo serviço de simulação, bastaria herdar da classe concreta do MDK que implementasse a interface IService, uma vez que toda a hierarquia já estaria implementada pelo MDK. Caso contrário, seria necessária a implementação de todos os métodos das interfaces superiores à IService, ou seja, os métodos das interfaces IComponent e IObject.

A Figura 5.12 mostra a hierarquia a ser construída sempre que um novo modelo SMP for implementado. Cada classe concreta deriva da classe concreta superior e da respectiva interface a ser implementada, exceto no caso da classe Objeto, que implementa apenas a respectiva interface.

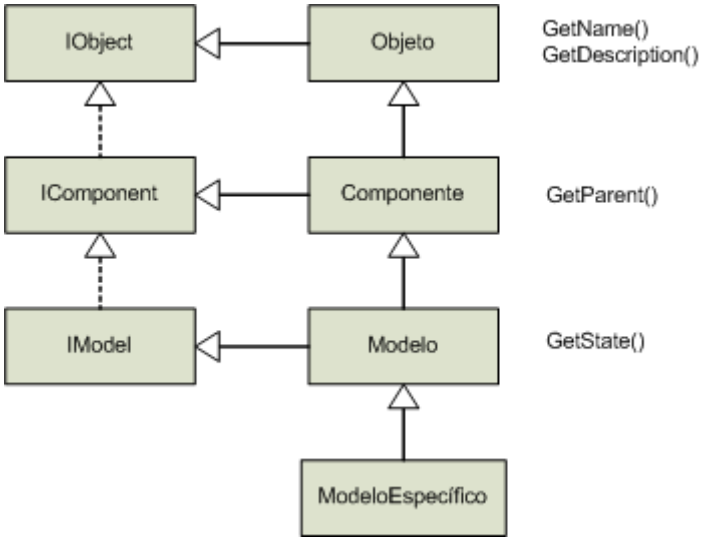


Figura 5.12 - Hierarquia para criação de um modelo específico no padrão SMP.

A criação de um novo modelo implica o mesmo derivar da classe Modelo, para que automaticamente os métodos GetName(), GetDescription(), GetParent() e GetState() sejam reusados, pois já estariam implementados pelo MDK, como mostra a Figura 5.12.

Para implementação do simulador, foi desenvolvido um MDK com as classes concretas das interfaces necessárias ao protótipo, bem como ao OBDH proposto. A Figura 5.13 mostra o diagrama de classes do MDK desenvolvido.

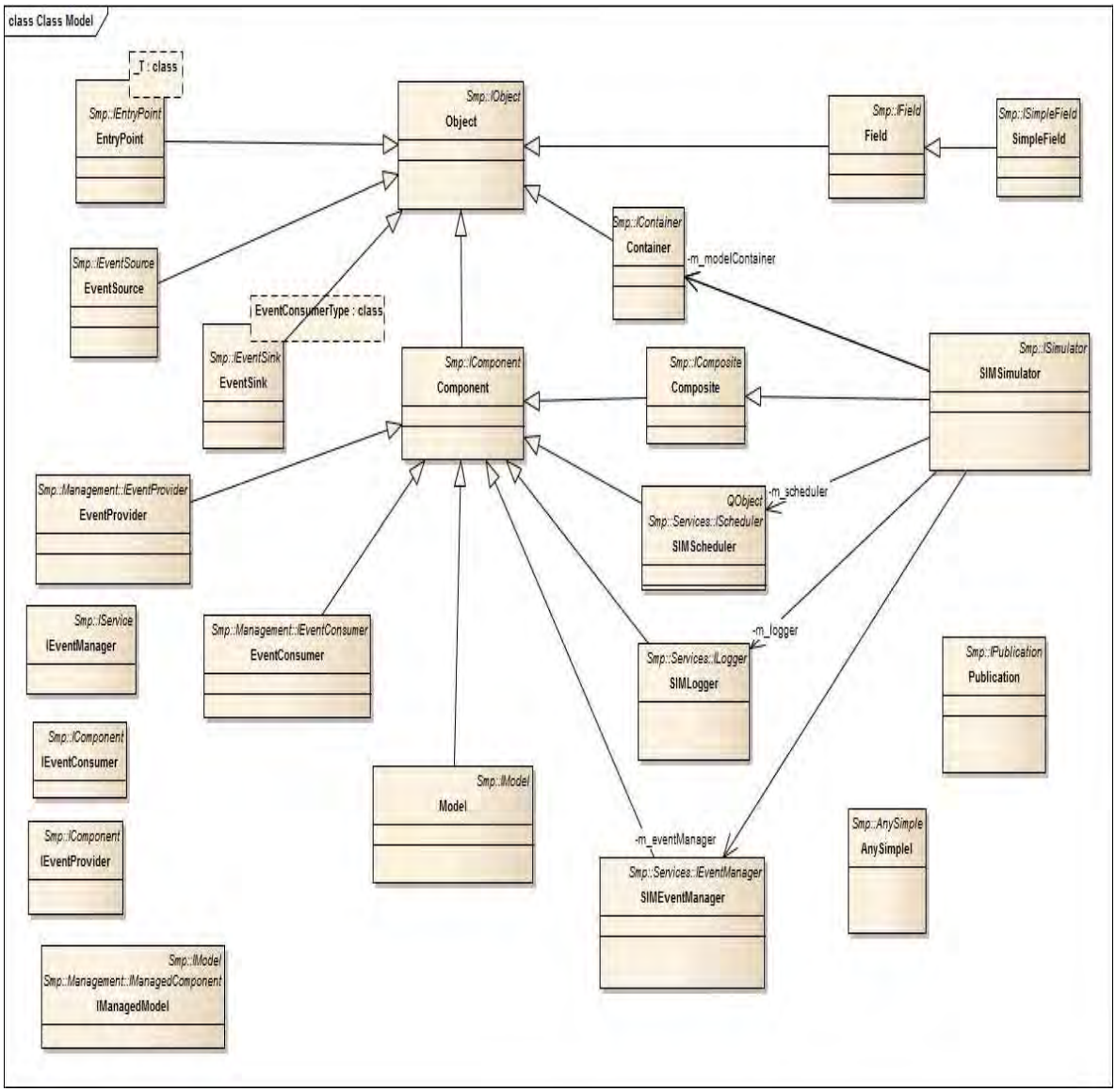


Figura 5.13 - Diagrama de classes do MDK com as classes necessárias ao protótipo.

Algumas soluções exigidas para implementação do SMP são específicas da plataforma ou da linguagem. Como exemplo, pode-se citar os tipos de dados e os containeres (vetores, listas, *maps*, etc). Para os tipos de dados, todavia, o SMP apresenta uma solução que permite a migração de um ambiente de desenvolvimento para outro, apenas alterando o mapeamento dos tipos de dados. Ainda assim, algumas restrições podem ser encontradas dependendo

de limitações da linguagem a ser utilizada. Já os containeres são específicos da linguagem, implicando solução única para cada mapeamento de linguagem.

A Figura 5.14 mostra o diagrama de classes dos subsistemas utilizando o MDK. Cada um dos modelos representando os subsistemas do protótipo do simulador foi desenvolvido derivando-se da classe Model do MDK, que por sua vez implementa a interface IModel e toda sua hierarquia.

Basicamente, as classes concretas dos subsistemas implementam os métodos virtuais da classe concreta do modelo SMP, como os de publicação, conexão e configuração, bem como os mecanismos necessários à comunicação com os outros modelos ou serviços da simulação.

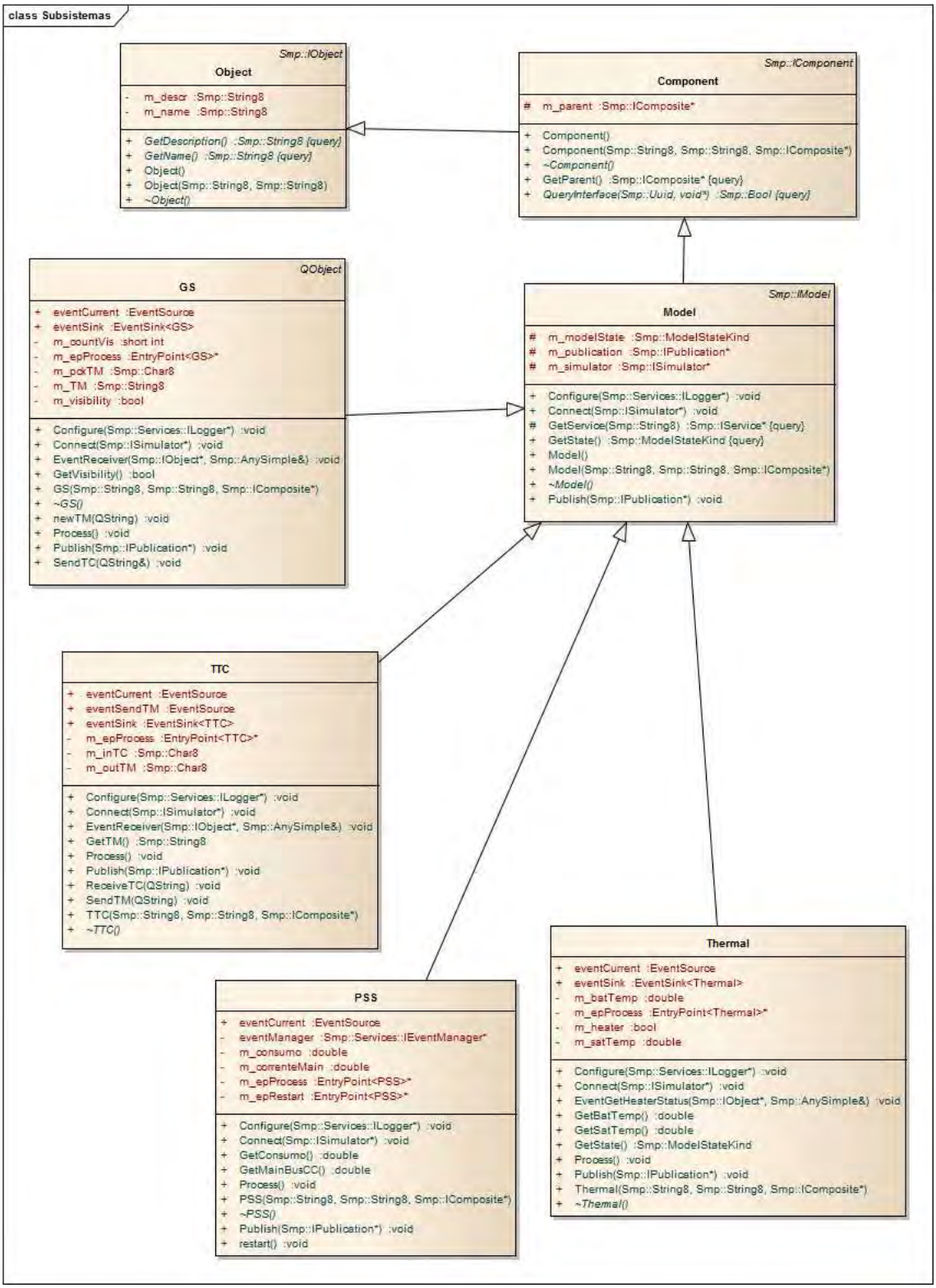


Figura 5.14 - Diagrama de classe dos subsistemas.

5.5 Mecanismos de comunicação implementados no protótipo

As comunicações ou trocas de dados entre os componentes, segundo o SMP, podem ser feitas de diferentes formas. Portanto, há que se analisar os requisitos da comunicação (tipo de dados a serem trocados, forma de notificação, dados retornados, etc.) para que se possa optar por este ou aquele mecanismo. As Figuras 2.6, 2.7, 2.8, 2.10 e 2.11 apresentadas na seção 2.2 detalham os mecanismos previstos no SMP.

A Figura 5.15 mostra alguns exemplos de mecanismos criados para comunicação entre os modelos de subsistemas do protótipo.

As classes concretas EventSink, EventSource, Publication e EntryPoint implementam operacionalmente as interfaces IEventSink, IEventSource, IPublication e IEntryPoint, respectivamente. Os subsistemas implementados nas classes TTC, PSS, Thermal e GS incluem, portanto, estes mecanismos, para que possam trocar dados entre si.

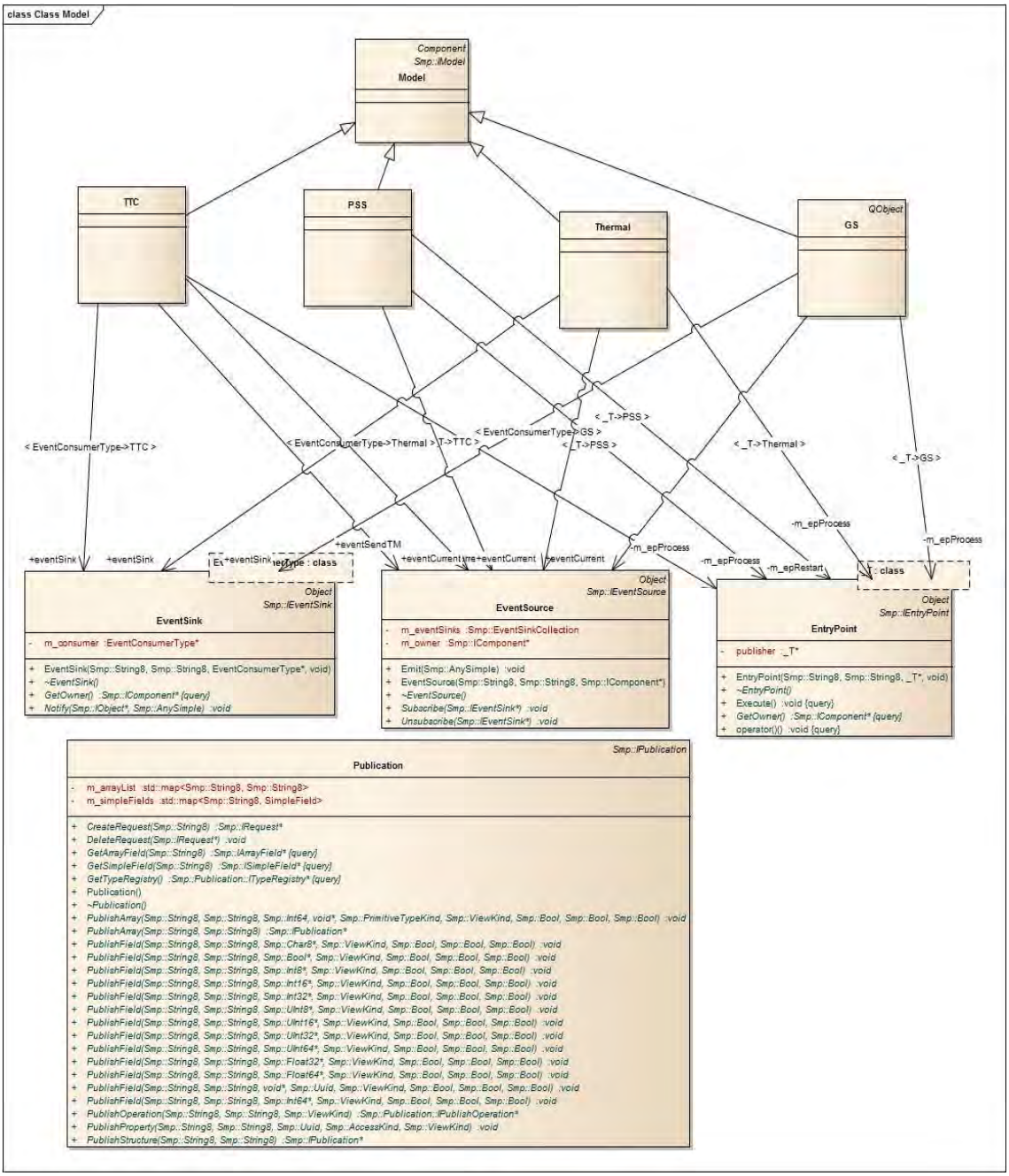


Figura 5.15 - Mecanismos de comunicação utilizados no protótipo.

A interface IEventSource (ver descrição em 2.2) permite ao componente SMP que a inclui, registrar outros componentes que devem ser notificados sempre que um evento for emitido pela classe que contenha a interface IEventSource.

A criação de um EventSource implica incluir um membro de dado deste tipo.

```
...  EventSource eventAction; ...
```

E, posteriormemente, a instanciação do mesmo:

```
... eventAction("EventAction", "Dispara EventoAcao", this) ...
```

O objeto que tiver que ser notificado de eventos ocorridos em outros objetos deve fazê-lo através de um membro interno do tipo EventSink.

Da mesma forma, há que se incluir um membro de dado.

```
...  EventSink<OBDHOnBoard> EventActionSink; ...
```

Seguido da sua instanciação.

```
... EventActionSink("GetEA", "Recebe EventAction", this, &OBDHOnBoard::ReceiveEventAction) ...
```

E, finalmente, o vínculo entre o EventSource e o EventSink, perfazendo a comunicação entre os objetos.

```
... objOBDHMon->eventAction.Subscribe(&objOBDHOB->EventActionSink); ...
```

Desta forma, sempre que um evento do tipo EventAction for disparado pelo objeto que o contém, o método ReceiveEventAction do objeto que contém o EventSink inscrito junto ao EventAction será acionado, executando seu código específico.

A notificação é feita como a seguir.

```
... (*dest)->Notify(sender, valor); ...
```

Desta forma, um “valor” é enviado pelo objeto “sender” ao objeto “dest”. O valor é qualquer faixa aceita pelos tipos de dado suportado na linguagem para a qual o SMP está sendo mapeada (int, string, bool, float, etc...).

Um *EntryPoint* deve ser usado para execução específica dentro de um objeto. Poder-se-ia utilizá-lo tal como um gatilho, disparado por um evento externo. Uma vez que o componente se registre para um determinando evento, toda vez que for notificado da ocorrência deste evento, o *EntryPoint* pode ser executado. Geralmente está associado à inicialização de um componente mas também pode ser utilizado para atividades pontuais. Como exemplo de aplicação pontual neste protótipo de simulador, o *EntryPoint* "ExecuteTimeTaggedCmd" está vinculado ao evento de recebimento do telecomando TC04, temporizado.

```
... m_execTimeTagged = new EntryPoint<OBDHOBSchedule>
("ExecTimeTagged", "Executa CMD temporizados", this,
&OBDHOBSchedule::ExecuteTimeTaggedCmd); ...
```

A Figura 5.16 mostra um diagrama de sequência UML contendo a comunicação entre os modelos, através de eventos e publicação dos dados pelos modelos.

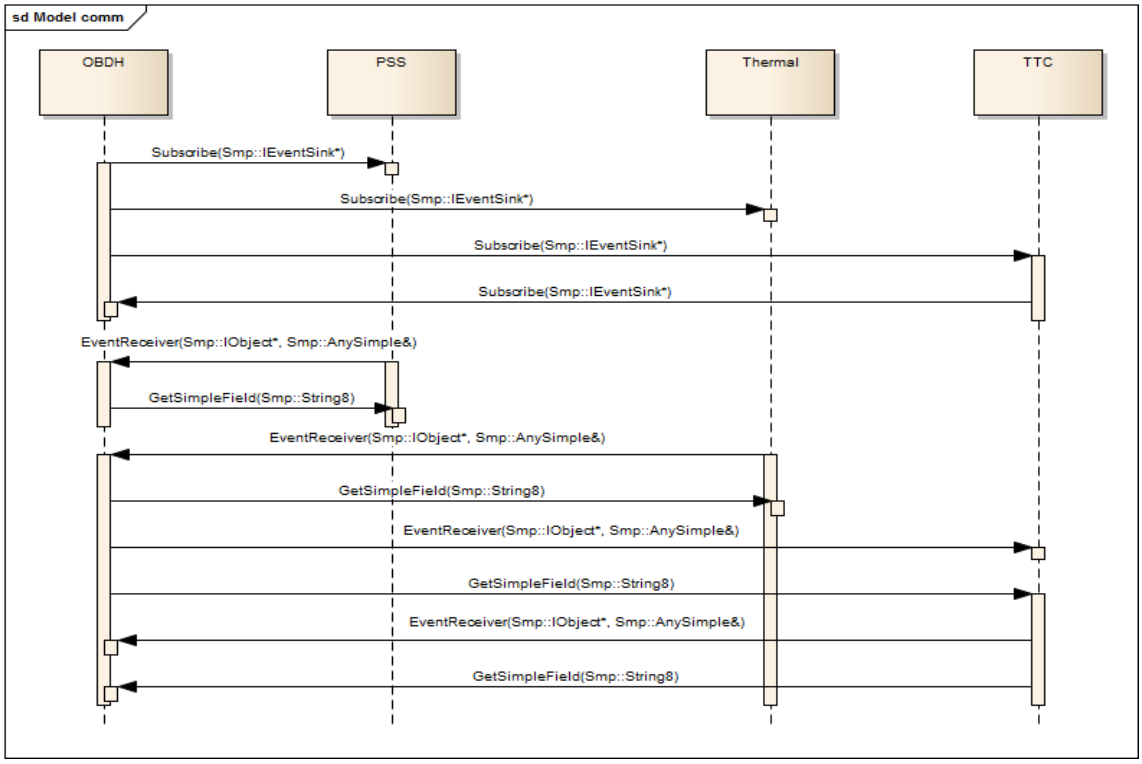


Figura 5.16 - Diagrama de sequência de comunicação.

A sequência de comunicação dá-se da seguinte forma:

a) Na criação do simulador, o OBDH se registra (método *Subscribe*) junto aos modelos de Térmica, PSS e TT&C, para ser notificado assim que estes modelos atualizarem seus valores internos;

b) Da mesma forma, o modelo TT&C também se registra junto ao OBDH para ser notificado assim que uma nova telemetria estiver disponível;

c) Os modelos de Térmica e PSS atualizam seus valores dentro do ciclo de escalonamento previsto e emitem um evento, que notifica o OBDH através do método *EventReceiver*;

d) O modelo OBDH busca estes novos valores (método *GetSimpleField*) e atualiza os dados de telemetria;

e) Após ter a telemetria atualizada, o OBDH emite um evento para o modelo TT&C (método *EventReceiver)*;

f) O modelo TT&C busca os novos dados de telemetria (método *GetSimpleField*) e os mantém disponíveis para o modelo da estação terrena (solo);

g) O mesmo processo ocorre no sentido inverso entre TT&C e OBDH. O primeiro, ao receber um telecomando da estação, notifica o OBDH, que por sua vez recupera o novo valor para processamento.

Quando um objeto gera um evento (como produzir um novo valor durante um passo de simulação), notifica os outros componentes inscritos, como mostrado a seguir.

```
... eventCurrent.Emit(AnySimpleI(true)); ...
```

O(s) objeto(s) notificado(s) pode(m), então, recuperar o valor publicado.

```
... Smp::ISimpleField* value = m_publication->GetSimpleField("BATTEMP");

    m_batTemp = value->GetValue().value.float64Value;   ...
```

Desta forma, todo o ciclo de comunicação fica estabelecido. Um modelo que produz dados recebe inscrições de outro(s) modelo(s) que desejam ser notificados assim que novos eventos forem emitidos. E cada modelo que produza dados a serem utilizados por outros modelos publica os mesmos para serem utilizados. A notificação pode ser opcional, caso a troca de dados possa ser assíncrona.

5.6 Os vários modelos de ODBH

Vários modelos de OBDH foram criados. Cada modelo possui um grupo diferente de serviços típicos de um computador de bordo. Cada serviço é um componente de software dentro da estrutura do OBDH, respeitando-se as interfaces definidas pelo padrão SMP.

Diferentes versões de um mesmo serviço foram desenvolvidas e encapsuladas em diferentes componentes.

Configurações diferentes de OBDHs foram criadas através da combinação de diferentes serviços ou de diferentes versões de um mesmo serviço.

Os serviços do PUS implementados de forma simplificada para testes da arquitetura do OBDH foram: Relatório de Estatísticas de Parâmetros, Agendamento de Operações, Monitoração de Parâmetros e Evento-Ação.

A Figura 5.17 mostra o diagrama da especialização do modelo de OBDH através de componentes de software.

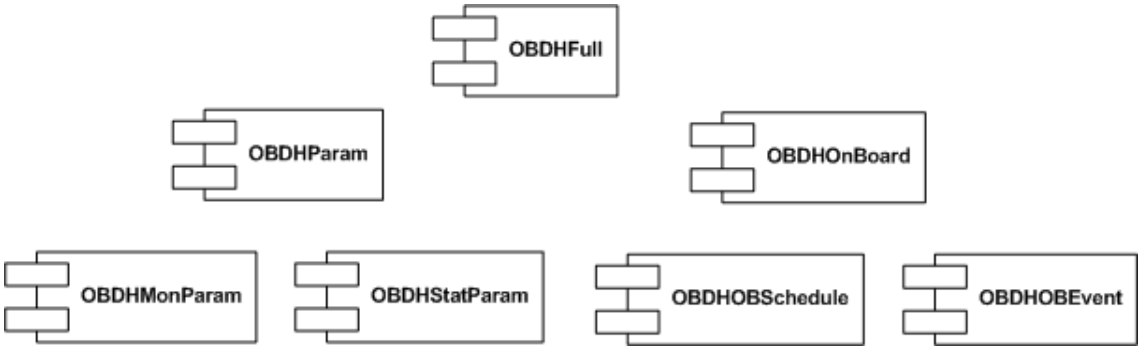


Figura 5.17 - OBDH especializado em diferentes componentes de software.

Os seguintes modelos de OBDH foram criados:

a) modelo OBDH completo (OBDHFull), contendo todos os serviços escolhidos,

b) OBDHParam - incluindo serviços associados a parâmetros (estatística e monitoração),

c) OBDHONBoard - incluindo agendamento e o serviço evento-ação,

d) OBDHMonParam - incluindo apenas a monitoração de parâmetros,

e) OBDHStatParam - incluindo apenas o serviço de estatísticas de parâmetros,

f) OBDHOBSchedule - incluindo apenas o agendamento,

g) OBDHEvent - incluindo o serviço evento-ação.

Como pode ser observado, foram implementados componentes de software com diferentes níveis de granularidade, desde um componente com todos os serviços até componentes com apenas um serviço.

Conforme detalhado no Capítulo 4 (Figura 4.2 e Tabela 4.1), para execução dos cenários foi feita a aglutinação de componentes de software de acordo com

os cenários previstos. Para cada um dos cenários foi feita a execução com o componente OBDHFull e depois com os componentes especializados através das funcionalidades.

A cada cenário em que um novo modelo de OBDH é utilizado no protótipo do simulador, novos componentes são incluídos ou retirados da configuração.

Cada vez que um componente é retirado ou inserido na configuração, os vínculos entre os componentes de software eram atualizados.

Por exemplo, o cenário três, definido no Capítulo 4.4 (serviços de monitoração de parâmetros e evento ação), exigiu três vínculos: do componente de monitoração com os subsistemas, do componente de evento-ação com os subsistemas e do componente de monitoração com o componente de evento-ação, já que este é acionado pelo componente de monitoração.

A seguir, os códigos mostrando estas conexões. Inicialmente a conexão entre o serviço de Monitoração e o serviço de Evento-Ação.

```
... objOBDHMon->eventAction.Subscribe(&objOBDHOB->EventActionSink); ...
```

Em seguida, a comunicação entre serviço Evento-Ação com o subsistema de Térmica.

```
... objOBDHOB->EventAction.Subscribe(&objTH->eventSink); ...
```

Finalmente, a vinculação entre o componente de monitoração e o subsistema de Térmica.

```
... objOBDHMon->eventTH.Subscribe(&objTH->eventSink); ...
```

Se o componente do serviço Evento-Ação fosse retirado do OBDH, o componente de monitoração continuaria emitindo informação de parâmetros fora de limite, mas não haveria um receptor para tratar esta informação.

Este Capítulo 6 mostrou todas as etapas para implementação do protótipo do simulador e também dos componentes de software responsáveis pelos serviços de OBDH baseados no padrão PUS.

Uma vez implementados, procedeu-se à execução dos cenários previstos no Capítulo 4.4, para verificação da arquitetura.

6 RESULTADOS

Este capítulo mostra os resultados obtidos com a execução do simulador quando configurado para atender cada um dos cenários previamente definidos.

O Simulador foi preparado para ser executado em duas etapas: uma com o OBDH como componente de software único, executando todos os serviços, outra com o OBDH combinado em diversos componentes, de acordo com os cenários, onde cada componente foi responsável por um ou mais serviços predefinidos.

Durante a simulação, os serviços do OBDH foram acionados, a fim de se verificar os mecanismos de comunicação e todas as interfaces do padrão SMP envolvidas.

6.1 Configuração inicial do protótipo do simulador

A configuração inicial define os valores iniciais internos aos subsistemas e ao ambiente de simulação. A mesma configuração inicial foi aplicada em cada execução do simulador. A Tabela 6.1 mostra os parâmetros de monitoração e seus respectivos limites inferior e superior. Qualquer valor fora dos limites sinaliza uma situação anormal no comportamento do subsistema sendo simulado.

Tabela 6.1 – Valores máximo e mínimo dos parâmetros.

Parâmetro	Unidade	Máximo	Mínimo
CONSUMO	W (Watt)	50	5
IMAIN	A (Ampere)	8	1
SATTEMP	ºC (Celsius)	35	5
BATTEMP	ºC (Celsius)	20	4

Estes parâmetros representam valores internos calculados pelos subsistemas. Para o PSS tem-se a potência (consumo) e a corrente do barramento principal. Já no subsistema de térmica são monitoradas as temperaturas. A tabela 6.1 também apresenta as unidades para cada parâmetro.

6.2 Execução dos cenários

O mesmo conjunto de dados e as respectivas telas foram usados em todos os cenários. Obviamente, dependendo do cenário em execução, diferentes valores eram esperados.

Esta seção mostra em detalhes a execução do cenário 3, conforme definido na tabela 4.1. Os resultados dos demais cenários estão descritos no Apêndice B deste documento.

A Figura 6.1 mostra a configuração do OBH utilizada no cenário 3, tomando-se por base a Figura 4.2.

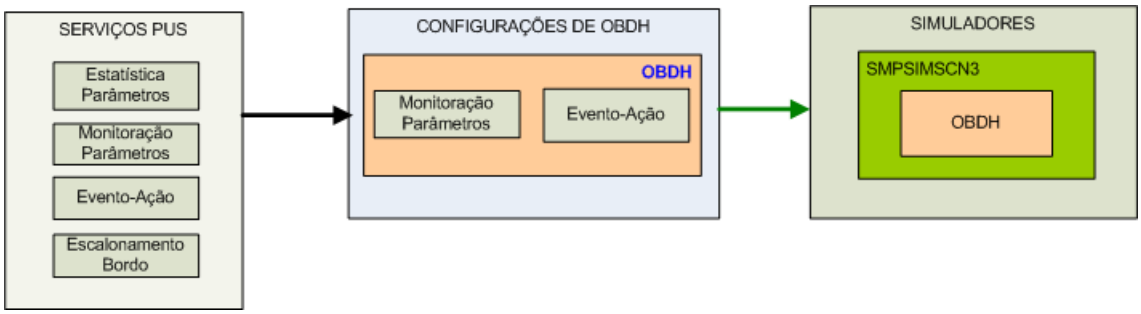


Figura 6.1 – Configuração do modelo do OBDH do cenário 3.

Neste cenário, os serviços de Monitoração e Evento-Ação foram simulados. Todos os mecanismos de comunicação foram implementados entre os componentes e o restante do simulador (ambiente e serviços). Ao se iniciar a simulação, os parâmetros listados no item anterior foram criados e, à medida que os passos da simulação ocorriam, os valores de monitoração eram contabilizados. Assim que um parâmetro ultrapassou algum limite (como previsto no Capítulo 6.1), o serviço Evento-Ação foi executado, visando a correção do valor do referido parâmetro.

A Figura 6.2 mostra a tela do protótipo do simulador.

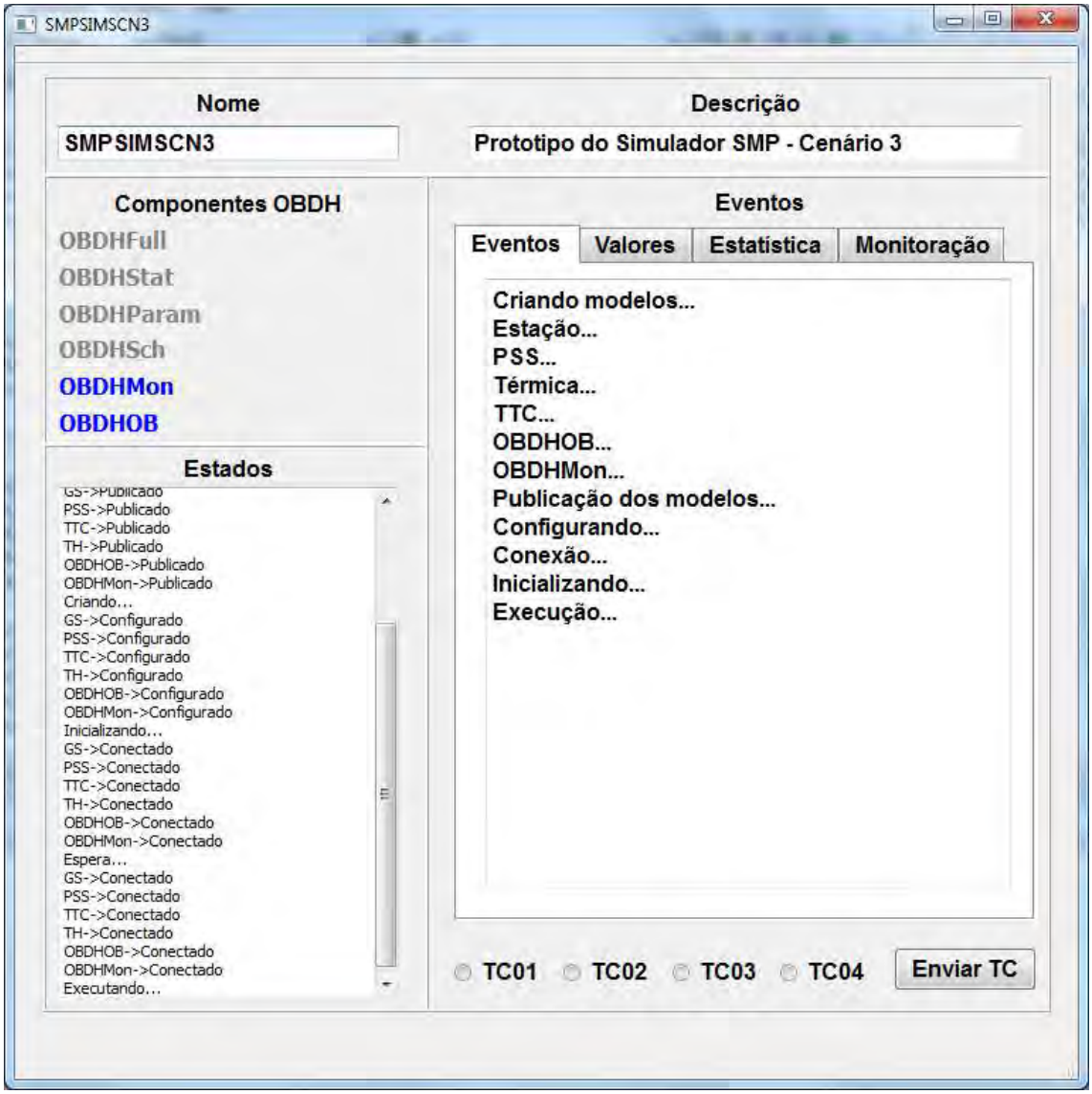


Figura 6.2 – Interface gráfica do protótipo do simulador.

Na parte superior da tela aparecem o nome e a descrição do simulador. No lado esquerdo superior, há a lista de todos os componentes de software criados para o simulador e em destaque estão os componentes selecionados para esta configuração de OBDH: OBDHMon, responsável pelo serviço de monitoração, e OBDHOB, responsável pelo serviço Evento-Ação. Ainda do lado esquerdo, abaixo, há a lista de estados pelos quais os subsistemas e o

simulador passaram, conforme mostrado na Figura 5.2. À direita da tela encontram-se as abas responsáveis por exibir os dados produzidos pelo simulador: eventos (mudança de estados, envio de telecomando), valores (em tempo real), dados estatísticos e de monitoração. Há ainda o recurso para o envio dos telecomandos.

A Figura 6.3 mostra a evolução dos valores dos parâmetros no decorrer da simulação.

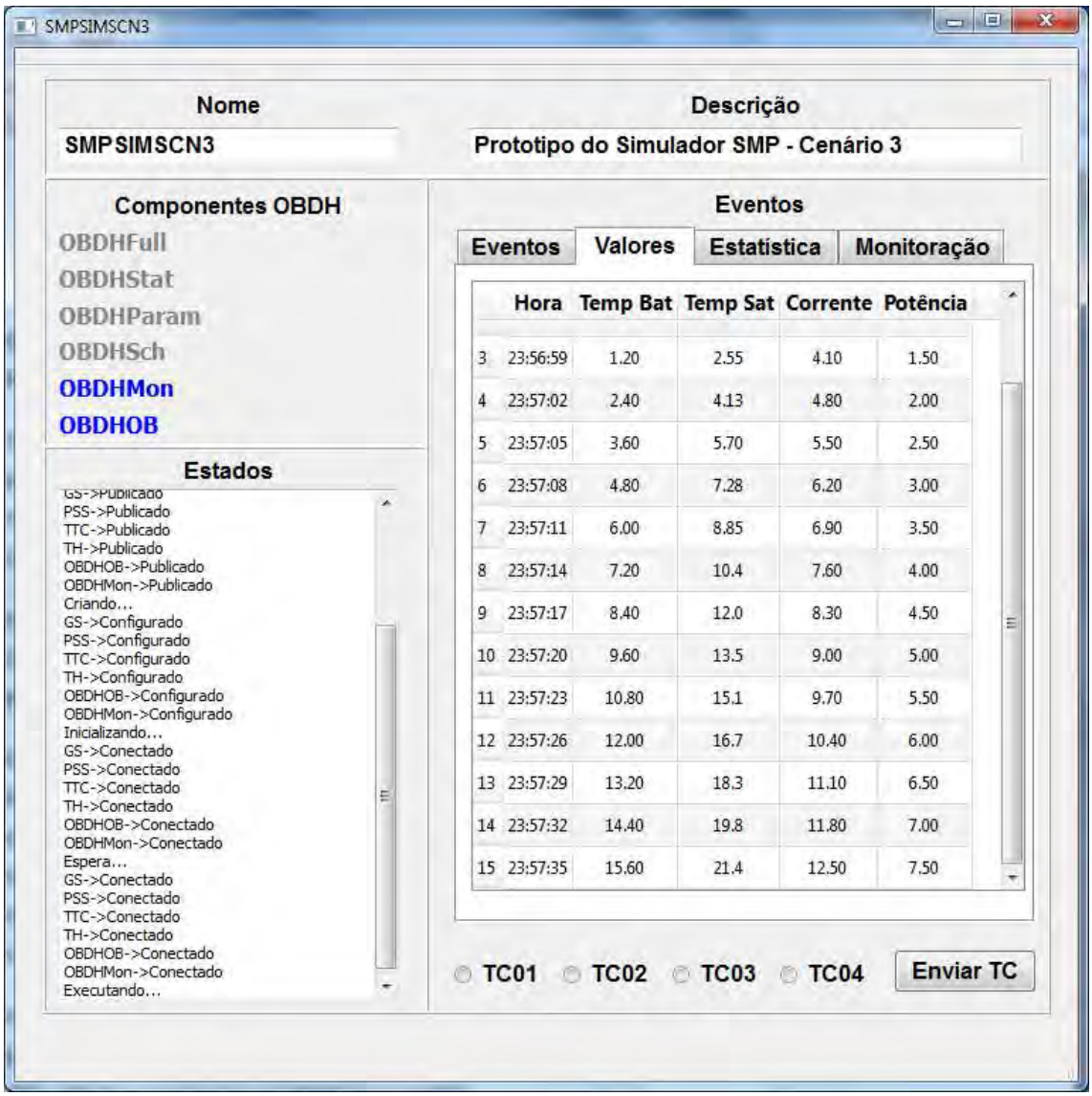


Figura 6.3 - Evolução dos valores dos parâmetros.

O serviço Evento-Ação monitora a temperatura da bateria e aciona o *heater* quando esta ultrapassa o limite inferior ou desliga o *heater* quando esta ultrapassa o limite superior. A Figura 6.4 mostra o momento em que o *heater* é acionado (temperatura abaixo do limite inferior, conforme definido na tabela 6.1). Nota-se o incremento da mesma, após o acionamento.

Eventos | Valores | Estatística | Monitoração

	Hora	Temp Bat	Temp Sat	Corrente	Potência
28	15:07:48	9.60	22.3	21.60	8.40
29	15:07:51	8.40	21.7	22.30	8.20
30	15:07:54	7.20	21.1	23.00	8.00
31	15:07:57	6.00	20.5	23.70	7.80
32	15:08:00	4.80	19.9	24.40	8.30
33	15:08:03	3.60	19.3	25.10	8.10
34	15:08:06	4.80	20.9	25.80	7.90
35	15:08:09	6.00	22.5	26.50	8.40
36	15:08:12	7.20	24.0	27.20	8.20
37	15:08:15	8.40	25.6	27.90	8.00
38	15:08:18	9.60	27.2	28.60	7.80
39	15:08:21	10.80	28.8	29.30	8.30

Figura 6.4 - Execução do serviço Evento-Ação no limite inferior

Da mesma forma, a Figura 6.5 mostra o momento em que o *heater* é desligado (temperatura acima do limite superior, conforme definido na tabela 6.1). Também é possível observar o decremento da temperatura quando o *heater* é desligado.

Eventos | Valores | Estatística | Monitoração

	Hora	Temp Bat	Temp Sat	Corrente	Potência
13	15:16:57	13.20	18.3	11.10	6.50
14	15:17:00	14.40	19.8	11.80	7.00
15	15:17:03	15.60	21.4	12.50	7.50
16	15:17:06	16.80	23.0	13.20	8.00
17	15:17:09	18.00	24.6	13.90	7.80
18	15:17:12	19.20	26.1	14.60	8.30
19	15:17:15	20.40	27.7	15.30	8.10
20	15:17:18	19.20	27.1	16.00	7.90
21	15:17:21	18.00	26.5	16.70	8.40
22	15:17:24	16.80	25.9	17.40	8.20
23	15:17:27	15.60	25.3	18.10	8.00
24	15:17:30	14.40	24.7	18.80	7.80
25	15:17:33	13.20	24.1	19.50	8.30

Figura 6.5 - Execução do serviço Evento-Ação no limite superior

O serviço de monitoração permite, além da verificação, pelo OBDH, dos parâmetros cujos limites foram excedidos, a obtenção de um relatório informando, para cada parâmetro, qual o limite ultrapassado (superior ou inferior) e qual valor o mesmo possuía quando excedeu o limite.

A Figura 6.6 mostra o envio do comando para obtenção do relatório de monitoração de parâmetros, conforme mostrado na seção 5.3.

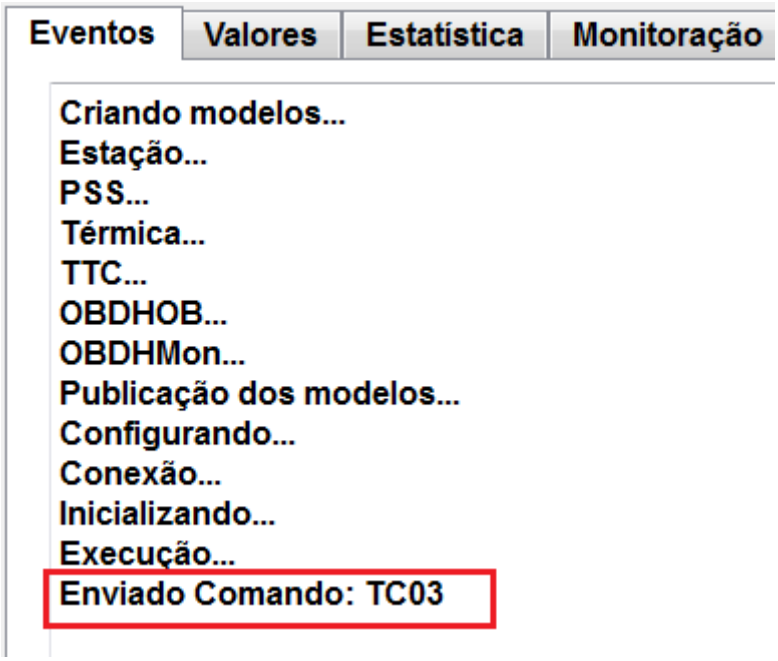
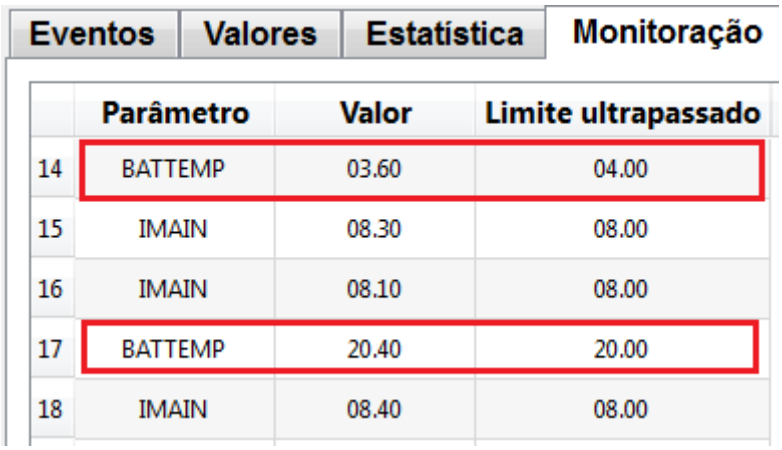


Figura 6.6 - Envio de comando para requisição do relatório de monitoração de parâmetros

A Figura 6.7 mostra o relatório contendo estes dados.



	Parâmetro	Valor	Limite ultrapassado
14	BATTEMP	03.60	04.00
15	IMAIN	08.30	08.00
16	IMAIN	08.10	08.00
17	BATTEMP	20.40	20.00
18	IMAIN	08.40	08.00

Figura 6.7 – Relatório de monitoração de parâmetros

As linhas destacadas mostram quando o parâmetro BATTEMP ultrapassou os limites inferior e superior, ou seja, qual limite ultrapassado (valor limite superior ou valor limite inferior) e qual valor o parâmetro possuía quando tal evento ocorreu.

A criação e execução do simulador a partir dos componentes de software contendo a implementação dos serviços foi uma atividade relativamente simples. O maior esforço foi direcionado na definição das entradas e saídas dos componentes de software a serem desenvolvidos para cada serviço do OBDH. Com os mesmos já disponíveis, para criação do simulador bastou a inclusão, no protótipo, do código responsável pelo vínculo entre os serviços, para o estabelecimento dos fluxos de dados. A criação das diferentes configurações de OBDH pela troca ou incremento/decremento de componentes foi igualmente simples, uma vez que estes já estavam prontos e disponíveis. Com isso, a interoperabilidade e o reúso preconizados pelo SMP ficaram evidentes.

A portabilidade não foi especificamente testada, porém o estudo do padrão SMP indica que é mais uma tarefa de migrar os tipos de dados para a plataforma desejada, além, obviamente, da adaptação do ambiente de desenvolvimento e criação do simulador, para posterior execução.

7 CONCLUSÃO

O trabalho desta dissertação propôs uma arquitetura de software flexível e reutilizável para o modelo do subsistema de computação de bordo para um simulador de satélites. A solução baseou-se nos padrões SMP e PUS. A arquitetura se mostrou viável, mesmo com as dificuldades e restrições existentes no padrão.

Para efeitos de simulação, e considerando o OBDH um elemento de software, definiu-se um modelo de OBDH com as funcionalidades típicas de um OBDH real. Um modelo é uma visão abstrata, portanto, um modelo de OBDH representará uma visão em alto nível dos serviços, sem estar vinculado a um OBDH específico de um determinado satélite. A utilização de padrões responsáveis pela organização e definição dos serviços comumente utilizados auxilia na modelagem dos subsistemas de OBDH dos satélites. Se a modelagem é feita com base em serviços genéricos e reusáveis, então o modelo pode ser aplicável em diferentes simuladores de satélites. Assim sendo, a utilização do PUS como base para a modelagem de OBDH permitiu definir uma solução baseada em serviços padronizados.

A parte prática desta dissertação que constou do desenvolvimento do protótipo simplificado do simulador e do MDK relacionado às interfaces necessárias ao protótipo permitiu sedimentar o conhecimento sobre o padrão SMP.

A exploração dos mecanismos de comunicação entre modelos, serviços de simulação e a camada de controle mostrou que uma tarefa importante de projeto é a escolha de qual mecanismo de comunicação é mais apropriado: publicação, eventos, *entrypoint* ou requisição direta ao componente.

7.1 Vantagens da arquitetura proposta

O uso do padrão SMP no desenvolvimento de simuladores apresenta inegável ganho, uma vez que foi estruturado por especialistas em projeto e

desenvolvimento de simuladores dentro da ESA, focando a solução de problemas reais com os quais estes profissionais se deparavam cotidianamente.

O padrão SMP mostra-se bastante promissor para desenvolvimento de simuladores, pois, impõe fortemente uma separação entre projeto e implementação, sendo o projeto baseado em modelos independentes de plataforma ou linguagem e que deve ser compatível com todas as interfaces do SMP.

O padrão contém os elementos necessários a um ambiente de simulação de satélites, bem como mecanismos de interligação entre os mesmos. A arquitetura do SMP através de sua divisão em modelos, serviços básicos e ambiente de simulação (controle), apresenta uma organização que facilita o uso.

A solução proposta permite a definição de modelos de OBDH escalonáveis, implementáveis em componentes de software. A correta análise do fluxo de dados nos modelos (entradas e saídas) possibilita o seu mapeamento para as interfaces de comunicação do padrão SMP. Uma vez que os componentes de software estejam implementados, uma dada configuração para um OBDH, em termos de funcionalidades esperada, é feita incluindo-se os componentes de software desejados, estabelecendo-se as comunicações das entradas e saídas, através dos mecanismos de comunicação já previamente incluídos nos componentes, quando da análise do fluxo de dados. Mesmo que um dado componente de software evolua para uma versão com maior fidelidade, no que tange ao domínio do problema por ele tratado, desde que as entradas e saídas se mantenham intactas, uma eventual substituição ou *upgrade* é feita sem modificações nos vínculos aos demais elementos do simulador com os quais o mesmo se relaciona. Em termos de desvantagem, há o fato de que os vínculos entre os objetos criados a partir dos componentes de software ainda não é feito de forma dinâmica. Em sendo estática, todas as modificações nos vínculos

entre os componentes de software obrigam a modificação do projeto, para geração de uma nova versão do simulador. Ou seja, não é possível, por exemplo, adicionar novos modelos ao simulador em tempo de execução. Caso se deseje isso, a simulação deve ser interrompida, para que o projeto do simulador seja modificado.

7.2 Dificuldades encontradas no uso do padrão SMP

A ESA tem investido no padrão SMP, procurando torná-lo cada vez mais sólido, porém a adoção deste padrão tem sido mais restrita às empresas e centros de pesquisa no âmbito da ESA. Desta forma, para que o mesmo torne-se mais amplamente utilizado ainda devem ser feitos ajustes e correções, até que o mesmo se torne uma norma ECSS.

A documentação não oferece respostas ou soluções, dentro das diretrizes de Engenharia de Software, a algumas situações no contexto de simulação de satélites, como, por exemplo, um modelo buscando um serviço da interface de simulação (a solução dada implica manter uma referência ao objeto da simulação em cada modelo), ou coleções de objetos (fortemente dependente da linguagem). Também possui inconsistências, como a referência a interfaces não documentadas, caso, por exemplo, a interface ILinkingComponent.

O escalonador não possui uma forma direta e prevista para ser acionado externamente. O acionamento deve ser feito através da interface ISimulator que contém os métodos de controle da execução, como pausa e reinício da simulação. A documentação não é clara quanto à solução a ser adotada, o que abre possibilidades de se adotar soluções em desacordo com a padronização preconizada pelo SMP ou pela técnica de orientação a objetos (como herança múltipla ou classes *friend*, por exemplo).

Outro ponto inconclusivo ocorre quando se tem vários modelos com passos de execução diferenciados. Ao se adicionar os mesmos no respectivo *container*

não há uma forma prevista de se definir a ordem de execução e a temporização diferenciada.

Para se fazer uso intensivo da característica de reuso do SMP é necessária a implementação de todo o *framework* na forma de um MDK, portado para uma determinada linguagem. Torna-se improdutivo ter que se desenvolver todos os métodos de uma dada hierarquia sempre que se necessitar inserir um novo componente SMP ou mecanismo em um simulador.

Percebe-se que o SMP acarreta algumas dificuldades para determinadas atividades esperadas em um simulador. Não há, por exemplo, uma forma direta de um modelo ter seus eventos mostrados em uma interface gráfica. O limite de comunicação é a interface ISimulator, que por sua vez não possui métodos nativos para comunicação com elementos externos ao SMP.

7.3 Limitações

Este trabalho está limitado ao escopo de modelos e comunicações, dentro do SMP, uma vez que o mesmo é bastante extenso. Não foram estudados, por exemplo, a parte da linguagem de *scripts* SMDL, alguns serviços como o de tempo (e suas implicações em um simulador de satélites, como aceleração do tempo de simulação), interfaces gerenciadas vinculadas ao SMDL.

A criação de um simulador de forma *hard-coded* implica considerável trabalho, com a inclusão manual das classes modificadas, contendo novos mecanismos de comunicação ou novos subsistemas e/ou serviços, para posterior compilação e geração do aplicativo.

Com a criação de componentes (DLLs), a tarefa de geração da aplicação é facilitada com a *linkagem* dos mesmos, sendo necessária a vinculação dos objetos, conforme mostrado no item 5.6.

7.4 Trabalhos Futuros

A principal proposta para trabalhos futuros é no sentido de se investir em ferramentas para modelagem nos moldes de ferramentas comerciais, como o Enterprise Architect. Além da característica de modelagem, a ferramenta também deveria apresentar funcionalidades de validação do modelo, para que se verifique a concordância dos mesmos com as interfaces do SMP. Posteriormente, estas ferramentas gerariam classes cuja estrutura interna esteja em consonância com o SMP (a ESA investe nesta direção, mas, como dito anteriormente, é uma iniciativa restrita).

Também se propõe estudo na linguagem de *script* do SMP com ênfase em ferramentas para geração e validação das mesmas. Também se sugere a utilização desta arquitetura dentro de um simulador e modelos que sejam criados a partir de scripts, que permitam a criação dinâmica do simulador, com a avaliação de eventuais vantagens e desvantagens.

REFERÊNCIAS BIBLIOGRÁFICAS

BARRETO, J. P.; AMBROSIO, A. M. **Um estudo sobre arquiteturas de simuladores de satélites**. São José dos Campos: INPE, versão: 2012-03-09. 51 p. Disponível em: <http://urlib.net/8JMKD3MGP7W/3BG6628>. Acesso em: 26 abr. 2012.

AMBROSIO, A. M.; CARDOSO, P. E.; BIANCHI NETO, J. Brazilian satellite simulators: previous solutions trade-off and new perspectives for the CBERS program. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2006), 9., 2006, Rome, Italy. **Proceedings...** 2006. p. 7. CD-ROM. (INPE-14068-PRE/9237). Disponível em: http://urlib.net/sid.inpe.br/mtc-m16@80/2006/08.21.15.01>. Acesso em: 16 abr. 2012.

AMBROSIO, A. M.; BARRETO, J. P.; GUIMARÃES, D. C. Satellite simulator requirements specification based on standardized space services. In: ISPE INTERNATIONAL CONFERENCE ON CONCURRENT ENGINEERING, 14th. (CE 2007), 2007, São José dos Campos. **Proceedings**... São José dos Campos: Springer, 2007. p. 171-179. CD-ROM; On-line. ISBN 978-184628-975-0. (INPE-15259-PRE/10080).

ARGÜELLO, L.; MIRÓ, L.; GUJER, J. J.; NERGAARD, K. SMP: A Step Towards Model Reuse in Simulation. **ESA Bulletin**, v 103, 2000. Disponível em < http://www.esa.int/esapub/bulletin/bullet103/arguello103.pdf>. Acesso em 29 nov 2011

AYYAZ, M.N.; SUDDLE, M.R.; ZAHID, S. Systems design of an economical and general-purpose on-board computer for low-earth-orbit micro-satellites. In: SECOND INTERNATIONAL CONFERENCE ON ELECTRICAL ENGINEERING (ICEE 2008), 2008, Lahore, Pakistan. **Proceedings…** IEEE, 2008. p. 1-9. ISBN 978-1-4244-2292-0. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4553947&isnumber=4553886> Acesso em 26 abr 2012

BARRETO, J. P.; HOFFMANN, L. T.; AMBROSIO, A. M. Using SMP2 standard in operational and analytical simulators. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2010), Huntsville, Alabama, EUA. **Proceedings...** AIAA, 2010. Papel. Disponível em: <http://pdf.aiaa.org/preview/2010/CDReadyMSO10_2129/PV2010_2267.pdf>. Acesso em: 26 abr. 2012.

CECHTICKY, V., MONTALTO, G., PASETTI, A., SALERNO, N. The AOCS framework. In: INTERNATIONAL ESA CONFERENCE ON SPACECRAFT GUIDANCE, NAVIGATION AND CONTROL SYSTEMS, 2010, Frascati, Italy. **Proceedings…** , 2010. Disponível em <http://www.pnp-software.com/Publications/Gnc02-V3.0.pdf>. Acesso em 07 jan 2012

CÔME, H.; IRVINE, M. The XMM Simulator - The Technical Challenges. **ESA Bulletin**, v 96, 1998. Disponível em < http://www.esa.int/esapub/bulletin/bullet96/COME.pdf>. Acesso em 30 nov 2011

COUCH, M.; CINA, G.; GONZALES, O.; MONTRONI, G.; PECCHIOLI, M. SCOS-2000 Release 5, A milestone in the evolution of the MCS infrastructure at ESOC. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS, 9TH, 2006, Rome, Italy. **Proceedings...** 2006. CD. Disponível em <http://pdf.aiaa.org/getfile.cfm?urlX=5%3A7I%276D%26XZ%22O%23S0WUWT%5B%5EPK%3B%3A4JL%22%0A>. Acesso em 29 nov 2011

DELHAISE, F.; BRU, T. Innovative concepts to reduce costs of Mission Control and Simulator for Lisa Pathfinder. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS, 9., 2006, Rome, Italy. **Proceedings...** 2006. CD. Disponível em: <http://pdf.aiaa.org/getfile.cfm?urlX=5%3A7I%276D%26XZ%22O%22S%20WUWT%5B%5EPK%3B%3A7%2AX%2C%0A. Acesso em 29 nov 2011

EUROPEAN COOPERATION FOR SPACE STANDARDIZATION (ECSS). ECSS. **Packet utilization standard**. 2003..Disponível em: <http://www.ecss.nl/>. Acesso em 16 dez 2011

_____. **Space engineering - software.** 2009. Disponível em: <http://www.ecss.nl/>. Acesso em 16 dez 2011

_____.**Simulation modelling platform.** 2011.Disponível em: <http://www.ecss.nl/>. Acesso em 16 dez 2011

EICKHOFF, J. **Simulating spacecraft systems**. Berlin: Springer Aerospace Technology, 2009. ISBN: (978-3-642-01275-4)

EUROPEAN SPACE AGENCY (ESA). ESA. **Simulation model portability**. 2001. Disponível em: <http://www.esa.int/TEC/Modelling_and_simulation/TEC2DCNWTPE_0.html> Acesso em: 14 jul 2011.

_____. **SMP 2.0 handbook.** EGOS-SIM-GEN-TN-0099, Issue 1, Revision 2, 2005. Disponível em: <http://www.eurosim.nl/support/manual_4_0/pdf/SMP_2.0_Handbook-1.2.pdf>. Acesso em 23 jun 2010

_____. **What is Galileo**. 2012. Disponível em < http://www.esa.int/esaNA/galileo.html> Acesso em 23 fev 2012

_____. **XMM Newton**. 2012. Disponível em < http://sci.esa.int/science-e/www/area/index.cfm?fareaid=23> Acesso em 23 fev 2012

_____. **LISA Pathfinder**. 2012. Disponível em < http://sci.esa.int/science-e/www/area/index.cfm?fareaid=40> Acesso em 23 fev 2012

FONSECA, L. INPE como provedor de dados de Observação da Terra gratuitos. In: MundoGEO#Connect, 2011, São Paulo. **Apresentação...** Disponível em: <mundogeoconnect.com/2011/arquivos/palestras/leila_fonseca-inpe_como_provedor_de_dados_gratuitos_de_observacao_da_terra.pdf >. Acesso em 14 out 2011

FRANCISCO, M. F. M. **Sistemas computacionais em aplicações espaciais**. São José dos Campos: INPE, 2003. 17 p. (INPE-9604-PUD/125). Disponível em: <http://urlib.net/sid.inpe.br/jeferson/2003/10.13.15.25>. Acesso em: 26 abr. 2012.

FRITZEN, P; SEGNERI, D.; PIGNÈDE, M. SWARMSIM – The first fully SMP2 based Simulator for ESOC. In: INTERNATIONAL WORKSHOP ON SIMULATION & EGSE FACILITIES FOR SPACE PROGRAMMES (SESP 2010), 11., 2010, Noordwijk, The Netherlands. **Proceedings…** 2010. Disponível em: http://www.congrex.nl/sesp_proceedings/. Acesso em 26 abr 2012.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns:** elements of reusable objected-oriented software. 12 ed, 1997, ISBN (0-201-63361-2)

HOMEM, M. T; PIGNÉDE, M; MERRI, M.; REGGESTAD, V; PIDGEON, A.; MATUSSI, S. The GALILEO simulator: a major step in software technology from single spacecraft to constellation simulators. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2006), 9., 2006, Rome, Italy. **Proceedings...** 2006. Disponível em <http://pdf.aiaa.org/getfile.cfm?urlX=5%3A7I%276D%26XZ%22C%2BR%20%5FUWT%5B%5EPK%3B%3A7%3AP%26%0A>. Acesso em 29 nov 2011

KÜHNE, T. What is a Model?. In: DAGSTUHL SEMINAR, 2005, Darmstadt, Germany. **Proceedings...** 2005. ISSN 1862-4405. Disponível em <http://drops.dagstuhl.de/opus/volltexte/2005/23/pdf/04101.KuehneThomas1.Paper.pdf>. Acesso em 09 jan 2012

LINDMAN, N.; DI NISIO, N.; SEBASTIÃO, N. The Implementation of the Simulation Model Portability 2 Specification at ESOC. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2006), 9., 2006, Rome, Italy. **Proceedings...** 2006. Disponível em <http://pdf.aiaa.org/getfile.cfm?urlX=5%3A7I%276D%26XZ%22C%2BP0OUWT%5B%5EPK%3B%3A7%3AL%2D%0A>. Acesso em 29 nov 2011

LUDEWIG, J. Models in software engineering – an introduction. **Software and Systems Modeling**, v. 2, n. 1, p. 5-14, March 2003. Disponível em < http://dx.doi.org/10.1007/s10270-003-0020-3 >. Acesso em 09 jan 2012

MATHWORKS. **Simulink**. The MathWorks, Inc. Disponível em: < http://www.mathworks.com/products/simulink/> Acesso em 27 jan 2012

MCNAMARA, P.; RACCA, G. Introduction to LISA Pathfinder. In: ESA Science, Reference LISA-LPF-RP-0002, 2009. Disponível em < http://lisa.gsfc.nasa.gov/Documentation/LISA-LPF-RP-0002_v1.1.pdf>. Acesso em 08 nov 2011

OBJECT MANAGEMENT GROUP, INC. (OMG). **Model driven architecture version 1.0.1.** 2003. Disponível em <http://www.omg.org/mda/>. Acesso em 17 Nov 2011.

NEMETH, S. M., DEMAREST, P. Research and Development in Application of the Simulation Model Portability Standard. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2010), 11., 2010, Huntsville, Alabama, EUA. **Proceedings...** 2010. Disponível em: <http://pdf.aiaa.org/getfile.cfm?urlX=6%3A7I%276D%26X%5BR%5B%22R0%5F%40P4S%5EQ%3AO%225JD%23%5DP%20%20%0A>. Acesso em 29 nov 2011.

P&P SOFTWARE GMBH. **OBS framework**. Desenvolvido por P&P SOFTWARE GMBH. 2003. Disponível em: < http://www.pnp-software.com/ObsFramework/doc/Home.html > Acesso em: 06.08. 2011.

PIDGEON, A.; STRAW, S.; BODEMANN, C.; IRVINE, M. Galileo constellation operations simulator. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2008), 10.,. 2008, Heidelberg, Germany. **Proceedings...** 2008. Disponível em: <http://pdf.aiaa.org/getfile.cfm?urlX=6%3A7I%276D%26X%5B%22S%2FR0CHP4S%5EQ%2A%2F%225Z%5C%24%5DP%20%20%0A. Acesso em 29 nov 2011.

PIGNÈDE, M; MORALES, J.; FRITZEN, P; LEWIS, J. Swarm Constellation Simulator. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2010), 11., 2010, Huntsville, Alabama, EUA. **Proceedings...** 2010. Disponível em <http://pdf.aiaa.org/getfile.cfm?urlX=6%3A7I%276D%26X%5BR%5F%28P0OOP4S%5EQ%3AO%225J%40%27%5F%20%20%20%0A>. Acesso em 29 nov 2011

PRESSMAN, R. S. **Software engineering, a practioner´s approach**. 4. Ed. Mcgraw-Hill,,2005.

REGGESTAD, V.; GUERRUCCI, D.; EMANUELLI, P.P; VERRIER, D. Simulator Development: the flexible approach applied to Operational Spacecraft Simulators. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2004), 8., 2004, Montreal, Canadá. **Proceedings...** 2004. Disponível em: <http://www.aiaa.org/spaceops2004archive/downloads/papers/SPACE2004sp-template00139F.pdf >. Acesso em: 29 nov 2011

ROLET, E.; CROENNE, D. SimGen: a future standard for simulations?. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS 2006), 9., 2006, Rome, Italy. **Proceedings...** 2006. Disponível em: <http://pdf.aiaa.org/getfile.cfm?urlX=5%3A7I%276D%26XZ%22C%2CS%40SUWT%5B%5EPK%3B%3A7%3AP%21%0A. Acesso em 29 nov 2011

SANTOS, W. A. **Adaptability, reusability and variability in software systems for space on-board computing**. 2008. 232 p. Tese (Doutorado em Engenharia Eletrônica e Computação). Instituto Tecnológico da Aeronáutica (ITA), São José dos Campos, 2008.

SOMMERVILLE, I. **Software engineering**. 6. ed., Addison-Wesley, 2010.

STELLATO, S.; ROMANI, E. Ground Segment Simulator: a tool for mission design and life cycle management. In: DATA SYSTEMS IN AEROSPACE CONFERENCE, 2005, Edinburgh, UK. **Proceedings...**, 2005. CD-ROM, p. 39. ISBN 92-9092-913-8

UML. **Unified modelling language version 2.4.1.** 2011. Disponível em <http://www.uml.org/>. Acesso em 14 Out 2011.

XML. **Extensible markup language version 1.1.** 2006. Disponível em <http://www.w3.org/TR/xml11/>. Acesso em 26 Ago 2011.

XSL. **Extensible stylesheet language version 1.1.** 2006. Disponível em <http://www.w3.org/TR/xsl/>. Acesso em 26 Ago 2011.

WILLIAMS, A. P. SIMSAT: An object oriented architecture for real-time satellite simulator. In: INTERNATIONAL SYMPOSIUM ON GROUND DATA SYSTEMS FOR SPACE MISSIONS OPERATIONS**,** 2., 1992, Pasadena, California, EUA. **Proceedings...**, 1992. p. 711-716. Disponível em: <http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19940019467_1994019467.pdf>

APÊNDICE A

O padrão SMP define um uma hierarquia de interfaces que facilitam a implementação de um simulador de satélites, contendo interfaces para os tipos e estruturas de dados, falhas, membros de dados, objetos, componentes, modelos, serviços, eventos, publicações, mecanismos de comunicação, etc. A Figura A.1 mostra a relação de todas as interfaces existentes no SMP.

- Smp::AnySimple
- Smp::Exception
  - Smp::DuplicateName
  - Smp::IArrayField::InvalidArraySize
  - Smp::IArrayField::InvalidArrayValue
  - Smp::IArrayField::InvalidIndex
  - Smp::IDynamicInvocation::InvalidOperationName
  - Smp::IDynamicInvocation::InvalidParameterCount
  - Smp::IDynamicInvocation::InvalidParameterType
  - Smp::IEventSource::AlreadySubscribed
  - Smp::IEventSource::InvalidEventSink
  - Smp::IEventSource::NotSubscribed
  - Smp::IField::InvalidFieldValue
  - Smp::IModel::InvalidModelState
  - Smp::InvalidAnyType
  - Smp::InvalidObjectType
  - Smp::IPersist::CannotRestore
  - Smp::IPersist::CannotStore
  - Smp::IPublication::InvalidFieldType
  - Smp::IRequest::InvalidParameterIndex
  - Smp::IRequest::InvalidParameterValue
  - Smp::IRequest::InvalidReturnValue
  - Smp::IRequest::VoidOperation
  - Smp::Management::IManagedContainer::ContainerFull
  - Smp::Management::IManagedModel::InvalidFieldName
  - Smp::Management::IManagedObject::InvalidObjectName
  - Smp::Management::IManagedReference::CannotRemove
  - Smp::Management::IManagedReference::NotReferenced
  - Smp::Management::IManagedReference::ReferenceFull
  - Smp::Management::IManagedSimulator::DuplicateUuid
  - Smp::Publication::ITypeRegistry::AlreadyRegistered
  - Smp::Publication::NotRegistered
  - Smp::Services::IEventManager::AlreadySubscribed
  - Smp::Services::InvalidEventId
  - Smp::Services::IScheduler::InvalidCycleTime
  - Smp::Services::IScheduler::InvalidEventTime
  - Smp::Services::NotSubscribed
- Smp::Management::IComponentCollection
  - Smp::Management::IManagedContainer
  - Smp::Management::IManagedReference
- Smp::IObject
  - Smp::IComponent
    - Smp::IAggregate
    - Smp::IComposite
      - Smp::ISimulator
        - Smp::Management::IManagedSimulator
    - Smp::IDynamicInvocation
    - Smp::IModel
      - Smp::IFallibleModel
      - Smp::Management::IManagedModel
    - Smp::IPersist
    - Smp::IService
      - Smp::Services::IEventManager
      - Smp::Services::ILinkRegistry
      - Smp::Services::ILogger
      - Smp::Services::IResolver
      - Smp::Services::IScheduler
      - Smp::Services::ITimeKeeper
    - Smp::Management::IEntryPointPublisher
    - Smp::Management::IEventConsumer
    - Smp::Management::IEventProvider
    - Smp::Management::IManagedComponent
      - Smp::Management::IManagedModel
  - Smp::IContainer
    - Smp::Management::IManagedContainer
  - Smp::IEntryPoint
    - Smp::ITask
  - Smp::IEventSink
  - Smp::IEventSource
  - Smp::IFactory
  - Smp::IFailure
  - Smp::IField
    - Smp::IArrayField
    - Smp::ISimpleField
      - Smp::IForcibleField
  - Smp::IReference
    - Smp::Management::IManagedReference
  - Smp::Management::IManagedObject
    - Smp::Management::IManagedComponent
  - Smp::Publication::IType
    - Smp::Publication::IEnumerationType
    - Smp::Publication::IStructureType
      - Smp::Publication::IClassType
- Smp::IPublication
- Smp::Publication::IPublishOperation
- Smp::IRequest
- Smp::IStorageReader
- Smp::IStorageWriter
- Smp::Publication::ITypeRegistry
- Smp::PrimitiveTypeValue
- Smp::Uuid

Figura A.1 - Relação das interfaces do SMP

APÊNDICE B

Neste apêndice serão mostrados os resultados dos demais cenários de verificação da arquitetura.

Conforme definido anteriormente, para cada cenário o OBDH foi composto por um ou mais componentes. Esta integração teve por objetivo verificar como se comportam os mecanismos do SMP, quando se substitui um componente por outro(s), contendo diferentes números de entradas ou saídas de dados. Para cada cenário será descrita a configuração adotada para o OBDH e os impactos em termos de comunicação dentro do simulador.

B.1 Cenário 1

Neste cenário, o objetivo é testar o serviço de agendamento de telecomando em um componente isolado.

O OBDH nesta configuração foi composto por um componente contendo o serviço de agendamento.

A Figura B.1 mostra a configuração do OBDH.

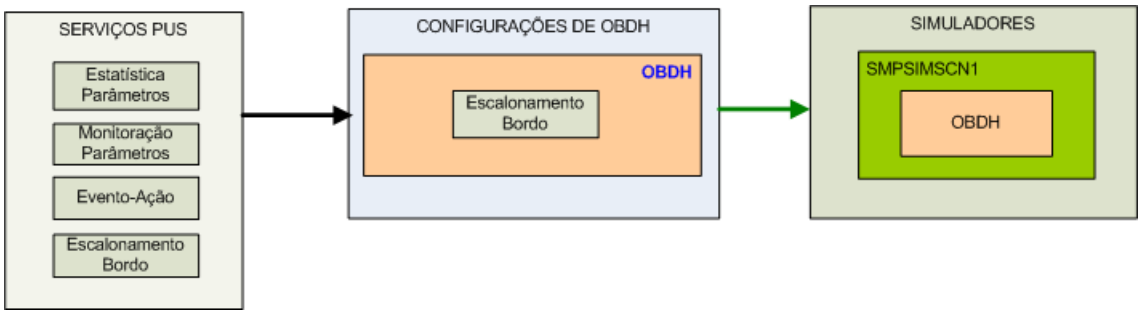


Figura B.1 – Configuração de serviços do OBDH do cenário 1

A Figura B.2 mostra os componentes inclusos na execução do cenário.

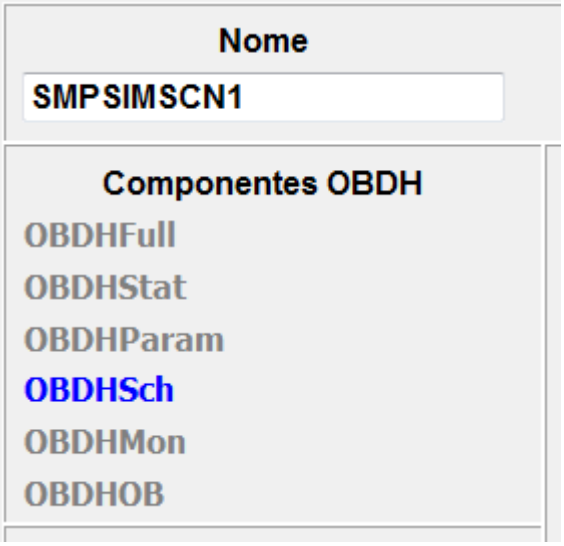


Figura B.2 – Componentes utilizados na execução do cenário 1

A Figura B.3 mostra a tela com o envio do telecomando e a hora para o qual o mesmo foi agendado.

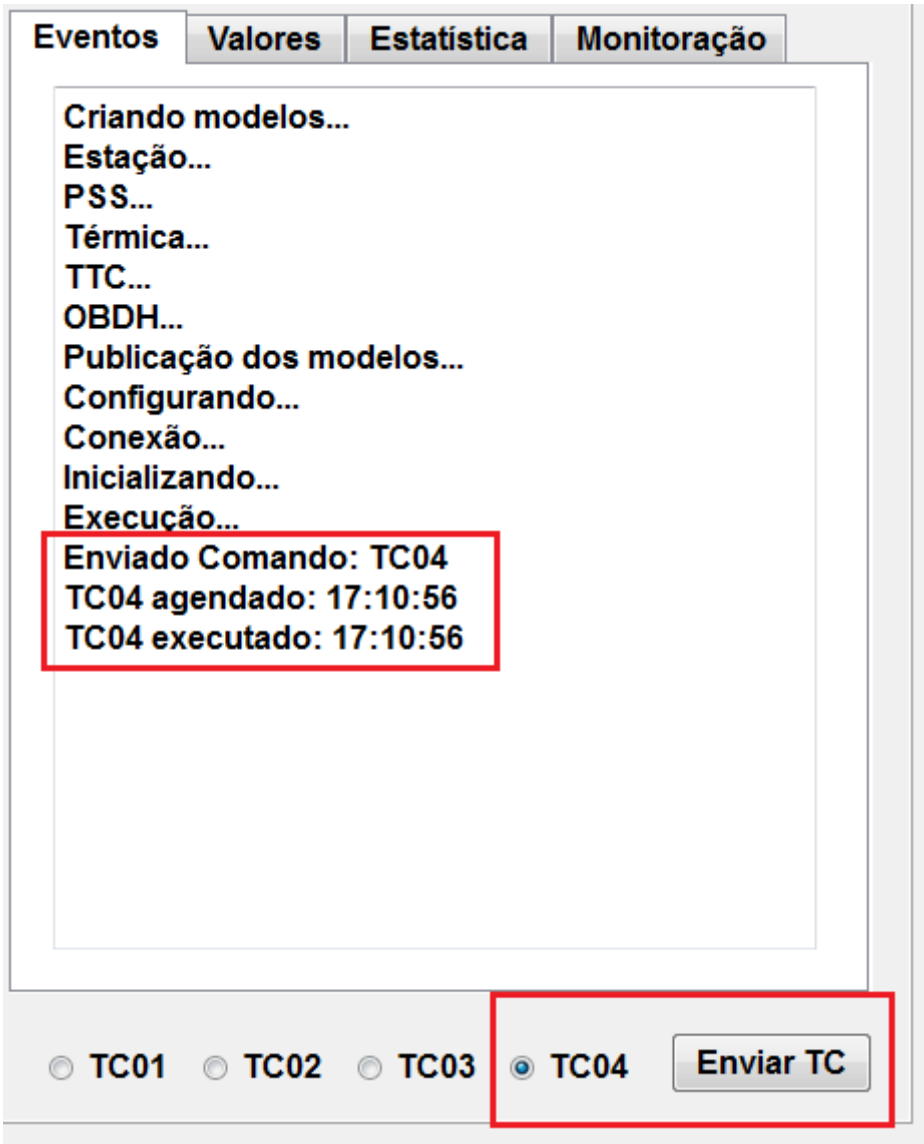


Figura B.3 – Agendamento e execução de telecomando

O código mostra como foi definida a comunicação entre este serviço no OBDH e o subsistema que recebeu o comando temporizado.

```
...

objTH->eventCurrent.Subscribe(&objOBDH->TCSink);

...
```

Na sequência, a comunicação tanto do agendamento quanto da execução pelo OBDH do comando.

```
...

objOBDH->srcOBDHTM.Subscribe(&objTTC->eventSink);

...
```

O subsistema destinatário do comando o recebeu corretamente e o mesmo foi executado no tempo previsto, alterando os valores dos parâmetros do subsistema de térmica, alvo do comando.

B.2 Cenário 2

Neste cenário, o objetivo é testar os serviços de agendamento de comando e estatísticas de parâmetros, ambos em componentes isolados.

O OBDH nesta configuração foi composto de dois componentes: um contendo o serviço de agendamento e outro contendo o serviço de estatísticas de parâmetros.

A modificação em relação ao cenário 1 foi a inclusão da comunicação entre o serviço de estatísticas e os modelos de térmica e PSS.

A Figura B.4 mostra o diagrama de configuração do OBDH.

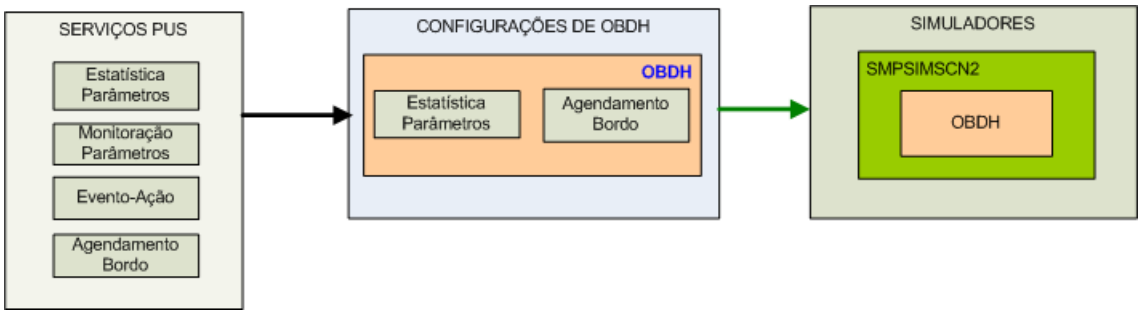


Figura B.4 – Configuração de serviços do OBDH do cenário 2

A Figura B.5 mostra os componentes inclusos na execução do cenário.

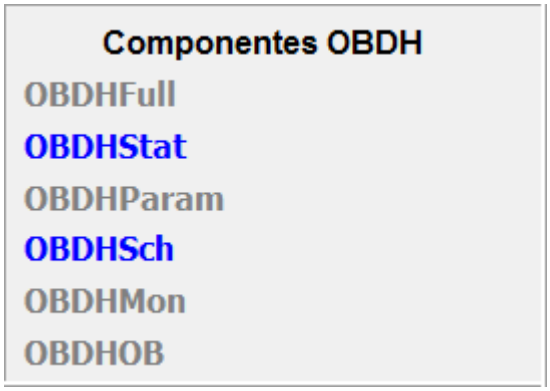


Figura B.5 – Componentes utilizados na execução do cenário 2

A Figura B.6 mostra a tela com o envio do telecomando e a hora para o qual o mesmo foi agendado.

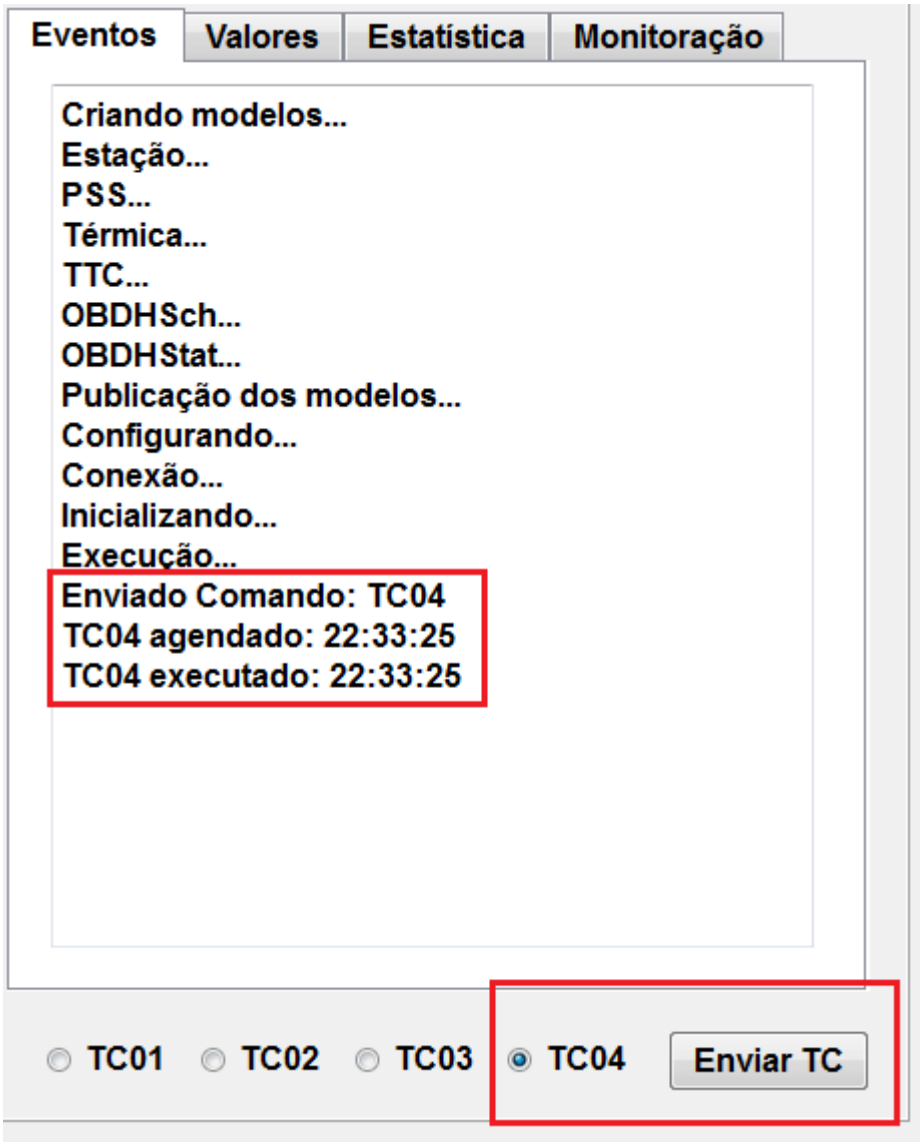


Figura B.6 – Agendamento e execução de telecomando

A Figura B.7 mostra a solicitação do relatório de estatística dos parâmetros, requisitados pelo envio do telecomando TC02.

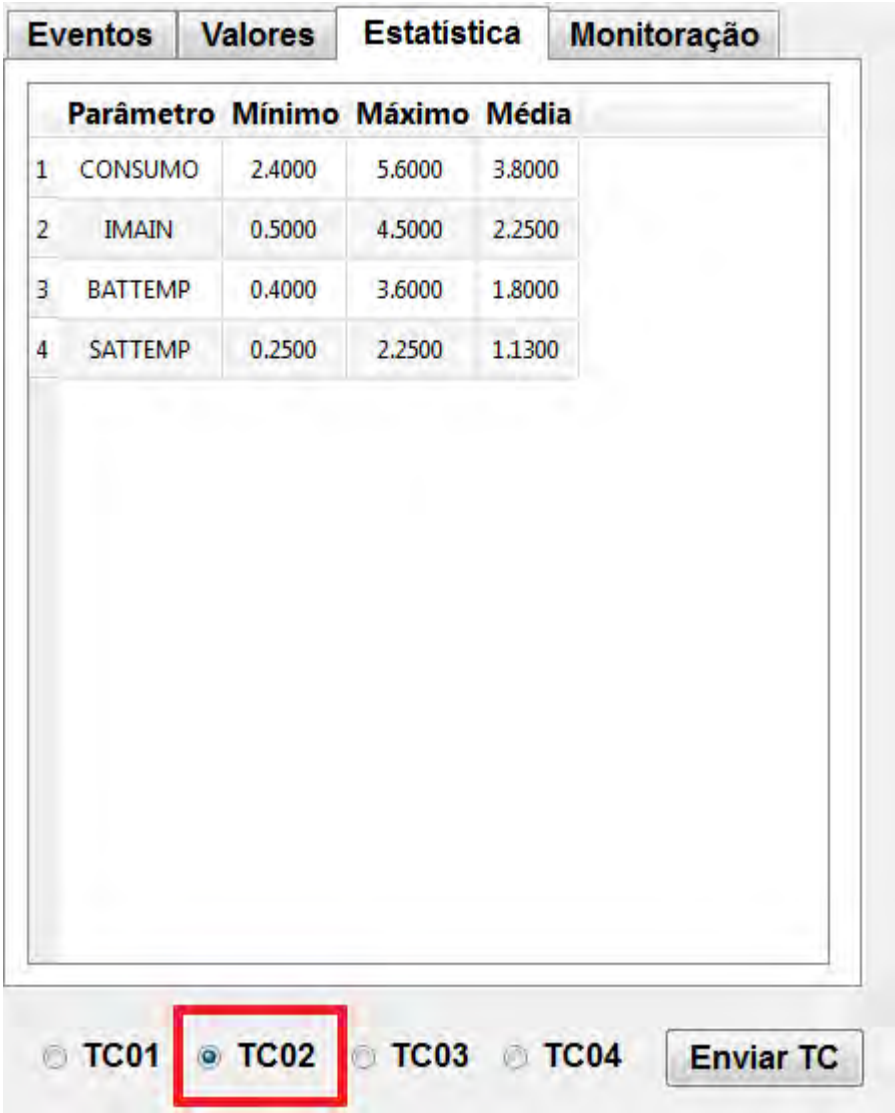


Figura B.7 – Relatório de estatística dos parâmetros

A vinculação entre o componente do OBDH que gera o relatório e o subsistema de TT&C que encaminha para a estação terrena para posterior visualização é feita conforme o código apresentado.

```
...

objOBDHStat->srcTM.Subscribe(&objTTC->eventSink);

...
```

Após a solicitação do relatório de estatística de parâmetros ter sido feita pelo *link* de comunicação, a mesma é retornada pela linha de comunicação definida abaixo.

```
...

objTTC->eventCurrent.Subscribe(&objOBDHStat->sinkTC);

...
```

A execução do comando temporizado ocorreu da mesma forma que no cenário 1. As estatísticas geradas para os parâmetros foram recuperadas através do envio do comando TC02, conforme mostrado na Figura B.7.

B.3 Cenário 4

Neste cenário, o objetivo é testar os serviços de agendamento de comando e monitoração de parâmetros em componentes isolados.

O OBDH nesta configuração foi composto de dois componentes: um contendo o serviço de agendamento e outro contendo o serviço de monitoração de parâmetros.

A Figura B.8 mostra o diagrama de blocos do cenário.

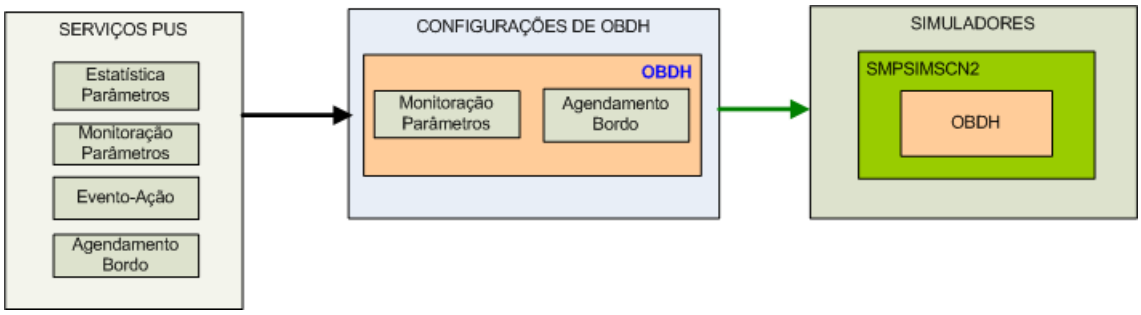


Figura B.8 – Configuração de serviços do OBDH do cenário 4

A Figura B.9 mostra os componentes que forma usados na simulação.

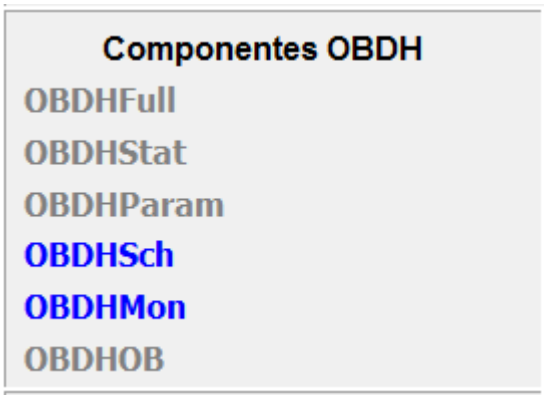


Figura B.9 – Componentes utilizados na execução do cenário 4

A Figura B.10 mostra o a execução do telecomando temporizado.

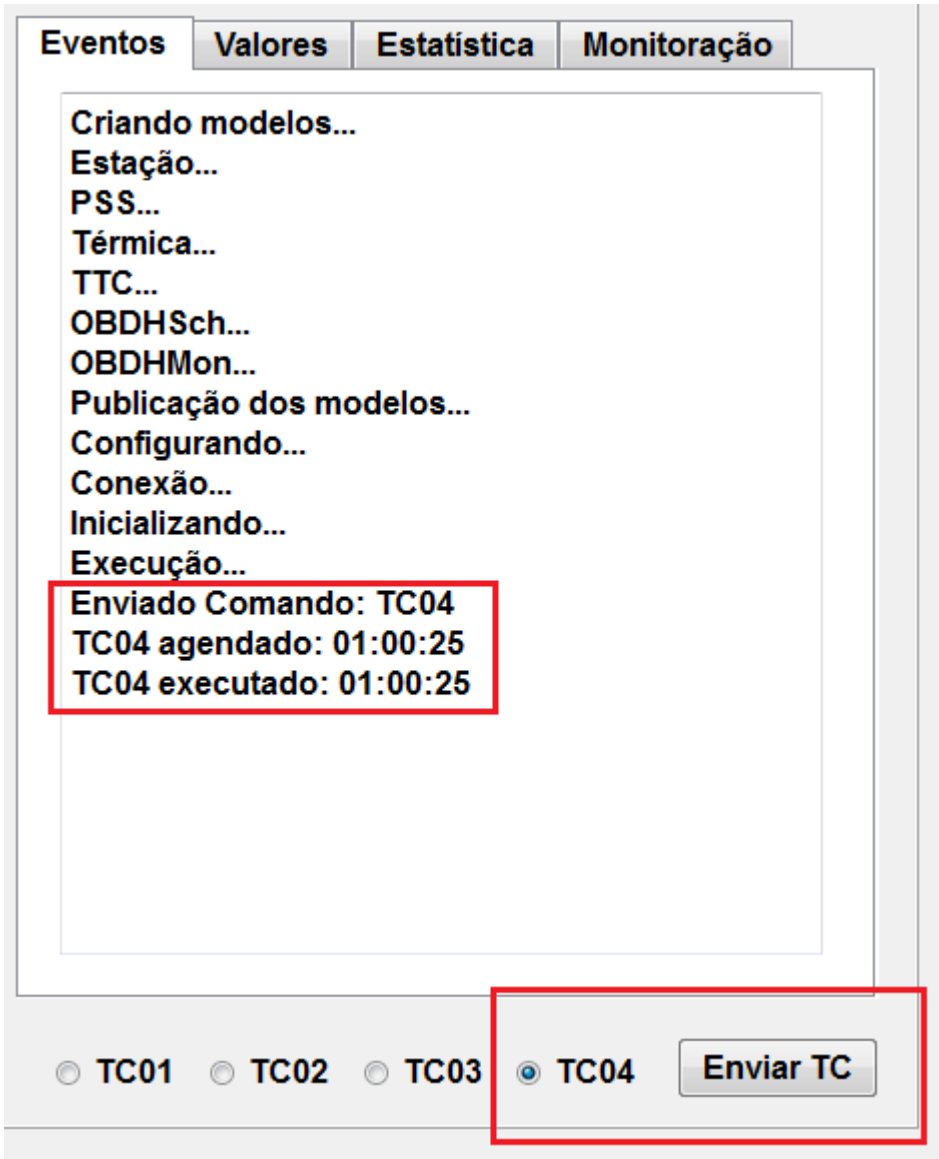


Figura B.10 – Agendamento e execução de telecomando

A Figura B.11 mostra a solicitação do relatório de estatística dos parâmetros, requisitados pelo envio do telecomando TC03.

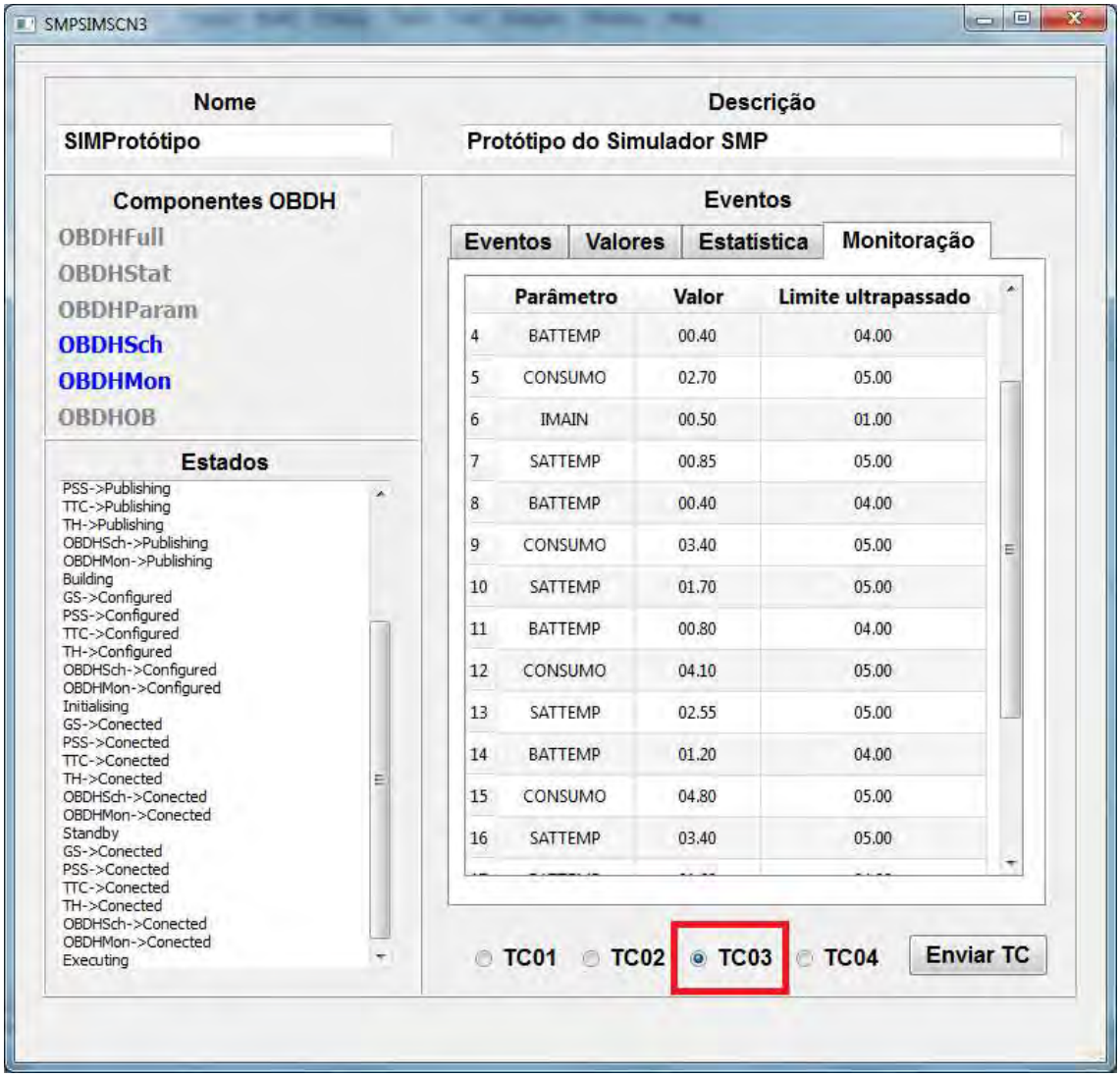


Figura B.11 – Relatório de estatística dos parâmetros

O vínculo entre o agendamento e sua execução com o serviço de térmica foi feito conforme mostrado no código a seguir.

```
...

objOBDHSch->eventTH.Subscribe(&objTH->eventSink);

...
```

Já entre a monitoração e os subsistemas, repetiu-se a mesma comunicação mostrada no cenário 3.

B.4 Cenário 5

Neste cenário, o objetivo é testar os serviços de agendamento de telecomando e estatísticas de parâmetros em componentes isolados, porém com fidelidade superior aos componentes utilizados no cenário 3, detalhado no Capítulo 6.2.

O OBDH nesta configuração foi composto de dois componentes: um contendo o serviço de monitoração de parâmetros e outro contendo o serviço de evento-ação.

Nenhuma modificação foi necessária nas vinculações feitas para execução do cenário 3. Apenas novos fluxos de comunicação de dados tiveram de ser definidos, para implementar os vínculos entre as novas entradas e saída dos componentes. Porém, a reutilização das interfaces foi total.