Tese apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica, como parte dos requisitos para obtenção do título de Mestre em Ciências no Curso de Engenharia Eletrônica e Computação, Área de Informática

Juliano de Almeida Monte-Mor

FÊNIX - UM FRAMEWORK PARA SIMULAÇÃO DE TRAJETÓRIAS DE FOGUETES DE SONDAGEM

Tese aprovada em sua versão final pelos abaixo assinados:

Prof. Dr. Adilson Marques da Cunha
Orientador

Prof. Dr. Homero Santiago Maciel
Pró-Reitor de Pós-Graduação e Pesquisa

Campo Montenegro
São José dos Campos, SP - Brasil
2007

Dados Internacionais de Catalogação-na-Publicação (CIP)

Divisão Biblioteca Central do ITA/CTA

de Almeida Monte-Mor, Juliano

Fênix - Um Framework para Simulação de Trajetórias de Foguetes de Sondagem / Juliano de Almeida Monte-Mor.

São José dos Campos, 2007.

220f.

Tese de Mestrado – Curso de Engenharia Eletrônica e Computação. Área de Informática – Instituto Tecnológico de Aeronáutica, 2007. Orientador: Prof. Dr. Adilson Marques da Cunha. .

1. Simulação Computacional. 2. Trajetória. 3. High Level Architecture - HLA. 4. Veículos Espaciais de Sondagem. I. Centro Técnico Aeroespacial. Instituto Tecnológico de Aeronáutica. Divisão de Ciência da Computação. II. Título.

REFERÊNCIA BIBLIOGRÁFICA

DE ALMEIDA MONTE-MOR, Juliano. **Fênix - Um Framework para Simulação de Trajetórias de Foguetes de Sondagem**. 2007. 220f. Tese de Mestrado – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Juliano de Almeida Monte-Mor

TITULO DO TRABALHO: Fênix - Um Framework para Simulação de Trajetórias de Foguetes de Sondagem.

TIPO DO TRABALHO/ANO: Tese / 2007

Juliano de Almeida Monte-Mor

Av. Julita Pires Bretas, 329, apto. 301 - Santa Helena

CEP 35.170-026 – Coronel Fabriciano - MG

FÊNIX - UM FRAMEWORK PARA SIMULAÇÃO DE TRAJETÓRIAS DE FOGUETES DE SONDAGEM

Juliano de Almeida Monte-Mor

Composição da Banca Examinadora:

Prof. Dr.	Nei Yoshihiro Soma	Presidente	-	ITA
Prof. Dr.	Adilson Marques da Cunha	Orientador	-	ITA
Prof. Dr.	Luiz Alberto Vieira Dias	Membro	-	ITA
Prof. Dr.	Sérgio Roberto Matiello Pellegrino	Membro	-	ITA
Prof. Dr.	Atair Rios Neto	Membro Externo	-	INPE

ITA

Dedico este trabalho primeiramente a Deus, por sua bondade e misericórdia, e depois a meus pais Divino e Maria José, por todo amor e ensinamento. Como homenagem também dedico àquele que possibilitou a sua realização, ao mestre Prof. Vakulathil Abdurahiman *(in memoriam)*.

Agradecimentos

A meus irmãos Roberto e Patrícia,
por sempre apoiarem o meu crescimento profissional.

A Aline pelo companheirismo e carinho durante os momentos difíceis.
Aos meus amigos por todo incentivo e ajuda, em especial a Gláucia, Laércio,
Giovani, Osvandre, Pascaly, Glauston e Washington.

Aos Profs. Dr. Vakulathil Abdurahiman e Dr. Adilson Marques da Cunha,
por todo aprendizado e confiança.

Ao Instituto de Aeronáutica e Espaço - IAE, principalmente a Lais, Sidney,
Raul, João e Alexandre, profissionais do ASE-V, por toda ajuda e orientação.

A Fundação Casimiro Montenegro Filho - FCMF, pelo apoio financeiro durante
o período de residência em São José dos Campos.

"If I have seen farther than others,

it is because I stood on the shoulders of giants."

— Sir Isaac Newton

Resumo

Cada vez mais os métodos, técnicas e ferramentas de Engenharia de *Software* têm sido aplicados para criação, manutenção e adaptação de sistemas de *software* aeroespaciais. A técnica de orientação a objetos tem provido o reuso de componentes de *software*, reduzindo o esforço para desenvolver aplicações similares num domínio específico. Este trabalho de pesquisa propõe um *Framework* para a Simulação de Trajetórias de Foguetes de Sondagem. Este *Framework* define um modelo abstrato para a construção de simuladores de trajetórias. Um Estudo de Caso e um Protótipo de Simulador foram desenvolvidos para avaliar a reusabilidade do *Framework* proposto. O protótipo provê cálculos de movimento de um foguete com um, três e seis graus de liberdade. Neste trabalho, também é avaliado o esforço necessário de incorporação da arquitetura de simulação distribuída definida no padrão *IEEE STD 1516 - High Level Architecture - HLA*. Uma análise dos principais resultados desta pesquisa propicia uma verificação do *Framework* proposto, quanto a reutilização significativa de código para a construção de modelos de simuladores de trajetórias. Esta análise também propicia uma avaliação do padrão HLA para a criação de simulações distribuídas, visando uma melhor padronização, interação e reusabilidade dos sistemas.

Palavras–Chave: Simulação Computacional, Trajetória, Foguetes de Sondagem, *High Level Architecture - HLA*

Abstract

Software Engineering methods, techniques and tools have been increasingly applied to build, maintain and adapt aerospace software systems. The Object-Oriented technique has provided software component reuse, decreasing the effort to develop similar application on specific domains. This research proposes a Framework for Sounding Rockets Trajectories Simulations. The Framework defines an abstract model to build trajectories simulators models. A Case Study and a Simulator Prototype were developed to evaluate the Framework reusability. The Prototype provides rocket movements calculations with one, three and six degrees of freedom. This work also provides the evaluation of needed effort to incorporate the distributed simulation architecture defined in IEEE STD 1516 - High Level Architecture (HLA). An analysis of the main research results provides a Framework verification, considering source code reusability for constructing trajectories simulators models. This analysis also provides an assessment of HLA standard to build distributed simulations, aiming better systems standardization, interaction, and reusability.

Keywords: *Computer Simulation, Trajectory, Rockets, High Level Architecture - HLA*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ALSP	*Aggregate Level Simulation Protocol*
API	Application Program Interface
ASE	Divisão de Sistemas Espaciais
CASE	*Computer Aided Software Engineering*
CDU	Caso De Uso
CTA	Comando Geral de Tecnologia Aeroespacial
DARPA	*Defense Advanced Research Projects Agency*
DCU	Diagrama de Casos de Uso
DDM	*Data Distribuition Management*
DIS	*Distributed Interactive Simulation*
GUI	*Graphical User Interfaces*
FED	*Federation Execution Data*
FEN	Fênix
FOM	*Federation Object Model*
HLA	*High Level Architecture*
IAE	Instituto de Aeronáutica e Espaço
IEEE	*Institute of Electrical and Electronics Engineers*
INPE	Instituto Nacional de Pesquisas Espaciais

ITA	Instituto Tecnológico de Aeronáutica
LIS	*Language Independent Specification*
M&S	*Modeling and Simulation*
NASA	*National Aeronautics and Space Administration*
OMT	*Object Model Template*
OO	Orientado a Objeto
OOAD	*Object Oriented Analysis and Design*
P&D	Pesquisa e Desenvolvimento
PNAE	Programa Nacional de Atividades Espaciais
POO	Programação Orientada a Objetos
RF	Requisitos Funcionais
RNF	Requisitos Não-Funcionais
ROSI	*Rocket Simulation*
RS	Restrição
RTI	*Runtime Infrastructure*
SIMNET	*Simulator Networking*
SIM	SIMSonda
SOM	*Simulation Object Model*
STD	*Standard*
UML	*Unified Modeling Language*
US DoD	*United States Departament of Defense*

1 Introdução

A atividade espacial contribui de maneira significativa para o desenvolvimento tecnológico e econômico de um país. Nesse sentido, o Brasil possui um Programa Nacional de Atividades Espaciais (PNAE), que tem por objetivo capacitar o país para desenvolver e utilizar tecnologias espaciais na solução de problemas nacionais e em benefício da sociedade brasileira (AEB, 2005).

Em termos de Pesquisa e Desenvolvimento (P&D), o PNAE incentiva o desenvolvimento de projetos em áreas correlatas à Espacial como Matemática Aplicada e Computacional, que possam diretamente contribuir para o avanço da ciência e da tecnologia espaciais. Assim, torna-se interessante a aplicação de métodos, técnicas e ferramentas já consagradas em outros domínios de conhecimento para o desenvolvimento de sistemas no âmbito militar e espacial.

A simulação computacional tem sido extensivamente aplicada ao longo dos anos numa significativa variedade de problemas militares. Existem no mundo muitas organizações e agências envolvidas na modelagem e simulação militar em diferentes áreas de aplicação. A simulação tornou-se uma metodologia indispensável para a solução de muitos problemas do mundo real.

Simulação é o processo de elaborar um modelo de um sistema real e conduzir expe-

rimentos com este modelo, tendo como propósito a compreensão do comportamento do sistema e a avaliação de diversas estratégias (SHANNON, 1975).

Segundo ANDRADÓTTIR (1998), quando o desempenho de um sistema de interesse depende da escolha de valores dos parâmetros de entrada, a simulação pode ser usada na tentativa de se determinar os valores ótimos desses parâmetros, possivelmente sujeitos a algumas restrições.

No setor aeroespacial, a simulação tem sido usada como uma ferramenta de auxílio na preparação de missões espaciais, pois o seu uso pode permitir a análise de desempenho dos veículos espaciais para diferentes estratégias de lançamento. O interesse maior deste trabalho concentra-se na simulação das trajetórias de foguetes de sondagem. Estes veículos são utilizados para missões sub-orbitais de exploração do espaço capazes de lançar cargas úteis compostas por um ou mais experimentos científicos e tecnológicos (AEB, 2007).

Na preparação de uma missão, a definição da trajetória a ser percorrida por um veículo, denominada trajetória nominal, tem influência direta sobre o sucesso da missão. A trajetória nominal é utilizada como referência para a configuração dos meios envolvidos no rastreio e na coleta de dados e dos procedimentos de guiagem e controle do veículo. Esta trajetória corresponde a previsão do caminho que o veículo percorre no espaço aéreo num período de tempo (LOUIS, 2006).

Geralmente, são realizadas várias simulações para a definição de uma trajetória nominal, pois a sua determinação deve obedecer, além das regras de segurança, ao propósito da missão que o veículo espacial deve cumprir (LOUIS, 2006). Por meio de simulações, possibilita-se a análise da viabilidade de uma trajetória, bem como a análise de parâmetros do projeto de um veículo espacial.

Além de auxiliar na definição de trajetórias espaciais, simuladores de trajetórias de foguetes também podem ser utilizados em outras aplicações. Um simulador poderia, por exemplo, ser parte de um ambiente para treinamento de uma equipe de segurança de vôo, ou de astronautas em caso de veículos tripulados. Neste caso, o simulador fornece diversos dados como posição e velocidade do veículo em função do tempo de vôo, permitindo o treinamento das equipes em condições normais, nominais e de falhas, além do teste integrado de diversos subsistemas, equipamentos, comunicações e configurações.

Estas atividades de treinamento e simulação das operações de lançamento devem ser executadas de forma exaustiva com as pessoas e os meios envolvidos nas missões, de forma a verificar se todas as funções serão cumpridas conforme o previsto, além de reduzir a ocorrência de falhas e atingir os padrões de segurança exigidos, evitando assim, desnecessárias destruições de veículos e suas cargas úteis, bem como riscos e prejuízos diversos.

Apesar da diversidade de aplicações da simulação de trajetórias de veículos espaciais, muitos modelos de simulação têm sido desenvolvidos conforme a necessidade, e de forma isolada e específica para a aplicação em questão, o que prejudica o seu reuso e aumenta o custo envolvido na construção, manutenção e adaptação de simuladores.

Considerando-se que as diferentes aplicações da simulação da trajetória de foguetes requerem o desenvolvimento de simuladores com características bastante similares, torna-se viável a construção de um modelo genérico de simulação de trajetórias de veículos espaciais.

Um modelo genérico de simulação pode ser usado para a construção de vários sistemas num determinado domínio, possibilitando a análise do impacto operacional de decisões de projeto. Devido ao uso geral da referida abordagem, o modelo desenvolvido deve ser aplicável a todas as instâncias no domínio escolhido e ainda ser capaz de capturar detalhes

específicos suficientes do sistema em questão (STEELE *et al.*, 2002).

Com o surgimento do paradigma da orientação a objetos, a tecnologia para desenvolvimento de modelos genéricos e reuso de componentes de *software* resultou na definição de um *framework*, i.e., um esqueleto de solução para uma classe de problemas similares (BUDD, 2002). Um *framework* pode ser usado como a base para o desenvolvimento de várias aplicações num mesmo domínio (MATTSSON, 1996).

Além de permitir o reuso de um modelo de simulação, um *framework* também pode possibilitar o reuso da infra-estrutura para a execução da simulação. Com este intuito, foi criado o *IEEE STD 1516 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)* (IEEE, 2000a; IEEE, 2000b; IEEE, 2000c; IEEE, 2003). Este padrão foi desenvolvido pelo *US Department of Defense - DoD*, através de um esforço conjunto entre o governo, o meio acadêmico e a indústria. O HLA combinou duas tecnologias antecessoras, *Distributed Interactive Simulation - DIS* e *Aggregate Level Simulation Protocol - ALSP*, sendo sua definição iniciada em 1995 (TRIVELATO, 2003). Trata-se de um *framework* para o desenvolvimento de sistemas de simulação num alto nível de abstração, através da especificação de interfaces e regras de projeto (KUHL; WEATHERLY; DAHMANN, 1999).

Uma das idéias básicas do padrão HLA é manter a funcionalidade da simulação separada da infra-estrutura de suporte à simulação. Este padrão foi projetado para facilitar a interoperabilidade entre simulações e promover o reuso de simulações e seus componentes. O HLA exige que as simulações forneçam a funcionalidade externa necessária para interagir com outras simulações por meio da infra-estrutura de suporte, não fazendo nenhuma especificação sobre a estrutura interna das simulações (KUHL; WEATHERLY; DAHMANN, 1999).

1.1 Objetivo

Este trabalho de pesquisa tem por objetivo principal desenvolver um *Framework* para a Simulação de Trajetórias de Foguetes de Sondagem, visando facilitar a construção, manutenção e adaptação de aplicações envolvendo suas trajetórias, e reduzir o desperdício dos recursos envolvidos.

Além disto, este trabalho tem por objetivo secundário avaliar o esforço para a adoção do padrão *IEEE STD 1516* na construção de sistemas de simulação distribuídos, visando auxiliar os profissionais relacionados com estes sistemas na escolha da tecnologia adequada ao seu projeto, e reduzir o desperdício de recursos envolvidos.

O enfoque deste trabalho não compreende a determinação do melhor modelo matemático para o cálculo da trajetória de um foguete de sondagem e nem a pesquisa de métodos numéricos para definição de uma trajetória nominal ótima, mas a criação de um componente de *software* reusável para a simulação de trajetórias destes foguetes. Assumiu-se, como premissa, que estão corretos os modelos de equações, descritos em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976), que serviram de base para a criação do *framework*.

1.2 Organização

Esta dissertação de mestrado foi divida conforme apresentado na Figura 1.1. O Capítulo 2 fornece o referencial teórico necessário para o desenvolvimento deste projeto, abordando *frameworks* orientados a objetos e *High Level Architecture - HLA*. O Capítulo 3 apresenta informações a respeito da simulação de trajetórias de foguetes de sondagem.

No Capítulo 4 descreve o modelo de um *framework* para simulação de trajetórias

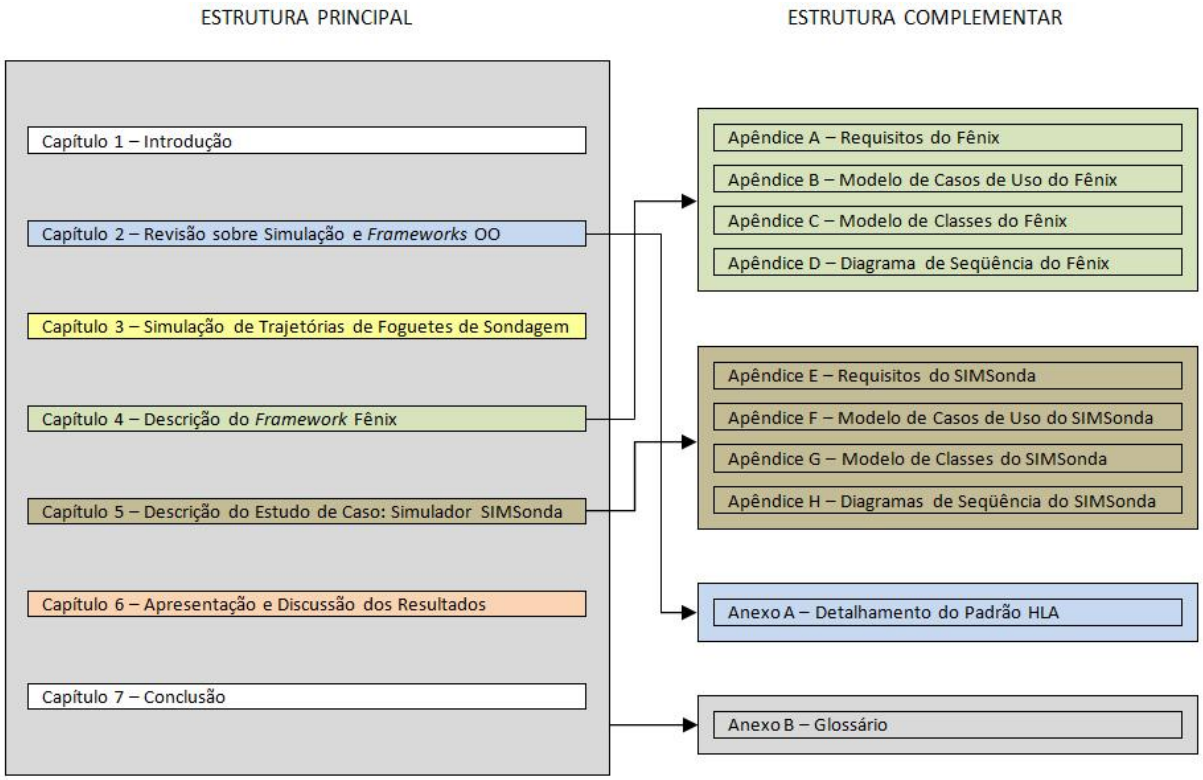

FIGURA 1.1 – Estrutra do Relatório de Pesquisa

de foguetes de sondagem, proposto como tema deste projeto de pesquisa. O Capítulo 5 descreve como reusar o *framework* proposto, através do desenvolvimento de um Estudo de Caso, englobando a criação de um protótipo de simulador, e de uma simulação distribuída com base no padrão HLA. Os resultados obtidos com a reusabilidade do *framework* e com a adoção do padrão HLA são apresentados e discutidos no Capítulo 6. Finalmente, o Capítulo 7 discorre sobre a conclusão e apresenta as contribuições, recomendações e sugestões de trabalhos futuros, pois aqui não se esgota o assunto em voga.

Complementam este documento alguns artefatos, i.e., qualquer tipo de informação criada, produzida, alterada ou usada pelos responsáveis no desenvolvimento de um sistema (RESENDE, 2006). Os Apêndices A a D abordam, respectivamente, os requisistos, os casos de uso, as classes e um diagrama de seqüência do *Framework* descrito no Capítulo 4. Da mesma forma, os Apêndices E a H apresentam, respectivamente, os requisistos, os casos de

uso, as classes e os diagramas de seqüência do Estudo de Caso descrito no Capítulo 5. O Anexo A fornece algumas informações adicionais sobre o padrão HLA. E para finalizar, no Anexo B definiu-se o glossário de termos e palavras utilizados neste relatório de pesquisa.

2 Simulação Computacional e *Frameworks* OO

Este Capítulo apresenta o embasamento teórico para o desenvolvimento desta pesquisa. A Seção 2.1 aborda conceitos de Simulação Computacional e uma visão geral do padrão IEEE STD 1516. Já a Seção 2.2 apresenta os fundamentos de *Frameworks* Orientados a Objetos (OO). E por fim, na Seção 2.3, são discutidos alguns trabalhos existentes considerados relevantes.

2.1 Simulação

Uma simulação pode ser entendida como a imitação da operação de um sistema ou processo do mundo real em relação ao tempo, envolvendo a criação de uma história artificial do sistema e a sua observação para extrair inferências relacionadas a características operacionais representadas. É um importante processo para a resolução de muitos problemas do mundo real (BANKS, 1998).

Os métodos de simulação computacional têm sido desenvolvidos e freqüentemente usados, desde o início da década de 60. Como princípio básico, um analista constrói um modelo de um sistema de interesse, escreve os programas que o contêm e usa um

computador para imitar o seu comportamento quando sujeito a uma variedade de políticas de operações, escolhendo a mais desejada (PIDD, 1997).

O uso da simulação computacional permite a experimentação de um modelo para demonstrar o provável efeito de várias políticas, e aquelas que produzirem os melhores resultados podem ser implementadas no sistema real. A idéia básica deste processo de experimentação é apresentada na Figura 2.1.

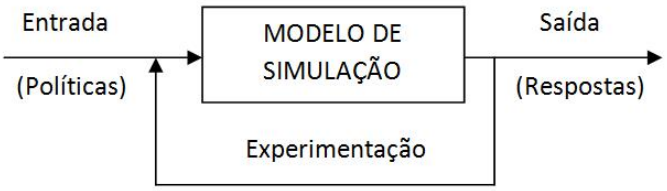

FIGURA 2.1 – Simulação como experimentação. (Adaptada de PIDD, 1997)

Este processo de experimentação torna-se interessante na preparação de missões de lançamento de veículos espaciais. Em particular, pode-se criar um simulador que experimente várias configurações para a definição de uma trajetória nominal, a fim de escolher aquela que garanta o cumprimento com sucesso dos objetivos pré-estabelecidos, sem comprometer a segurança das pessoas e meios envolvidos.

Além disso, um simulador de trajetórias pode interagir com outros sistemas compondo um ambiente de treinamento e simulação de missões espaciais. Para permitir a interoperabilidade deste simulador com outros simuladores e/ou aplicações pode-se utilizar o *IEEE STD 1516 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA).*

Com o uso deste padrão torna-se possível, por exemplo, interligar simulações da trajetória de um foguete com um sistema de visualização em três dimensões, que facilite a análise visual da atitude do foguete em vôo. O padrão HLA ajuda a combinar esses

sistemas numa simples mas abrangente simulação.

Como pretende-se avaliar a adoção do *IEEE STD 1516* para integração de sistemas num ambiente de treinameno e simulação, a Seção 2.1.1 introduz os seus conceitos principais, sendo maiores detalhes apresentados no Anexo A.

2.1.1 *High Level Architecture (HLA)*

As especificações do padrão HLA consideradas em conjunto definem um *framework* para construção e execução de simulações em grande escala. Os seguintes termos derivam-se do padão HLA:

- **Federação** - um sistema de simulação criado a partir da união de simulações;
- **Federado** - cada simulação combinada para formar a Federação; e
- **Execução da Federação** - uma sessão de execução simultânea de todos os Federados.

Numa Federação, qualquer número de sistemas de simulação (Federado), geralmente distribuído fisicamente, pode ser reunido num ambiente de simulação unificado para endereçar as necessidades de novas aplicações.

O *IEEE STD 1516* divide-se em quatro documentos, descritos a seguir:

- **1516. *Framework* e Regras** (IEEE, 2000a) - um conjunto de regras que descrevem o princípio geral definindo a arquitetura do padrão HLA.
- **1516.1. Especificação da Interface de Federado** (IEEE, 2000b) - uma descrição da interface funcional entre Federados e a Infra-estrutura de Execução, denominada *Runtime Infrastructure (RTI)*;

- **1516.2. Especificação do Gabarito de Modelo de Objeto (*Object Model Template - OMT*)** (IEEE, 2000c) - uma especificação de formato e estrutura para documentar modelos de objetos; e

- **1516.3. Processo de Desenvolvimento e Execução de Federação** (IEEE, 2003) - práticas recomendadas para a construção e execução de Federações.

Segundo Shaw e Garlan (1996), a definição abstrata de uma arquitetura de *software* envolve a descrição de: elementos usados para a construção de sistemas, interações entre esses elementos, padrões que guiam a criação do sistema, e restrições sobre esses padrões.

As partes que formam a HLA mapeiam elementos, interações e padrões da seguinte maneira (KUHL; WEATHERLY; DAHMANN, 1999):

- **Elementos** - as regras HLA e a especificação da interface HLA definem os elementos de uma Federação como os Federados, a RTI e o modelo comum de objetos;

- **Interações** - as regras HLA e a especificação da interface HLA definem as interações dos Federados com a RTI. A interação entre Federados sempre ocorre por intermédio da RTI. Para uma dada Federação, o seu modelo de objetos define os tipos de dados transmitidos nas interações entre Federados e a RTI. O OMT é um meta-modelo para todos os Modelos de Objetos da Federação (*Federation Object Model - FOM*), ou seja, ele define a estrutura válida de cada FOM; e

- **Padrões** - os padrões de composição permitidos no HLA são restringidos pelas regras HLA e definidos pela especificação da interface HLA.

2.1.1.1 Visão Funcional

Uma Federação corresponde a um "contrato" entre Federados que concordam em executar uma simulação distribuída, especificando "como" e "o quê" será simulado, sendo constituída dos seguintes elementos (KUHL; WEATHERLY; DAHMANN, 1999):

- Um *software* de suporte;
- Um modelo comum de objetos para a troca de dados; e
- Algum número de Federados.

A RTI corresponde ao *software* responsável pela execução simultânea da Federação, permitindo que os Federados executem juntos para alcançar os objetivos da Federação. Durante a execução de uma Federação, um Federado pode se conectar a RTI com o intuito de tornar-se seu membro.

Os Federados podem consistir de diversos processos, executando em diversos computadores. Um Federado pode representar uma plataforma, como um simulador de *cockpit*, ou uma simulação agregada, como a simulação completa do fluxo de tráfego áereo nacional. Ele também pode representar uma única entidade, como um veículo, ou modelar diversas entidades, como todos os veículos numa cidade. Ele ainda pode representar ferramentas utilitárias ou aplicações para suportar participantes humanos na simulação.

Dessa forma, Federados reunem-se numa Federação e são vistos como componentes na arquitetura do padrão HLA, conforme observado na Figura 2.2.

Numa Federação envolvendo operações de lançamento, um dos Federados corresponderá a um simulador de trajetórias de foguetes, sendo os outros federados definidos de

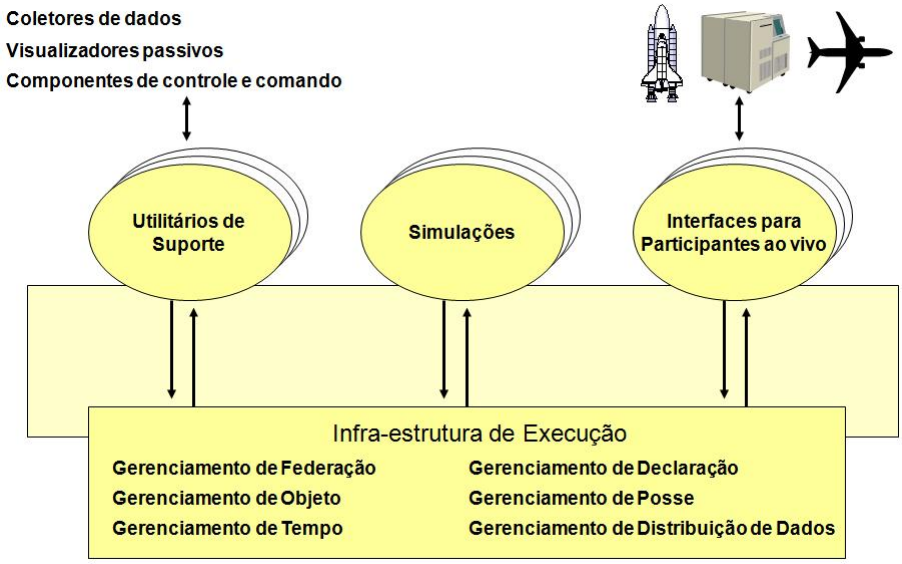

FIGURA 2.2 – Componentes de *software* no HLA. (Adaptada de KOMPETENZZENTRUM HLA, 2003)

acordo com os objetivos da Federação. Diferentes Federados serão necessários em Federações com diferentes propósitos, seja para treinamentos de equipes, para visualização de trajetórias ou para testes de equipamentos, dentre outras.

2.1.1.2 Especificação da Interface

O documento *IEEE STD 1516.2* apresenta uma especificação independente de linguagem (*Language Independent Specification - LIS)* para suportar comunicação entre simulações num domínio de simulação distribuída. Esta especificação define as interfaces e serviços padrão usados pelos Federados para suportar trocas eficientes de informações quando participam de uma execução distribuída de Federação.

O uso desses serviços e interfaces padrão aumenta a capacidade para reuso de Federados individuais. A RTI provê serviços para os Federados numa forma análoga a um sistema operacional distribuído, provendo serviços às aplicações. As interfaces são organizadas em sete grupos de serviços básicos para gerenciamento de: federação, declaração,

objeto, posse, tempo, distribuição de dados e serviços de suporte.

Essas interfaces encontram-se baseadas num paradigma "Embaixador" *(Ambassador)*. Um Federado comunica-se com a RTI usando seu "Embaixador-RTI" *(RTI-Ambassador)* e, da mesma forma, a RTI comunica-se com um Federado através de seu "Embaixador-Federado" *(Federate-Ambassador)* (KOMPETENZZENTRUM HLA, 2003). Estes "Embaixadores" correspondem a objetos em linguagens orientadas a objetos e a comunicação entre eles ocorre normalmente através da chamada de métodos associados a esses objetos. A Figura 2.3 ilustra esta idéia.

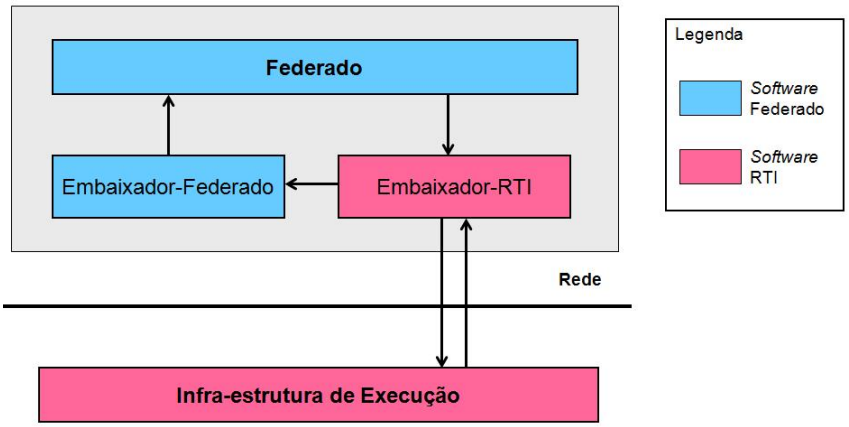

FIGURA 2.3 – O paradigma "Embaixador". (Adaptada de KOMPETENZZENTRUM HLA, 2003)

A definição dos serviços na especificação das interfaces não faz nenhuma referência a qualquer linguagem de programação, apenas apresenta Interfaces de Programação de Aplicação (*Application Program Interface - API*) em várias linguagens. O padrão HLA requer que as comunicações entre Federados usem uma API padrão, provendo uma especificação para as interfaces funcionais entre os Federados e a RTI. Esta espeficicação foi proposta como uma API em diferentes formas incluíndo CORBA IDL, C++, Ada95 e Java (BUSS; JACKSON, 1998).

O padrão HLA define uma arquitetura e não uma implementação. A especificação da

interface é abstrata, definindo não apenas a interface que a RTI apresenta para os Federados, mas também a que os Federados apresentam para a RTI. Diferentes implementações da RTI e dos Federados podem atender a interface como especificada.

Esta padronização das interfaces de comunicação facilita o desenvolvimento, a manutenção e a evolução da RTI e dos Federados. Não faz parte do escopo deste trabalho o desenvolvimento de uma RTI. Para implementar os Federados torna-se necessário apenas o desenvolvimento dos métodos abstratos especificados em sua interface.

2.1.1.3 Gabarito de Modelo de Objeto

O documento *IEEE STD 1516.3* descreve o *Object Model Template (OMT)*, que padroniza modelos de objetos, com o objetivo de facilitar a interoperabilidade entre simulações e o reuso dos componentes de uma simulação.

Os modelos de objetos descrevem um Federado, através de um Modelo de Objetos da Simulação (*Simulation Object Model - SOM*), ou uma Federação, através do Modelo de Objetos da Federação (*Federation Object Model - FOM*).

Durante o desenvolvimento de uma Federação, os seus membros precisam entrar em acordo sobre a natureza das comunicações a serem realizadas. O FOM tem o propósito principal de prover uma especificação para os dados trocados entre Federados num formato comum e padronizado. Ele estabelece o "modelo de contrato da Federação", necessário para alcançar a interoperabilidade entre os Federados. O seu conteúdo inclui a enumeração de todas as classes de objetos e de interações pertinentes à Federação e a especificação, respectivamente, dos atributos e parâmetros que caracterizam essas classes.

Por outro lado, na formação da Federação tem-se que determinar a construção de

Federados individuais. Um SOM especifica os tipos de informações que um Federado pode prover e receber de outros Federados. Usa-se o formato OMT para descrever tanto os FOMs quanto os SOMs. Então, SOMs também caracterizam-se por objetos e seus atributos, e interações e seus parâmetros. Em muitos casos, essas características comuns até permitem que os SOMs sejam integrados como componentes de um FOM, facilitando a sua construção.

A qualquer momento durante a execução da Federação, no máximo um Federado deve responsabilizar-se pela manutenção de um ou mais atributos da instância de uma classe de objeto. Este Federado provê novos valores do(s) atributo(s) invocando os serviços da RTI. Um Federado precisa se inscrever nos atributos para receber as atualizações de seus valores. A cada atualização a RTI invoca o método correspondente na implementação da interface do Federado. O Federado provendo novos valores "atualiza" atributo(s), enquanto que o Federado recebendo esses novos valores "reflete", internamente, o novo estado da instância.

As instâncias de uma classe de objeto persistem durante a execução da Federação, mas as instâncias de uma classe de interação são apenas criadas para o processo de compartilhamento da informação, sendo apagadas automaticamente após a transferência das mensagens. Pode-se considerar que classes de objeto descrevem "coisas", enquanto classes de interação descrevem "eventos" (TOLK, 2001).

A definição de quais objetos ou interações são compartilhados numa Federação depende dos Federados envolvidos e do seu propósito. Numa Federação de lançamento de um foguete, por exemplo, um Federado representando o veículo espacial responsabiliza-se por atualizar os dados referentes à sua trajetória em função do tempo, sendo esses atributos de um objeto compartilhado na Federação. Além disso, pode-se utilizar interações

para divulgar a ocorrência de eventos como: mudanças de fase ou de modelo, apogeu ou impacto.

2.1.1.4 Regras

As Regras, descritas no documento *IEEE STD 1516*, estabelecem princípios e convenções para se alcançar as interações apropriadas entre os Federados, durante a execução de uma Federação. Estas regras compreendem 05 regras de Federação e 05 regras de Federados.

- Regras de Federação:

    1. Federações devem ter um FOM documentado, de acordo com o OMT;
    2. Numa Federação, toda representação de instância de objeto associado à simulação deve estar nos federados e não na RTI;
    3. Durante a execução de uma Federação, toda troca de dados entre os Federados membros deve ocorrer via a RTI;
    4. Durante uma execução de Federação, os Federados membros devem interagir com a RTI, de acordo com a especificação de sua interface; e
    5. Durante uma execução de Federação, um atributo de uma instância pode pertencer a, no máximo, um Federado, em dado momento.

- Regras de Federados:

    1. Federados devem ter um SOM documentado, de acordo com o OMT;
    2. Federados devem ser capazes de atualizar e/ou refletir qualquer atributos de uma instância e enviar e/ou receber interações, como especificado em seus

SOMs;

3. Federados devem ser capazes de transferir e/ou aceitar posse de atributos de uma instância, dinamicamente, durante uma execução de federação, como especificado em seus SOMs;
4. Federados devem ser capazes de variar as condições sob as quais eles fornecem atualizações de atributos de instância, como especificado em seus SOMs; e
5. Federados devem ser capazes de gerenciar o seu tempo local, de modo que permitam coordenar troca de dados com outros membros da Federação.

Para facilitar a interoperabilidade e a reusabilidade, o padrão HLA faz a separação lógica entre as funcionalidades da simulação provida pelos seus membros e os serviços básicos necessários para troca de dados, comunicação e sincronização fornecidos pela RTI.

Por definir uma arquitetura reusável para qualquer tipo de Federação, a adoção do padrão HLA torna-se aplicável para a criação de ambientes de simulação como os requeridos no domínio militar. Para analisar a sua adoção na construção de simuladores para missões de lançamento será desenvolvida, como Estudo de Caso, uma federação simples envolvendo trajetórias de foguetes.

2.2 *Frameworks* Orientados a Objetos

A Programação Orientada a Objetos (POO) fornece ao desenvolvedor de *software* a vantagem de permitir um aumento em sua produtividade, pois este pode facilmente reutilizar código escrito por ele ou por outro programador.

Na POO, as classes podem se relacionar de três maneiras: uma classe filha pode herdar

todos os dados e comportamentos da classe pai; uma classe pode fazer uso de outra classe; e uma classe pode se associar a outra para compor uma terceira mais geral.

As classes denominadas concretas possuem todos os seus métodos implementados e permitem a criação de instâncias. Por outro lado, uma classe abstrata possui um ou mais métodos abstratos. Como não são implementados, eles são especificados apenas para definir o comportamento das classes filhas concretas.

Uma classe abstrata representa entidades e conceitos abstratos, não possuindo instâncias. Ela define um gabarito *(template)* para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade), que é compartilhada por um grupo de classes derivadas (filhas). Cada uma das classes derivadas completa a funcionalidade da classe abstrata implementando o comportamento abstrato de modo específico.

Os *frameworks* representam um progresso do princípio de programação orientada a objetos, pois definem uma solução esqueleto para uma classe de problemas similares. Sua estrutura consiste num conjunto de classes que cooperam entre si e juntas incorporam uma solução reusável para um problema. Eles são muito utilizados, por exemplo, na criação de interfaces gráficas ao usuário *(Graphical User Interfaces - GUIs)*, em que um *framework GUI* provê um grande conjunto de classes abstratas para a construção de itens como: botões, menus, caixas de texto, dentre outros.

Assim como classes abstratas funcionam como um molde para as suas subclasses, um projeto constituído por classes abstratas funciona como um molde para aplicações, consistindo num *framework* orientado a objetos (CCUEC/UNICAMP, 1999). Define-se um *framework* também como:

- Uma infra-estrutura de classes que provêem o comportamento para implementar aplicações dentro de um domínio através dos mecanismos de especialização e composição de objetos, típicos das linguagens orientadas a objetos (CCUEC/UNICAMP, 1999);

- Um conjunto de classes que incorporam um projeto abstrato para soluções a uma família de problemas relacionados (JOHNSON; FOOTE, 1988);

- Um conjunto de classes cooperantes que constroem um projeto reutilizável para uma específica classe de *software* (DEUTSCH, 1989); e

- Um componente de *software* provendo larga escala de reuso de análise, de projeto e de código (LANDIN; NIKLASSON, 1995).

Esta última definição destaca-se pois apresenta as principais vantagens da construção de *frameworks*. O reuso de análise ocorre porque um *framework* descreve os objetos de importância, os relacionamentos entre os objetos e como grandes problemas são quebrados em problemas menores. O reuso do projeto ocorre quando o projeto do *framework* contém algoritmos abstratos e define as interfaces e os obstáculos que uma implementação precisa satisfazer (BUDD, 2002).

Numa aplicação tradicional, o código definido pelo programador controla todo o fluxo de execução do programa, ocasionalmente invocando métodos de bibliotecas para alguma função específica. Por outro lado, em uma aplicação que utiliza *framework*, o fluxo de execução do programa é controlado pelo *framework*. Por esta razão, as classes derivadas do *framework*, possuem o mesmo fluxo de execução (BUDD, 2002).

As aplicações específicas são construídas especializando-se as classes do *framework* para fornecer a implementação de alguns métodos, embora geralmente a maior parte da

funcionalidade da aplicação seja herdada (CCUEC, 1998). O programador apenas altera os métodos do *framework*, ele não consegue alterar a estrutura de controle da aplicação, que é codificada pelas superclasses do *framework*. Estas superclasses invocam os métodos que devem ser implementados para cada aplicação em particular (BUDD, 2002).

A utilização de um *framework* pode ser considerada como produtiva, já que pode ser usado como a base para o desenvolvimento de várias aplicações num domínio de aplicação. Isto difere do modo normal de desenvolvimento de uma aplicação orientada a objetos (MATTSSON, 1996). Esta diferença no processo de desenvolvimento é mostrada na Figura 2.4.

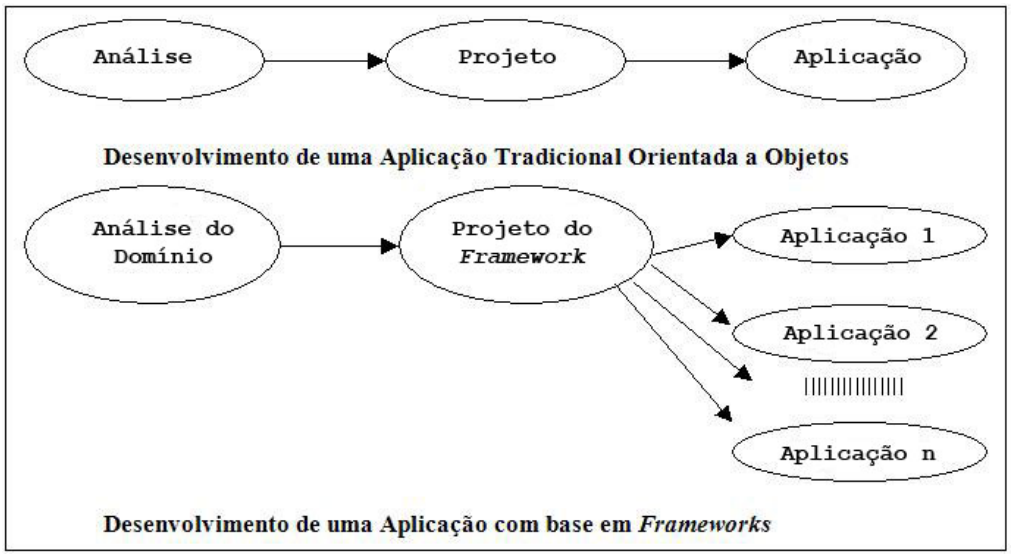

FIGURA 2.4 – Comparação de Desenvolvimento Tradicional e Baseado em *Framework*. (Adaptada de MATTSSON, 1996)

A reusabilidade alcançada no desenvolvimento de *software*, com base em *frameworks*, serviu como motivação para a realização deste trabalho. No *Framework* a ser descrito no Capítulo 4, definiu-se o fluxo de controle para execução das simulações de trajetórias dos foguetes. Cada simulador construído para um propósito específico irá reusar a estrutura criada, implementando o seu comportamento específico.

2.2.1 Metodologia de Desenvolvimento

Segundo Bosch *et al.* (1998), o desenvolvimento de um *framework* é diferente do desenvolvimento de uma aplicação padrão. A diferença chave está no fato do *framework* ter que cobrir todos os conceitos relevantes num domínio, enquanto uma aplicação somente está concentrada com os requisitos da aplicação em particular.

Poucas metodologias voltadas para o desenvolvimento de *frameworks* são conhecidas. Dentre elas, destacam-se a Metodologia de Projeto Dirigido por Exemplo *(Example-Driven Design)* e a Metodologia de Projeto Dirigido por Pontos de Flexibilização *(Hot Spot Driven Design)* (SILVA; PRICE, 1997).

A Metodologia de Projeto Dirigido por Exemplo caracteriza-se pela ênfase à análise do maior número possível de aplicações já desenvolvidas no domínio tratado, para a obtenção de conhecimento do domínio. Esta metodologia estabelece as seguintes etapas para a produção de um *framework*:

1. Análise do domínio (as aplicações existentes são a principal fonte de informações);
2. Definição de uma hierarquia de classes que possa ser especializada para abranger as aplicações de exemplo (um *framework*); e
3. Teste do *framework* através do seu uso no desenvolvimento dos exemplos (implementar, testar e avaliar cada aplicação usada na primeira etapa, utilizando para isto, o *framework* desenvolvido).

Já a Metodologia de Projeto Dirigido por Pontos de Flexibilização enfatiza a busca de *hot spots*, i.e., as partes da estrutura de classes do *framework* a serem mantidas flexíveis, para possibilitar sua adaptação a diferentes aplicações do domínio. Esta metodologia

estabelece basicamente a mesma seqüência de etapas da anterior, porém, sua ênfase está na identificação das partes da estrutura que variam em diferentes aplicações, e a seleção de soluções de projeto adequadas a estes casos.

Para o desenvolvimento do *Framework* apresentado no Capítulo 4, adotou-se a Metodologia de Projeto Dirigido por Pontos de Flexibilização, pois o enfoque esteve na criação da estrutura comum de diferentes simuladores de trajetórias de foguetes, identificando as partes a serem mantidas flexíveis para implementação específica num simulador em particular.

2.2.2 Processo de Desenvolvimento

As duas metodologias, citadas na Seção 2.2.1, descrevem como produzir *frameworks* sem estabelecer um processo detalhado de construção de modelos que dirija o desenvolvimento. Elas estabelecem o processo de desenvolvimento em linhas gerais, sem se ater à definição de técnicas de modelagem.

O processo de desenvolvimento de um *framework* depende da experiência da organização no domínio do problema endereçado por ele, sendo que organizações mais experientes podem adotar processos mais avançados. Johnson (1993) e Wilson e Wilson (1993), citados por Mattsson (1996), propuseram três processos para desenvolvimento de *frameworks*. O primeiro, denominado Processo de Desenvolvimento Baseado em Experiências de Aplicação, define as seguintes atividades para o desenvolvimento de um *framework*:

1. O desenvolvimento de duas aplicações de teste num problema;

2. A partir da primeira iteração, identificam-se as características comuns às duas aplicações, extraíndo-as para um *framework*;

3. Com o intuito de verificar se estas características extraídas estão corretas, as aplicações passam novamente pelo processo de desenvolvimento, só que agora com base no *framework*;

4. Se as características não forem corretas, o *framework* deverá ser reescrito e as experiências obtidas usadas para desenvolvimento da próxima versão; e

5. Com base nesta nova versão, novas aplicações podem ser desenvolvidas.

Outro processo citado refere-se ao Processo de Desenvolvimento Baseado em Análises do Domínio, que considera as seguintes atividades para a criação do *framework*:

1. A análise de domínio do problema para identificar e entender as abstrações conhecidas. Esta atividade requer análises das aplicações existentes, uma tarefa difícil, e que somente torna-se possível se a organização tiver desenvolvido aplicações no referido domínio;

2. Uma vez que as abstrações foram identificadas, desenvolve-se o *framework* juntamente com uma aplicação de teste, modificando-o quando necessário; e

3. A próxima atividade refere-se ao desenvolvimento de uma segunda aplicação com base no *framework*. Novamente, deve-se modificar o *framework*, se necessário, e revisar a primeira aplicação, para que assim, ela trabalhe com as mudanças introduzidas. Dessa forma, o *framework* se desenvolve à medida que novas aplicações são criadas.

Por fim, a Figura 2.5 resume os elementos comuns do último processo abordado, denominado Processo Geral de Desenvolvimento de *Framework*, cujas atividades foram descritas a seguir:

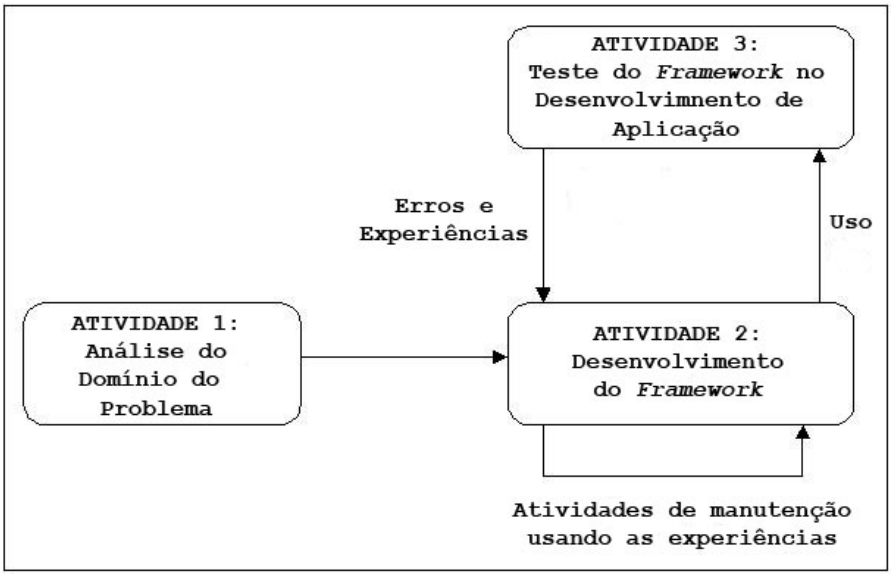

FIGURA 2.5 – O Processo Geral de Desenvolvimento do *Framework*. (Adaptada de MATTSSON, 1996)

1. **Análise de Domínio do Problema** - é executada sistematicamente através do desenvolvimento de uma ou várias aplicações, onde identificam-se as abstrações chaves;

2. **Desenvolvimento do *Framework*** - consideram-se as abstrações chaves e os problemas encontrados para a criação de uma versão nova do modelo abstrato; e

3. **Teste do *Framework* no Desenvolvimento de Aplicação** - para verificar sua reusabilidade, desenvolve-se uma ou mais aplicações com base no *framework*.

Após repetir as Atividades 2 e 3 um determinado número de vezes, o *framework* atinge um nível de maturidade aceitável e pode ser liberado para reuso multi-usuário na organização.

Para a construção do *Framework* proposto no Capítulo 4, adotou-se o Processo Geral de Desenvolvimento de *Frameworks*, por enfatizar o desenvolvimento incremental do *Framework* até se tornar estável. Como o *Framework* desenvolvido não considera foguetes dotados de mecanismos de controle, e.g, (IAE, 1993), o seu processo de desenvolvimento

poderá ser estendido, em pesquisas futuras, a fim de buscar a sua evolução, englobando simulações de trajetórias para tais tipos de veículos espaciais.

2.3 Trabalhos Relevantes

Esta Seção aborda dois trabalhos relacionados a este projeto de pesquisa, que se destacaram entre os trabalhos semelhantes pesquisados. Ressalta-se a dificuldade em se obter informações sobre projetos similares, uma vez que simulação militar constitui uma área que envolve pouca divulgação.

Não foram encontradas ainda informações sobre a utilização de *frameworks* OO e *HLA* no desenvolvimento de projetos brasileiros de foguetes. Uma das contribuições deste trabalho consiste na aplicação e avaliação destas tecnologias num Estudo de Caso direcionado ao setor aerospacial.

No primeiro trabalho, intitulado "Modelos Genéricos de Simulação de Veículos Lançadores Reusáveis", Steele *et al.* (2002) enfatiza que o tempo dispendido no processo de análise de sistemas por meio de simulações aumenta consideravelmente quando modelos de múltiplos sistemas são comparados. Conduzir um estudo de simulação torna-se mais complicado em duas condições: primeiro, quando detalhes do sistema ainda são desconhecidos por este ainda estar no princípio da elaboração, e segundo, quando se compara dois ou mais sistemas competitivos.

Então, o problema consiste em como desenvolver um modelo válido e confiável de um sistema não existente ou em elaboração, num modo conveniente, e compará-lo com modelos similares de sistemas competitivos. Para isso, o autor sugere o desenvolvimento de um ambiente genérico de simulação do domínio em questão.

Ele aponta que, apesar de um modelo mais abstrato de sistema ser mais difícil de construir e validar, torna-se uma vantagem a reusabilidade do modelo resultante nas implementações de vários sistemas especifícios no domínio em questão. Além disso, como muitos passos envolvidos na construção e simulação dos múltiplos sistemas são compartilhados por causa dos objetivos comuns, reduz-se o tempo necessário no estudo e elaboração do modelo genérico.

De acordo com Mackulak e Cochran (1990), citado por Steele *et al.* (2002), 45% do esforço de um projeto de simulação está na construção, formulação e tradução do modelo. Então, implementar um modelo genérico, aplicável a muitos sistemas, pode reduzir uma porção significativa do tempo na elaboração de simuladores.

Esta foi a principal motivação para a criação do *Framework* descrito no Capítulo 4, pois pretende-se reduzir o tempo para a construção e manutenção de simuladores de trajetórias. Para pesquisadores interessados na simulação do lançamento de foguetes, um modelo abstrato pode ajudá-los na criação de simuladores para avaliar e escolher modelos matemáticos mais precisos para, por exemplo, representar a atmosfera terrestre ou a propulsão dos motores do foguete.

No segundo trabalho, intitulado "Uma Avaliação da High Level Architecture (HLA) como um *Framework* para a Modelagem e Simulação na NASA", Reid (2000) avalia a aplicação do padrão HLA como uma tecnologia para as simulações de missões espaciais da *National Aeronautics and Space Administration (NASA)*. O propósito do seu estudo foi determinar se o padrão HLA corresponde à especificação de uma tecnologia com potencial para o uso nas simulações da NASA, envolvendo numerosas aplicações separadas e simulando aspectos complexos e variados de uma missão espacial completa. Exemplos destes aspectos são naves espaciais, instrumentos embarcados, telemetria, estações de rastreio,

sistemas de solo, dinâmicas de vôo, e no caso de missões científicas, objetos naturais e fenômenos estudados.

Para aprender sobre o padrão HLA e verificar sua aplicabilidade ao domínio dos problemas de interesse da NASA, Reid (2000) construiu um protótipo de sistema de simulação baseado nessa tecnologia. Este protótipo consistiu de diversos simuladores baseados no padrão HLA interagindo como Federados numa Federação. Seu enfoque esteve na avaliação do HLA como uma tecnologia dirigida a missões espaciais, não sendo objetivo a criação de um simulador para uso numa missão real.

Este protótipo consistiu de quatro pequenos e especializados simuladores: um controlador de nave espacial, um calculador de órbita, um simulador da terra, e uma estação de rastreio. Estes quatro Federados correspondem a aplicações completamente separadas, que executam concorrentemente, mas não de forma independente, em processos individuais e opcionalmente em computadores isolados.

Reid (2000) concluiu que o padrão HLA mostrou-se adequado para os tipos de simulações para as quais foi projetado, sendo uma tecnologia sólida e bem elaborada. Ele sugere que este padrão seja considerado pela NASA e pela ampla comunidade de modelagem e simulação no desenvolvimento de simuladores, pois entende que o HLA provê um padrão que reduzirá, de forma otimista, o custo e o tempo de desenvolvimento de sistemas de simulação e aumentará suas capacidades por facilitar o reuso e a interoperabilidade de simuladores integrantes. Por unir-se a um padrão comum, os simuladores não precisam ser limitados à aplicação para a qual eles foram especificamente projetados, estando disponíveis para o uso em outros sistemas de simulação.

Segundo Reid (2000), a implementação do padrão HLA pode ser uma tecnologia central em certos tipos de simulações de missões espaciais, incluindo as que envolvem múltiplas

naves espaciais voando em formação. Este padrão pode também servir como uma arquitetura para aplicações científicas que modelam sistemas naturais. Ele não aumentará a confiabilidade de simuladores, mas seu projeto modular e distribuído tornará fácil a construção de sistemas de simulação de alta confiabilidade. Ele ressalta que o padrão HLA torna-se especialmente bem adaptado como base para simulações semelhantes a jogos, em que um pequeno "universo" está sendo simulado, com múltiplos atores chegando, interagindo e saindo. Também tem potencial como um *framework* para integrar numerosos simuladores separados num largo sistema de simulação, pois a natureza distribuída inerente do padrão pode permitir essencialmente escalabilidade ilimitada.

Além de recomendar o padrão HLA para a criação de sistemas de simulação distribuídos, Reid (2000) também visualiza um ambiente de simulação baseado neste padrão para suportar uma missão espacial desde as fases iniciais do seu ciclo de vida. Os responsáveis pelo planejamento das missões e os cientistas começariam seus estudos com um ou dois simuladores interagindo neste ambiente.

À medida que houvesse progresso no desenvolvimento de uma missão e uma melhor definição dos requisitos, o ambiente cresceria, com mais aplicações sendo adicionadas para o sistema e outros simuladores sendo refinados ou substituídos. Neste sentido, este ambiente se tornaria uma ferramenta de modelagem sofisticada para examinar cenários do tipo "o que aconteceria se".

Quando uma missão atingisse estágios avançados de desenvolvimento, este ambiente de simulação seria expandido para compor um sistema de teste e integração para gerar telemetria de alta confiabilidade, alimentando os sistemas atuais de solo ou até mesmo uma nave espacial real. Já neste sentido, o ambiente se tornaria uma plataforma para o treinamento de operações das equipes de funcionários, provendo uma plataforma para a

avaliação da segurança de respostas para as anomalias e outras divergências em relação ao plano original da missão

Reid (2000) destaca ainda que o ambiente iria se tornar grande e complexo e necessitaria de poder de computação maior do que capacidade de qualquer computador individual, mas como o HLA baseia-se em componentes, essa complexidade seria gerenciável. Além disso, como suporta completamente computação distribuída, uma simples adição de processadores poderia ser realizada, sendo alguns deles em locais remotos.

Apesar de todas as vantagens discutidas, o autor considera que o padrão HLA, provavelmente, não define uma arquitetura apropriada para simulações que apenas produzem dados para algum sistema único e externo ou para simulações que não envolvem uma coleção de objetos interagindo. Nestes casos, a arquitetura se torna apenas uma camada extra e não necessária, provendo pouco ou nenhum valor adicional. Isto desprezando o fato de que tais simuladores poderiam, no futuro, virar componentes de grandes simulações baseadas no HLA.

A experiência compartilhada no trabalho de Reid (2000) serviu de motivação para o estudo e a análise do padrão HLA. Após todas as vantagens apresentadas, torna-se plausível recomendar o padrão HLA para a construção de simuladores direcionados para o setor espacial, visando facilitar a integração dos mesmos na construção de um ambiente integrado para a simulação de missões espaciais completas.

Com a adoção do padrão HLA, um simulador de trajetórias torna-se um componente de Federações, produzindo dados para outros sistemas como: plataformas de treinamento dos astronautas para vôos tripulados; ambientes para treinamento de equipes de solo, principalmente relacionadas à segurança de vôo; ambientes para testes dos sistemas embarcados em veículos espaciais, dos equipamentos de rastreio ou de telemetria, ou outros

sistemas de solo; dentre outras aplicações.

A arquitetura para simulação distribuída definida pelo padrão HLA, juntamente com o *Framework* apresentado no Capítulo 4, podem auxiliar o trabalho dos pesquisadores envolvidos no planejamento e na preparação de missões espaciais. Assim, além de avaliar a reusabilidade proporcionada pelo referido *Framework*, também pretende-se medir, em termos quantitativos, o esforço necessário para o desenvolvimento de Federados e de uma Federação envolvendo lançamento de foguetes de sondagem.

3 Simulação de Trajetórias de Foguetes de Sondagem

3.1 Uso de Simulação de Trajetórias

Os foguetes de sondagem constituem um meio consideravelmente útil e de baixo custo para as pesquisas da atmosfera e ionosfera, pesquisas de novos materiais e processos em ambiente de microgravidade (IAE, 2007; AEB, 2007). A Figura 3.1 ilustra os principais foguetes de sondagem brasileiros desenvolvidos em parcerias entre o Comando-Geral de Tecnologia Aeroespacial (CTA) e indústrias do setor aeroespacial.

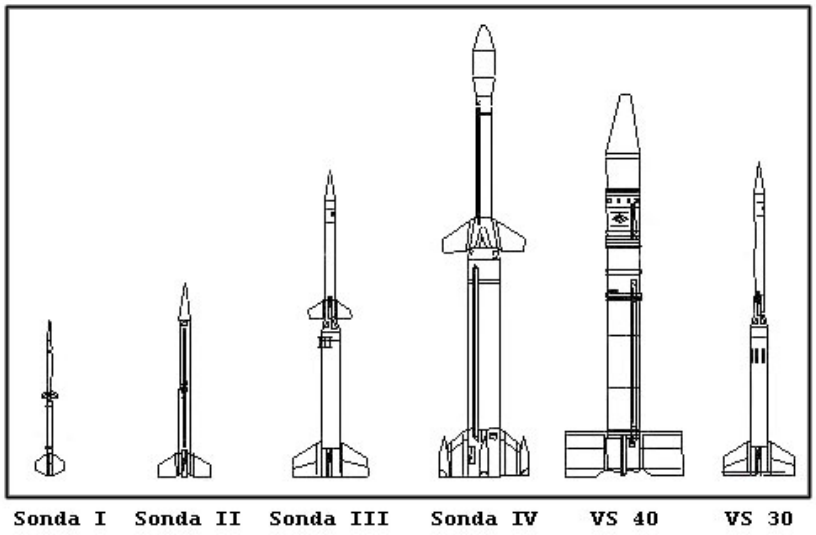

FIGURA 3.1 – Foguetes de Sondagem Brasileiros. (Adaptada de RIBEIRO, 1998)

Os foguetes de sondagem brasileiros atuais possuem até dois estágios, sendo capazes de transportar cargas úteis científicas e tecnológicas, de até 500 kg, para experimentos na faixa de 70 a 1000 km de altitude, dependendo do modelo.

Uma campanha de lançamento desses foguete caracteriza-se pelo conjunto de atividades planejadas para: o treinamento de meios operacionais, o lançamento de veículos espaciais e o rastreio de suas cargas úteis num centro de lançamento (CLA, 2007).

Na preparação das campanhas, realiza-se atividades de integração, montagem e testes, envolvendo o veículo e a sua carga útil, além dos meios de solo auxiliares e demais interfaces com os centros de lançamento. Após a fase de verificação e testes, realizam-se treinamentos para o lançamento simulado do foguete, incluindo contagens regressivas simuladas.

Durante a preparação, deve-se definir o caminho que o foguete irá percorrer para cumprir a sua missão. A trajetória nominal corresponde a previsão do caminho que o veículo percorre no espaço aéreo num período de tempo, sendo obtida através de simulação (LOUIS, 2006).

Uma Equipe de Segurança de Vôo (SVO) tem como responsabilidade controlar os riscos inerentes à trajetória que o veículo percorre. O trabalho da SVO começa dias antes do lançamento, avaliando as condições meteorológicas e a área de dispersão de impacto, dentre outras atividades. Geralmente, simula-se trajetórias anômalas ou situações adversas, para o treinamento dos procedimentos executados pela equipe de solo. Estas simulações visam minimizar os riscos envolvidos, caso ocorra uma situação semelhante à simulada no vôo real.

A SVO deve conhecer a trajetória nominal do veículo, pois se num lançamento real ocorrer alguma falha que resulte numa trajetória diferente da nominal, ela pode acionar

remotamente um dispositivo de conclusão de vôo. Normalmente, realiza-se várias simulações para a determinação da trajetória nominal de uma campanha de lançamento, pois além de garantir o cumprimento com sucesso da missão, a trajetória deve seguir regras de segurança.

Para realizar uma simulação da trajetória de um foguete necessita-se de diversas informações como: características do veículo (por exemplo: massa, propulsão, geometria, aerodinâmica); posição do dispositivo fixado ao solo no qual lança-se o veículo, denominado lançador; dados sobre a seqüência de eventos de vôo, como tempo de ignição e separação dos motores; dentre outras. Quanto aos resultados de uma simulação, destacam-se a altitude máxima alcançada pelo veículo, denominada apogeu, e as coordenadas do local onde o veículo se chocou com a superfície terrestre, chamado ponto de impacto, e.g., (LOUIS, 2006).

Num lançamento real, pequenas variações indesejáveis em parâmetros do vôo, como ventos fortes, acima do normal, ou pesagem errada do veículo, provocam o seu desvio do percurso previsto e, conseqüentemente, do seu ponto de impacto simulado. Por causa disto, o processo de definição da trajetória nominal ainda prevê o lançamento em condições adversas, com a necessidade do cálculo de trajetórias alternativas, que nunca devem violar o cumprimento da missão (LOUIS, 2006).

Para criar estas trajetórias alternativas, realiza-se várias simulações prevendo determinados ajustes em alguns parâmetros. Por exemplo, pode-se alterar o azimute e a elevação do lançador, de modo que o veículo possa cumprir o caminho correto apesar dos efeitos dos ventos. Este ajuste deve obedecer aos limites impostos pelo centro de lançamento.

A trajetória nominal e as trajetórias alternativas possuem pontos de impacto próximos. Assim, a SVO pode interditar a região composta pelos vários pontos de impacto simulados,

como provável local para a queda de partes ou do veículo inteiro, numa tentativa de garantir a segurança das pessoas e dos meios envolvidos no lançamento real.

Uma trajetória alternativa pode tornar-se a trajetória nominal do vôo se minutos antes do lançamento, por exemplo, as condições climáticas na região do lançador não estiverem como previstas. A troca da trajetória nominal minimiza riscos futuros, pois uma trajetória mais compatível com as condições do lançamento estará mais próxima da trajetória do vôo real.

Além disso, durante uma operação de lançamento utiliza-se alguns meios, como radares e telemetria, para rastrear e monitorar o foguete durante o seu vôo. Os dados gerados nestas atividades possibilitam uma comparação posterior com os dados calculados, possibilitando uma análise completa pós-vôo.

Existem casos em que não ocorre o rastreio do veículo pelos meios utilizados por diversas causas. Essa falha pode ocorrer por alguns instantes ou mesmo durante todo o restante do vôo. Como tentativa de contornar esse problema de registro incompleto dos dados, pode-se obter dados simulados para esses instantes. Este cálculo é conhecido como reconstituição da trajetória, baseada na posição e velocidade do veículo num determinado instante de vôo (LOUIS, 2006).

Dessa forma, um simulador de trajetórias de foguetes torna-se uma ferramenta importante para a preparação de campanhas de lançamento. Ele é usado para definir as trajetórias nominal e alternativas, e também a região para interdição. Além disso, um simulador possibilita o treinamento das equipes envolvidas numa operação de lançamento e o teste integrado dos sistemas do foguete e dos meios de solo.

3.2 Programa Atualmente Utilizado no IAE

O Instituto de Aeronáutica e Espaço (IAE) utiliza, desde a década de 70, como ferramenta de suporte para a definição da trajetória nominal, o programa *ROSI - Rocket Simulation* (KRAMER; CRAUBNER; ZIEGLTRUM, 1976; IAE, 2005), para analisar o movimento de um corpo rígido, como um veículo de sondagem multiestágios, no espaço tridimensional, considerando sua translação e rotação.

O programa apresenta os resultados obtidos com a simulação da trajetória num relatório com os principais parâmetros do veículo relativos ao tempo do vôo simulado. Para promover a validação do ROSI, realizou-se um estudo comparativo abrangendo os resultados deste programa com os dados de vôos reais, sendo o resultado desta análise considerado satisfatório (PEREIRA; YAMANAKA, 2006), citado por (LOUIS, 2006).

Este programa é constantemente atualizado e validado, principalmente em sua forma de apresentação e integração tais como outros aplicativos, como analisadores gráficos (LOUIS, 2006).

O ROSI foi desenvolvido, numa forma estruturada, em linguagem FORTRAN, possuindo mais de 7.000 linhas de código. Para armazenar os dados de entrada, de saída e temporários utilizados na execução de uma simulação, o programa mantém um único vetor com dimensão superior a 1000 posições.

Não é possível classificar o ROSI como um *software*, pois este não foi produzido por um Processo de Engenharia de *Software*. Trata-se de um programa (i.e., código-fonte) obsoleto, que não possui requisitos, modelo conceitual, e etc. A estrutura complexa do programa torna-se um empecilho ao estendê-lo para outros propósitos. As alterações no código devem ser bem planejadas para não comprometer o seu funcionamento.

Louis (2006) também desenvolveu um *software*, denominado jROSI, para o cálculo de trajetória extrapolada, como uma adaptação ou tradução do programa ROSI, para a linguagem Java. Neste desenvolvimento, Louis (2006) manteve a estrutura original do programa em FORTRAN, para preservar o algoritmo já qualificado, fazendo apenas uma otimização para explorar os recursos da nova linguagem. Ele realizou várias simulações usando uma mesma base de dados, e após comparar os resultados de ambos os programas, ele considerou os resultados satisfatórios para seu trabalho de pesquisa.

3.2.1 Modelos de Simulação do ROSI

O programa ROSI adota três modelos para o cálculo do movimento de um veículo espacial, considerando um, três ou seis graus de liberdade.

As equações gerais de movimento consistem no fundamento para análise, processamento ou simulação do movimento de veículos de sondagem. As principais equações para calcular o movimento do veículo no espaço tridimensional, considerando a sua translação e rotação, encontram-se descritas em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Para a integração numérica das equações de movimento, o programa adota o método desenvolvido por Nordsieck (1962).

O equacionamento completo considerado pelo programa ROSI encontra-se documentado com maiores detalhes em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Constatou-se algumas inconsistência nesta documentação, provocadas principalmente devido a erros de digitação.

Alguns parâmetros presentes nas equações de movimento podem sofrer freqüentes descontinuidades, como uma variação abrupta de massa causada pelo fim da queima e li-

beração de um estágio. Para contornar estas descontinuidades pode-se dividir o vôo em algumas fases de modo que estas descontinuidades não ocorram dentro de uma mesma fase de vôo.

A Figura 3.2 apresenta, de forma ilustrativa, o modelo de fases de vôo de um veículo de sondagem, onde também podem ser verificadas as separações de seus estágios. O modelo de fases consiste em etapas pré-definidas a serem cumpridas em ordem cronológica, sendo associadas a ocorrência de eventos na simulação.

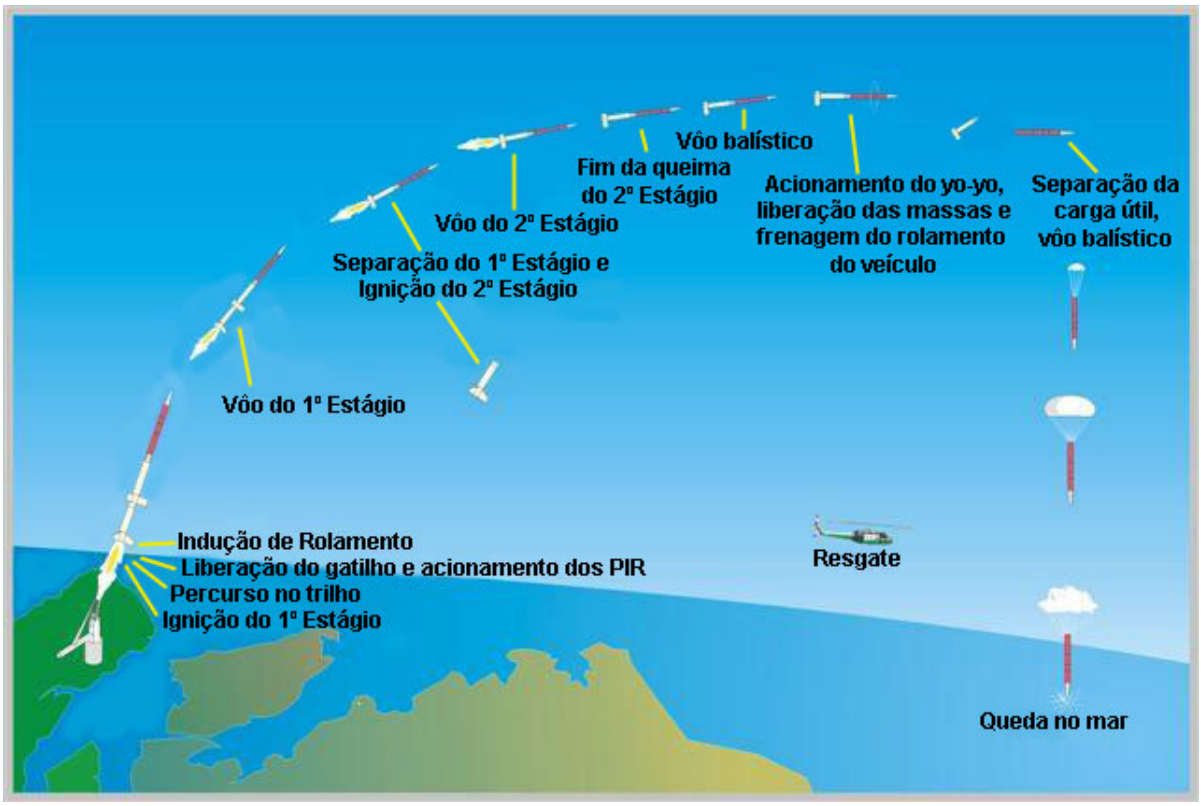

FIGURA 3.2 – Fases de vôo de um veículo de sondagem. (Adaptada de LOUIS, 2006)

Os modelos de simulação do ROSI organizam a trajetória num modelo de fases, sendo cada fase de vôo associada a um modelo matemático para o cálculo do movimento do foguete. O arquivo de entrada define a configuração de cada fase da simulação, através de um conjunto de parâmetros e tabelas.

Para o vôo simulado do veículo preso ao trilho do lançador, define-se uma fase associada

ao modelo matemático de movimento numa dimensão (1D). As fases de vôo dentro da atmosfera relacionam-se ao modelo matemático de movimento em seis dimensões (6D), considerando a translação e rotação do foguete no espaço tridimensional. Já as fases de vôo balístico, associam-se ao modelo de matemático de movimento em três dimensões (3D), considerando apenas a translação do veículo no espaço tridimensional.

Os modelos de simulação 1D e 3D correspondem a simplificações do modelo 6D. O propósito do uso destes três modelos deve-se a redução da quantidade de cálculos realizados durante a execução de uma simulação e, conseqüentemente, do tempo de execução. Isto foi importante na década de 70, com recursos computacionais disponíveis bastantes limitados.

Com base nestes modelos de simulação do ROSI, foi proposto no Capítulo 4 um *framework* para simulação de trajetórias de foguetes, que estabelece uma simulação abstrata através de um modelo orientado a objetos, com o propósito facilitar a compreensão, criação e manutenção de modelos matemáticos e de fases de simuladores envolvendo trajetórias espaciais.

Cabe ressaltar que o foco deste trabalho de pesquisa não se encontra na escolha do melhor modelo matemático para o cálculo de trajetórias de foguetes, mas na criação de um *framework* que possibilite reuso de código para implementar estes modelos na construção de simuladores.

4 Um *Framework* para Simulação de Trajetórias de Foguetes de Sondagem

Este Capítulo aborda o projeto de desenvolvimento de um *Framework* para a Simulação de Trajetórias de Foguetes de Sondagem, denominado Fênix.

Os diagramas apresentados se baseiam na *Unified Modeling Language - UML* 2.0 (GUEDES, 2005), construídos através do Ambiente de Desenvolvimento Integrado (*Integrated Development Environment - IDE*) Borland JBuilder, versões 2005 e 2007.

4.1 Arquitetura do Fênix

O *Framework* Fênix baseia-se na arquitetura de *software* para a simulação de trajetórias de foguetes, concebida pelo autor, e apresentada na Figura 4.1. A seguir, serão descritos os módulos que integram esta arquitetura:

- **Módulo de Interface com o Usuário** - através deste modúlo o usuário fornece os dados de entrada para a simulação de lançamento, que correspondem a parâmetros e tabelas para controle da simulação e definição das entidades do sistema como: veículo, terra, atmosfera, lançador, dentre outras. Além disso, esse módulo também

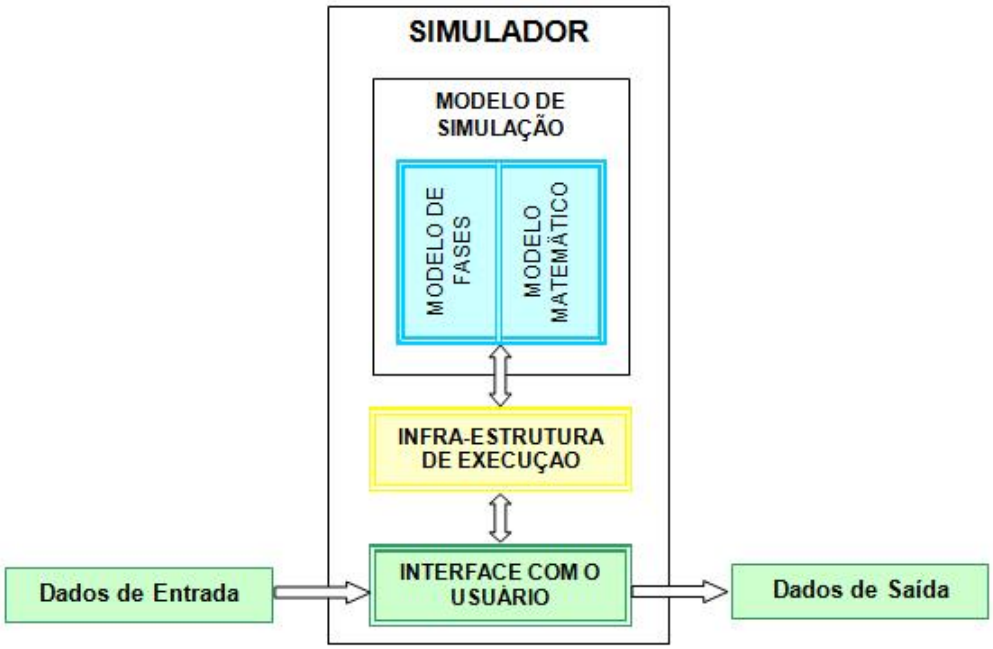

FIGURA 4.1 – Arquitetura de *software* para simulação de trajetórias de foguetes

apresenta ao usuário os resultados obtidos durante a simulação;

- **Módulo do Modelo de Simulação** - este módulo define um modelo para a simulação de trajetórias de um foguete, sendo sub-dividido nos seguintes módulos:
  - **Módulo do Modelo de Fases** - os parâmetros e tabelas de entrada são utilizados na criação de um modelo estruturado por fases, compreendendo desde o lançamento do veículo até o seu impacto na superfície da terra. As mudanças de fases ocorrem devido a ocorrência de eventos como liberação de estágios, apogeu, dentre outros, e
  - **Módulo do Modelo Matemático** - cada fase estabelece um modelo de equações para o calcular o movimento do foguete em função do tempo; e
- **Módulo de Infra-estrutura de Execução** - define o fluxo de controle e de dados da simulação. Este módulo proporciona o avanço de tempo para o progresso da execução da simulação.

Esta arquitetura permite a execução de simulações de trajetórias de foguetes para diferentes propósitos. O Módulo do Modelo de Fases coordena a execução subsequentes de fases, controlando o avanço de tempo para o progresso da simulação. Independente do modelo da simulação adotado, a estrutura e o fluxo de controle para execução das fases pode ser o mesmo em diferentes simuladores. Quanto ao Módulo do Matemático, o conjunto de equações pode variar de um simulador para o outro, mas o fluxo de controle e de dados para a integração das equações diferenciais de movimento também continua o mesmo.

O *Framework* Fênix propõe um Módulo do Modelo de Simulação abstrato, abrangendo os fluxos de controle e de dados comuns nos diversos simuladores de lançamento dos foguetes de sondagem. Ele captura as funcionalidades comuns, proporcionando o reuso de código e servindo como um molde para a criação de simuladores para diferentes propósitos, como mostrado na Figura 4.2.

	***Framework* Fênix**	**Modelo de Simulação Concreto**
Modelo de Fases	Estruturas de dados e rotinas para execução das fases de simulação	Estruturas de dados e rotinas para configuração de cada fase específica
Modelo Matemático	Estruturas de dados e rotinas para integração das equações de movimento do foguete	Estruturas de dados e rotinas que definem as equações de movimento em cada fase específica

FIGURA 4.2 – Reuso de código dos modelos

O *Framework* Fênix é um gabarito para construir modelos para simulação de trajetórias de foguetes de sondagem, pois além de incorporar o comportamento comum de diversos modelos, ele define como deve ser implementado o comportamento especifico de cada um. O Fênix estabelece uma estrutura de classes, quase sempre abstratas, e o re-

lacionamento entre elas. Assim, os modelos específicos precisam apenas implementar o comportamento abstrato dessas classes, especializando-o quando necessário.

4.2 Processo de Desenvolvimento do Fênix

Para o desenvolvimento do Fênix adotou-se a Metodologia de Projeto Dirigido por Pontos de Flexibilização, descrita na Seção 2.2.1, e o Processo Geral de Desenvolvimento do Framework, descrito na Seção 2.2.2, do Capítulo 2. A seguir são descritos os Passos realizados no referido processo de construção do *Framework*:

1. **Análise do Domínio** - ocorreu o estudo do código-fonte do programa ROSI em Fortran, juntamente com o desenvolvimento de um simulador semelhante, mas orientado a objetos. Este passo permitiu compreender os fluxos de controle e de dados envolvidos na simulação de foguetes de sondagem;

2. **Desenvolvimento do *Framework*** - com base nos problemas encontrados e nas classes comuns e pontos de flexibilização identificados, desenvolveu-se uma versão nova do *Framework* Fênix; e

3. **Teste do *Framework* no Desenvolvimento de Aplicação** - para refinar os requisitos e validar o modelo abstrato proposto pelo Fênix, desenvolveu-se, como Estudo de Caso, um protótipo de simulador de lançamento, que engloba a construção de três modelos baseados no Fênix, considerando-se uma, três ou seis dimensões para o movimento do veículo.

   Enquanto o *Framework* Fênix não se tornou estável, repetiu-se continuamente os Passos 2 e 3. A cada ciclo de desenvolvimento, o protótipo foi avaliado por uma

Equipe de Engenheiros da Divisão de Sistemas Espaciais (ASE), do Instituto de Aeronáutica e Espaço (IAE), do Comando Geral de Tecnologia Aeroespacial (CTA). O Capítulo 5 descreve a construção de um protótipo de Simulador de Lançamento de Foguetes de Sondagem.

4.3 Requisitos do Fênix

A especificação de requisitos do *Framework* Fênix foi descrita numa adaptação do Documento de Especificação de Requisitos de Software do *Rational Unified Process - RUP* (JACOBSON; BOOCH; RUMBAUGH, 1999), disponível no Apêndice A.

Neste documento, dividiu-se as funcionalidades desejadas em dois grupos. Quanto ao requisitos desejados para o Módulo do Modelo de Fases, o *Framework* deve propiciar: a divisão da simulação em fases de vôo e o controle de execução da simulação. Quanto aos requisitos do Módulo do Modelo Matemático, ele deve propiciar: o modelo das entidades envolvidas e o cálculo do movimento do foguete considerando diferentes graus de liberdade.

Quanto à qualidade, o *Framework* deve propiciar a criação de modelos utilizando a *Unified Modeling Language - UML* e ferramentas *Computer-Aided Software Engineering - CASE*. Além da aplicação de métodos, técnicas e ferramentas da Engenharia de *Software*, em conformidade com o Processo Geral de Desenvolvimento de *Frameworks*.

Quanto à confiabilidade, o *Framework* Fênix deve propiciar a precisão numérica adequada dos dados durante a simulação. Quanto à implementação, ele deve propiciar a utilização da linguagem orientada a objetos Java e de *Integrated Development Environments - IDEs*. Quanto à documentação, o Fênix deve propiciar a produção de artefatos durante o desenvolvimento do protótipo.

Quanto às limitações do modelo, o *Framework* Fênix pode ser reutilizado apenas para o desenvolvimento de simuladores na linguagem Java, e tem permissões irrestritas para reuso.

4.4 Casos de Uso do Fênix

Mapeou-se os requisitos funcionais do *Framework* Fênix conforme o diagrama de casos de uso mostrado na Figura 4.3, cuja documentação se encontra disponível no Apêndice B. Nesta Figura, identificou-se os casos de uso do Fênix através da sigla FEN-CDU, seguida de sua numeração.

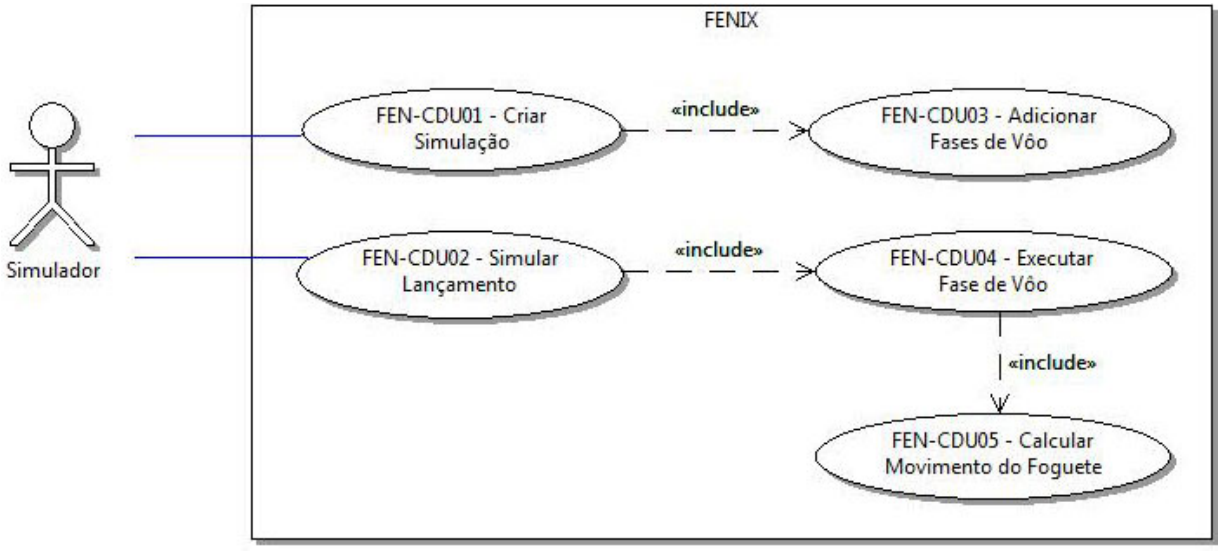

FIGURA 4.3 – Diagrama de Caso de Uso do *Framework* Fênix

O diagrama aborda a interação entre um simulador e o Fênix, mostrando as funcionalidades que o *Framework* deve disponibilizar para a aplicação que o reutiliza. Um simulador pode utilizar o *Framework* para criar uma simulação, com todas as suas fases e os modelos das entidades envolvidas, e também para simular o lançamento calculando o movimento do foguete em função do tempo de simulação.

4.5 Estrutura de Pacotes de *Software* do Fênix

Com base na arquitetura idealizada na Seção 4.1, elaborou-se a estrutura de pacotes de *software* ilustrada na Figura 4.4. Organizou-se o *Framework* Fênix numa estrutura de componentes, de acordo com os princípios definidos pela orientação a objetos.

FIGURA 4.4 – Estrutura de Pacotes de *Software* para o *Framework* Fênix

Dividiu-se o *Framework* em três pacotes: *matematico*, *fases* e *util*. O pacote *fases* estabelece o modelo da simulação numa forma estruturada por fases abstratas, e define o fluxo de controle para a execução seqüencial dessas fases. O pacote *matematico* determina o modelo abstrato das entidades envolvidas no cálculo da trajetória de um foguete de sondagem em função do tempo de vôo. Por fim, o pacote *util* abrange classes de suporte necessárias aos modelos matemático e da simulação.

4.6 Classes do Fênix

Foi concebido o modelo de classes do *Framework* Fênix, para o atendimento dos casos de usos propostos. A Figura 4.5 apresenta a organização das classes dos pacotes

matematico e simulacao, sendo maiores detalhes abordados no Apêndice C.

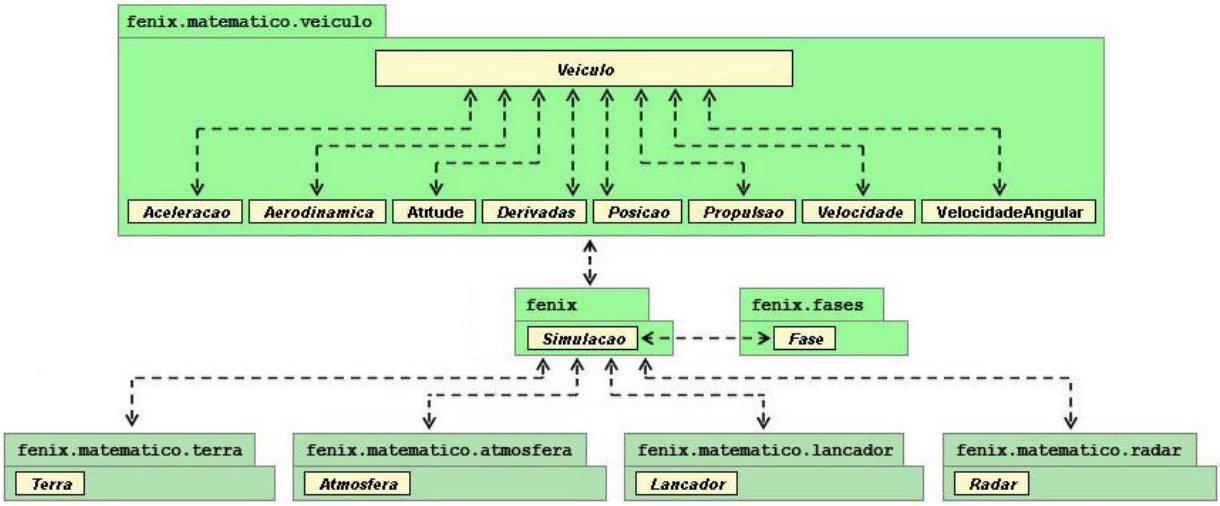

FIGURA 4.5 – Diagrama de classe do *framework* por módulos

Os pacotes *veiculo*, *terra*, *atmosfera*, *lancador* e *radar* possuem classes abstratas que representam as entidades envolvidas na simulação, correspondendo a, respectivamente, Veículo Espacial, Planeta Terra, Atmosfera Terrestre, Lançador e Radar.

Como a orientação a objetos permite que um componente seja substituído por outro equivalente, implementando o mesmo comportamento especificado, pode-se desenvolver modelos mais complexos destas entidades, sem a necessidade de mudanças em outros pacotes.

Por exemplo, para representar a entidade Atmosfera criou-se uma classe abstrata de mesmo nome, que define métodos abstratos para o cálculo das forças e dos momentos aerodinâmicos. Estes métodos serão implementados de acordo com o modelo da Atmosfera mais adequado à simulação em questão. Quando um modelo mais preciso for descoberto, basta reescrever estes métodos conforme definidos para o novo modelo, não sendo preciso alterar código em outras classes do *Framework*.

A classe *Simulacao* especifica métodos abstratos para acessar estas entidades envolvidas na execução de uma simulação de lançamento. As demais classes do *Framework*

Fênix manipulam as entidades através destes métodos, apesar da definição do modelo concreto de cada uma ocorrer posteriomente, seja durante a criação de um simulador, ou até mesmo, em tempo de execução. Esta abordagem proporciona flexibilidade, uma vez que adia a escolha do modelo matemático destas entidades, assumindo apenas a existência do modelo abstrato.

A classe *Simulacao* não possui objetos que representam estas entidades, os seus métodos abstratos possibilitam que o modelo matemático destas entidades seja implementado posteriormente. Ao estender o *Fênix*, um simulador tem por responsabilidade modelar estes objetos e garantir que o restante do *Framework* possa acessá-los através deste métodos.

Por exemplo, a classe *Simulacao* especifica um método abstrato, chamado *Veiculo()*, para que as demais classes do Fênix possam acessar o comportamento associado ao objeto que representa a nave espacial. Durante a execução de uma simulação este objeto pode assumir diferentes implementações para as classes do pacote *fenix.matematico.veiculo*.

O modelo do veículo implementado num simulador consiste numa visão interna da classe *Simulacao*. Já o modelo especificado pelo Fênix corresponde a sua visão externa. Esta classe manipula um modelo abstrato de um foguete, sendo que cada fase pode adotar o modelo matemático específico para o seu propósito.

A classe *Simulacao* também estabelece o comportamento para adicionar e executar as fases de vôo. Para construir uma fase específica de um simulador, deve-se estender a classe *Fase*, tornando concreto o comportamento definido para inicialização e finalização da fase, determinação da condição de término da fase, pré e pós integração das equações, escolha do algoritmo de integração e do conjunto de equações diferenciais que definem o movimento do foguete.

No pacote *veiculo*, as classes *Posicao*, *Aceleracao*, *Velocidade*, *VelocidadeAngular* e *Atitude* armazenam e manipulam as grandezas que definem o movimento do foguete, correspondendo a, respectivamente, posição, aceleração, velocidade, velocidade angular e atitude. Já a classe *Derivadas* especifica o cálculo da variação destas grandezas em função do tempo. Por último, as classes *Aerodinamica* e *Propulsao* possuem métodos que definem, respectivamente, as características de aerodinâmica e propulsão do veículo para o cálculo destas grandezas.

A classe *Veiculo* define métodos abstratos para acessar as grandezas referentes ao movimento do foguete, não possuindo objetos que representam estas grandezas. Neste caso, o modelo matemático (conjunto de equações) escolhido para calcular estas grandezas, corresponde a uma visão interna da instância da classe Veiculo e também pode ser definido posteriormente. Este métodos permitem que as demais classes consultem as grandezas relativas ao foguete, não importando como foram calculadas internamente.

Como o Fênix não abrange foguetes dotados de sistemas de controle, ajustes neste *Framework* serão necessários para suportar os modelos matemáticos que representam tais sistemas. O Processo Geral de Desenvolvimento de *Frameworks* prevê que o Fênix pode evoluir a medida que novos simuladores forem desenvolvidos.

4.7 Implementação do Fênix

Utilizou-se o IDE Borland Jbuilder, versões 2005 e 2007, para a implementação do *Framework* Fênix na linguagem Java, segundo o paradigma da orientação a objetos. Além disso, reusou-se a biblioteca *Jama*, disponível em (NIST, 2005), para implementar estruturas de dados do tipo matrizes.

O método *Executa()*, da classe *fenix.Simulacao*, define o fluxo de controle principal para a execução de uma simulação, como especificado pelo caso de uso FEN-CDU 02 - Simular Lançamento. O diagrama de seqüência observado na Figura 4.6 mostra os principais eventos implementados neste método, descritos com maiores detalhes no Apêndice D.

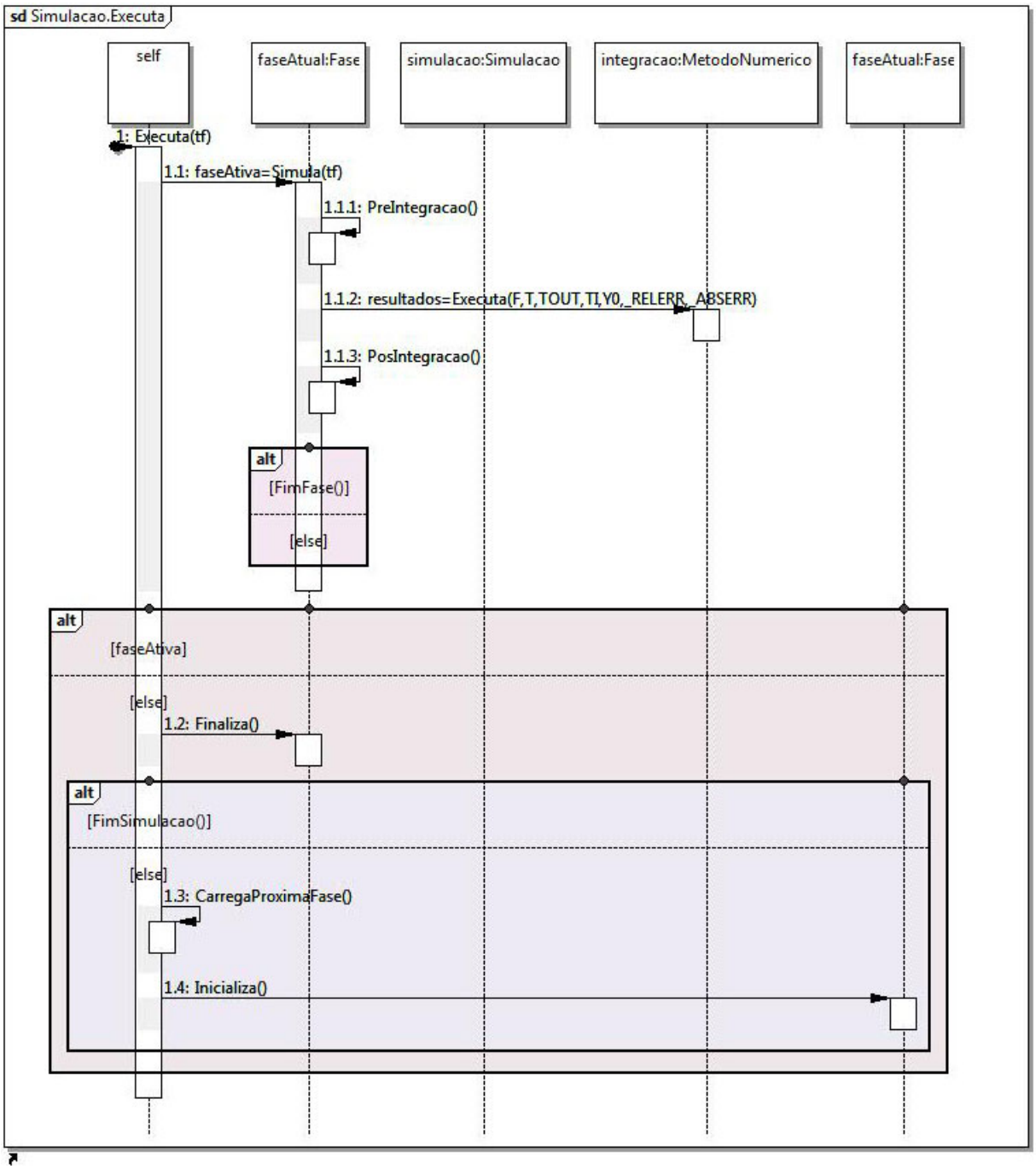

FIGURA 4.6 – Método Executa da classe Simulacao

Para o progresso da simulação, um Simulador solicita o avanço de tempo através da

invocação deste método, fornecendo o tempo desejado como parâmetro. Este método então solicita o avanço de tempo da fase atual, integrando a variação de movimento do veículo no intervalo considerado. Se após isto ocorrer o fim da fase, carrega-se a próxima ou retorna ao fim da simulação.

Para o reuso do *Framework* Fênix na construção de simuladores, o seu código precisa ainda ser verificado, a fim de garantir a validade de seus resultados para aplicações reais. O Fênix permite um Simulador realizar o caso de uso FEN-CDU 01 - Criar Simulação, através da implementação de um modelo concreto da classe *fenix.Simulacao* e das demais classes abstratas definidas nos pacotes *fenix.fases* e *fenix.matematico*.

Juntamente com o *Framework* Fênix, também desenvolveu-se um Estudo de Caso para verificar o seu modelo abstrato, descrito no Capítulo 5, que englobou a criação de três experimentos com base no Fênix, permitindo avaliar o seu nível de reusabilidade.

5 Estudo de Caso - Simulador de Trajetórias de Foguetes de Sondagem

Este Capítulo aborda o desenvolvimento de um Estudo de Caso elaborado para demonstrar o reuso do *Framework* Fênix em três experimentos. Desenvolveu-se um Protótipo de Simulador de Trajetórias de Foguetes de Sondagem, denominado SIMSonda, abrangendo a construção dos modelos de simulação 1D, 3D e 6D, utilizados pelo programa ROSI, descritos na Seção 3.2.1, do Capítulo 3.

O Estudo de Caso também propiciou demonstrar a utilização do padrão *IEEE STD 1516 - High Level Architecture (HLA)* em três experimentos. Idealizou-se uma Federação de Visualização Remota de Trajetórias de Foguetes, envolvendo a construção de três Federados, sendo o Federado que representa o Simulador SIMSonda responsável pela divulgação dos dados da trajetória simulada de um foguete, num ambiente de simulação distribuído.

5.1 Arquitetura do SIMSonda

A arquitetura proposta para o *Framework* Fênix, na Seção 4.1 do Capítulo 4, foi redefinida para o Simulador SIMSonda, conforme mostrado na Figura 5.1.

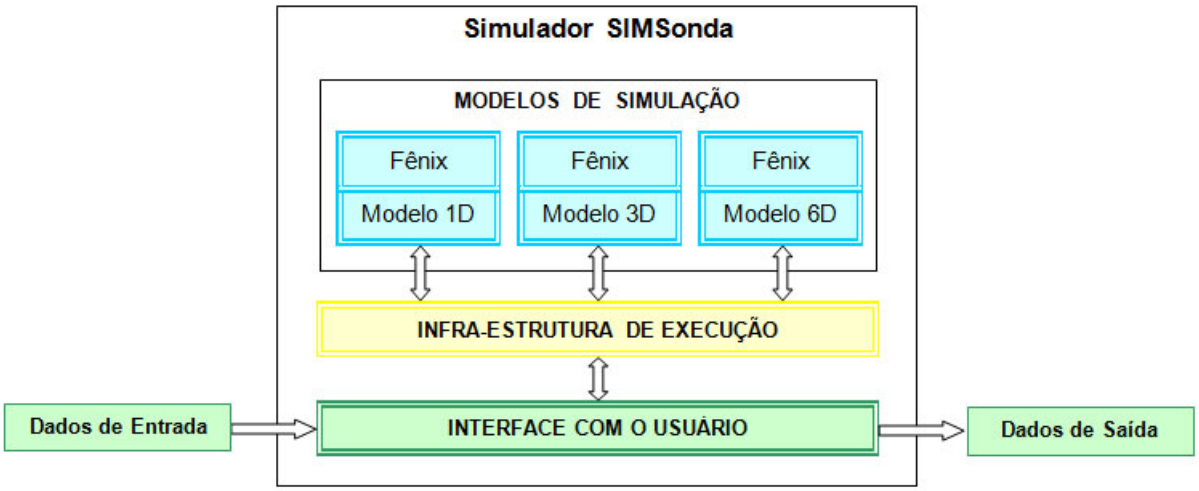

FIGURA 5.1 – Ambiente para simulação de trajetórias de veículos de sondagem

No Módulo dos Modelos de Simulação o *Framework* Fênix foi estendido para construção de três modelos concretos para a simulação de trajetórias. Os Modelos 1D, 3D e 6D calculam o movimento de um veículo em, respectivamente, uma, três ou seis dimensões. A estrutura do SIMSonda permite alterar dinamicamente o Modelo de Simulação durante a execução de uma simulação, associando cada fase de vôo a um destes modelos.

Com relação à Infra-Estrutura de Execução, desenvolveu-se uma versão *stand-alone* simples, que avança o tempo da simulação até o impacto do veículo com a superfície da terra. Já para executar uma simulação distribuída, utilizou-se uma versão acadêmica do *software* pRTI versão 1.3 (PITCH, 2005), que consiste numa implementação do padrão HLA, utilizando a linguagem Java.

Este *software* faz uso de tecnologias de sistema de computação distribuídos, proporcionando a comunicação e a sincronização necessárias para simulações distribuídas de lançamento de um foguete.

Para avaliar o esforço necessário para incorporar o padrão HLA no desenvolvimento de simuladores idealizou-se uma Federação de Visualização Remota da Trajetória de Fo-

guetes, composta por três federados: o Simulador SIMSonda, um Sistema de Visualização Remoto e um Sistema Coordenador, como ilustra a Figura 5.2.

FIGURA 5.2 – Federação do Estudo de Caso

O Federado Simulador SIMSonda calcula e publica os dados da trajetória do foguete a cada avanço de tempo na Federação. O Federado Sistema de Visualização reflete as atualizações desses dados remotamente, apresentando-as na interface com o usuário. Já o Federado Sistema Coordenador registra os pontos de sincronismo da Federação, além de permitir ao usuário suspender a execução da simulação.

5.2 Processo de Desenvolvimento do SIMSonda

Desenvolveu-se o Simulador SIMSonda, segundo o paradigma da Engenharia de *Software* do Ciclo de Vida da Prototipação (PRESSMAN, 2002; SOMMERVILLE, 2003). No projeto de desenvolvimento de uma versão completa de um simulador de trajetórias de foguetes, sugere-se a adoção do *Rational Unified Process - RUP* como Processo de Desenvolvimento.

O Processo de Desenvolvimento do Protótipo do SIMSonda representa parte do Processo de Desenvolvimento do *Framework* Fênix, como discutido na Seção 4.2, do Capítulo 4. A construção do SIMSonda permitiu verificar o modelo de classes do Fênix, proporci-

onando a sua evolução até um modelo considerado estável.

5.3 Requisitos do SIMSonda

Como um *framework* também possibilita o reuso de análise, os requisitos e restrições do *Framework* Fênix, descritos no Apêndice A, devem ser atendidos pelos simuladores que se derivam dele. Os requisitos funcionais do Fênix, especificados de forma abstrata, tornam-se concretos quando reutilizados para o Simulador SIMSonda.

Por exemplo, o Fênix deve propiciar a criação de modelos representando as entidades envolvidas: terra, atmosfera terrestre, veículo, radar e lançador. Já o Simulador SIMSonda deve propiciar a criação dos modelos destas entidades adotados pelo programa ROSI, descritos em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Para representar a terra, adotou-se os modelos *Geoid Internacional* ou IAU 1964. Já para a atmosfera terrestre, adotou-se o modelo *U.S. Standard Atmosphere* 1962. O levantamento de requisitos do SIMSonda encontra-se documentado com maiores detalhes no Apêndice E.

O Simulador SIMSonda deve propiciar as seguintes funcionalidades adicionais: a aquisição dos dados de entrada no formato utilizado pelo ROSI; a divulgação dos dados de saída através de tabelas e gráficos e nos formatos utilizados pelo programa ROSI; e a sua interação com outras aplicações geograficamente distribuídas.

Quanto à implementação, o SIMSonda deve propiciar a utilização da infra-estrutura de *runtime* pRTI1.3 (PITCH, 2005), que implementa o padrão HLA. Quanto às limitações do sistema, o Simulador SIMSonda pode ser utilizado apenas em computadores que possuam a *Java Virtual Machine (JVM)* 1.6 ou superior instalada. Além disto, para uma simulação distribuída ele pode ser utilizado apenas em computadores que disponham do *hardware*

de interface de rede.

5.4 Casos de Uso do SIMSonda

A Figura 5.3 mostra as funcionalidades que Simulador SIMSonda disponibiliza para o Usuário e para o Sistema de Visualização Remoto. Além disso, o diagrama aborda a interação entre o Simulador SIMSonda e o *Framework* Fênix. A documentação detalhada deste diagrama encontra-se disponível no Apêndice F. Nesta Figura, identificou-se os casos de uso do SIMSonda através da sigla SIM-CDU, seguida de sua numeração.

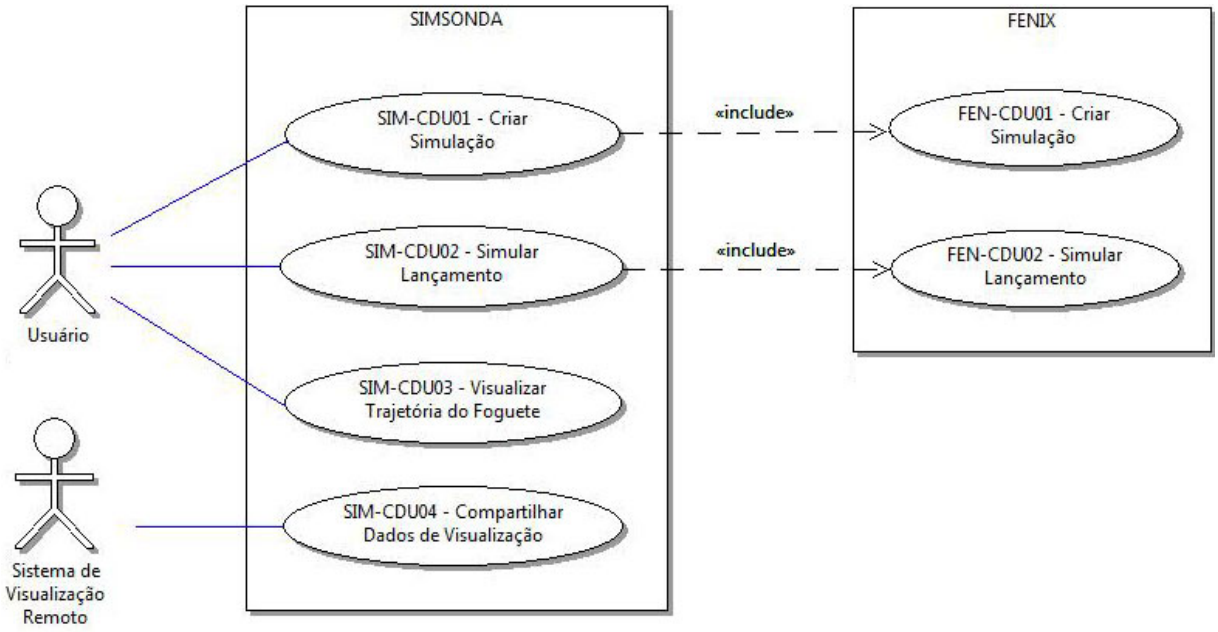

FIGURA 5.3 – Diagrama de Caso de Uso do Simulador

A Figura 5.3 apresenta a interação entre os atores e o Simulador SIMSonda, além da sua interação com o *Framework* Fênix. O Usuário pode utilizar o SIMSonda para criar uma nova simulação, simular o lançamento de um foguete e visualizar os resultados da simulação. O SIMSonda compartilha dados da trajetória com o Sistema de Visualização Remoto.

5.5 Estrutura de Pacotes de *Software* do SIMSonda

Com base na arquitetura de *software* idealizado na Seção 5.1, definiu-se a estrutura de pacotes, apresentada na Figura 5.4, para a construção do Protótipo do Simulador SIMSonda.

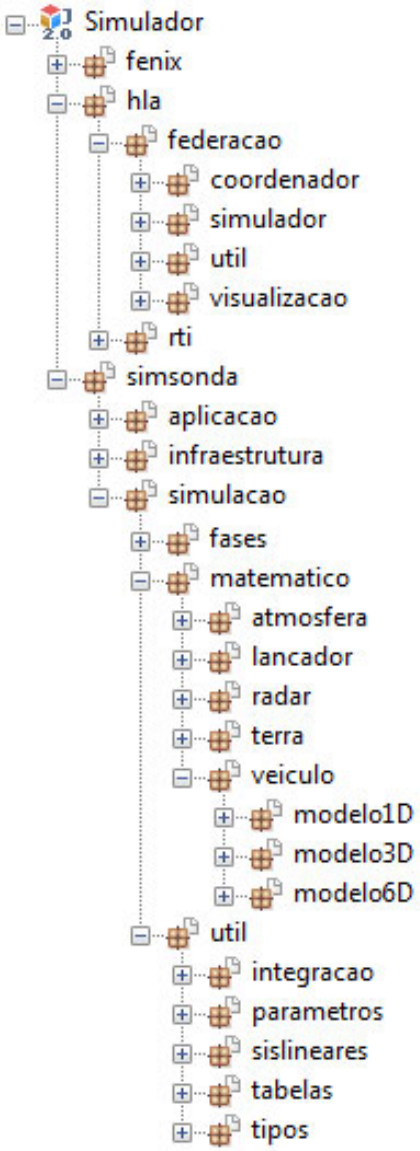

FIGURA 5.4 – Estrutura de Pacotes de *Software* para o Simulador SIMSonda

O projeto do Simulador SIMSonda divide-se em três pacotes principais: *fenix*, *hla* e *simsonda*. Desenvolveu-se o Protótipo do Simulador no pacote *simsonda*, com base no *Framework* especificado no pacote *fenix*, descrito no Capítulo 4.

O pacote *hla.rti* refere-se a uma biblioteca disponibilizada juntamente com o *software* pRTI, para o desenvolvimento dos Federados e Federações. No pacote *hla.federacao* desenvolveu-se os Federados da Federação para Visualização Remota da Trajetória nos pacotes *coordenador*, *simulador* e *visualizacao*.

O pacote *simsonda* divide-se em *aplicacao*, *infraestrutura* e *simulacao*. O pacote *simsonda.aplicacao* refere-se ao Módulo de Interface com o Usuário, abordado na Seção 5.1.

O Módulo de Infra-Estrutura de Execução foi desenvolvido numa versão simples, no pacote *simsonda.infraestrutura*, que controla o avanço de tempo da simulação e a visualização dos dados da trajetória calculada. Esta Infra-estrutura foi criada para que o Simulador SIMSonda seja independente do *software* pRTI.

Os modelos para a simulação da trajetória de foguetes de sondagem foram estabelecidos no pacote *simsonda.simulacao*. Ele consiste de três experimentos para reuso do *Framework* Fênix. O pacote *simsonda.simulacao.fases* corresponde ao modelo de fases da simulação, onde se definiu os tipos de Fase 1D, 3D e 6D.

O pacote *simsonda.simulacao.matematico* determina o modelo das entidades envolvidas na simulação, com base no equacionamento descrito em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Este pacote divide-se nos pacotes *atmosfera*, *lancador*, *radar*, *terra* e *veiculo*. Desenvolveu-se três modelos de veículos espaciais nos pacotes *modelo1D*, *modelo3D* e *modelo6D* do pacote *veiculo*, especializando-se o modelo abstrato de veículo definido no *Framework* Fênix, considerando-se, respectivamente, um, três e seis graus de liberdade.

5.6 Classes do SIMSonda

Para a construção do Módulo Modelos de Simulação do Simulador SIMSonda, herdou-se o modelo de classes do *Framework* Fênix, implementando seu comportamento abstrato para os Modelos 1D, 3D e 6D, como ilustra a Figura 5.5. O modelo de classes estabelecido para o SIMSonda encontra-se descrito detalhadamente no Apêndice G.

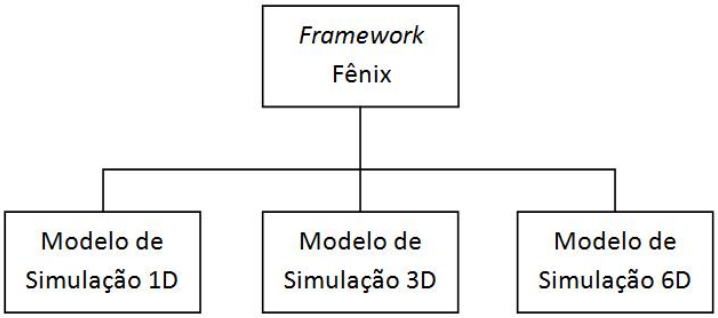

FIGURA 5.5 – Reuso do Fênix na criação dos modelos do SIMSonda

Com relação ao Módulo do Modelo Matemático, criou-se classes concretas representado as entidades atmosfera, radar, lançador, terra e veículo. A Figura 5.6 apresenta como a classe *Veiculo*, do *Framework* Fênix, foi reutilizada para criar os diferentes modelos de veículos espaciais. O mesmo ocorreu para as demais classes do pacote *fenix.matematico.veiculo*, como: *Posicao*, *Velocidade*, *Derivadas*, dentre outras.

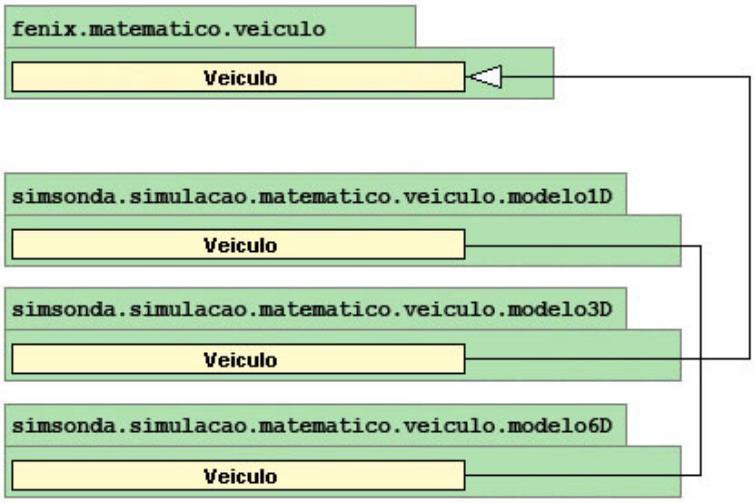

FIGURA 5.6 – Especialização da classe Veículo

Quanto ao Módulo do Modelo de Fases, estendeu-se a classe *fenix.simulacao.Fase* obtendo-se um modelo concreto para os diferentes tipos de fases do SIMSonda, como mostra a Figura 5.7. Cada tipo de fase relaciona-se a um dos modelos matemáticos que representam o veículo. Isso demonstra como o Fênix possibilita alterar facilmente os modelo matemáticos das entidades entre as fases de vôo.

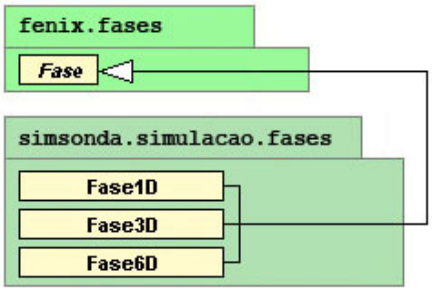

FIGURA 5.7 – Especialização da classe Fase

Com relação ao Módulo de Infra-estrutura de Execução, desenvolveu-se uma infra-estrutura simples no pacote *simsonda.infraestrutura*. Além disso, para realizar uma execução da Federação para Visualização Remota de Trajetória, utilizou-se o *software* pRTI (PITCH, 2005), que disponibiliza um conjunto de classes para permitir a comunicação entre os Federados e a RTI.

Cada Federado precisa implementar a interface *FederateAmbassador*, usada pela RTI para invocar métodos do Federado. Além disso, um Federado pode usar a interface *RTI-Ambassador* para invocar os serviços disponibilizados pela RTI. A Figura 5.8 ilustra como o Federado SIMSonda utiliza estas interfaces de comunicação.

O Federado SIMSonda obtém uma instância da interface *RTIAmbassador* e, ao se conectar numa Federação na RTI, fornece a sua instância da classe *FedAmbImpl*, que implementa a interface *FederateAmbassador*. O mesmo também também ocorre para os Federados Coordenador e Sistema de Visualização.

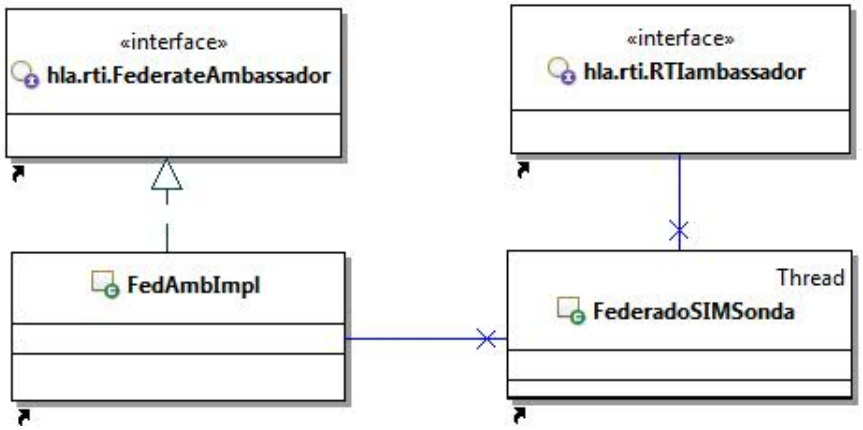

FIGURA 5.8 – Implementação das interfaces de comunicação

5.7 Implementação do SIMSonda

A implementação do Protótipo do Simulador SIMSonda ocorreu segundo o paradigma da orientação a objetos, utilizando o IDE Borland JBuilder, versões 2005 e 2007. Reusou-se a biblioteca *CloseAndMaxTabbedPane*, disponível em (JAVAWORLD.COM, 2004), para inserir na interface gráfica um componente do tipo painel com múltiplas guias personalizadas. Além disso, para produzir os gráficos utilizou-se a biblioteca *JFreeChart*, disponível em (JFREE.ORG, 2000). A Figura 5.9 apresenta o Módulo de Interface com o Usuário do SIMSonda.

A Figura 5.9 ilustra como pode ser realizada a entrada de dados numa simulação, sendo as fases organizadas numa estrutura do tipo árvore. Buscou-se uma interface mais amigável ao usuário, permitindo adicionar os parâmetros de cada fase pelo seu código ou descrição e preencher as suas tabelas através de um *grid*, facilitando a configuração das fases.

O *Framework* Fênix define estruturas de dados abstratas para armazenar, respectivamente, os parâmetros e as tabelas de entrada. Já no Simulador SIMSonda, estas estruturas foram implementadas, de forma concreta, conforme o formato e a identificação utilizada

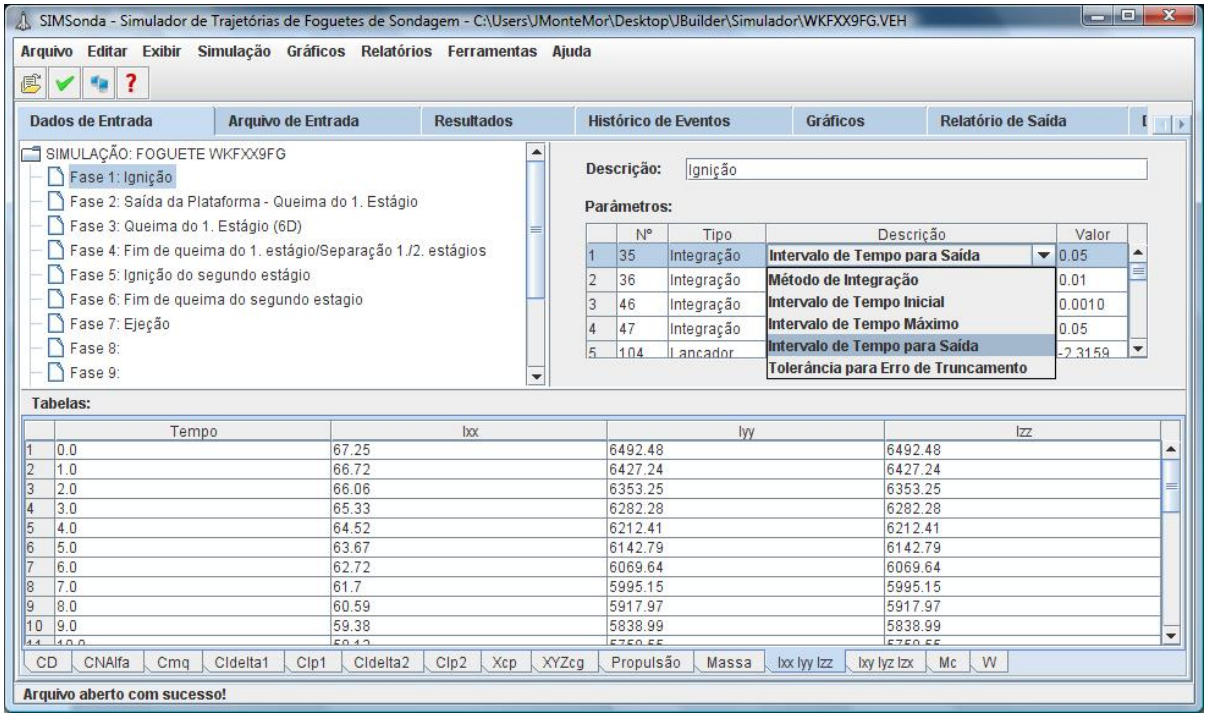

FIGURA 5.9 – Interface visual para entrada de dados

pelo programa ROSI, descritos em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). A Figura G.10, do Apêndice G, apresenta as classes que manipulam os parâmetros e tabelas de entrada, além dos dados de saída da simulação.

A Figura 5.10 mostra a forma tradicional de entrada através de um arquivo texto utilizada pelo programa ROSI. O formato deste arquivo de entrada foi mantido para o Simulador SIMSonda.

As Figuras 5.11 e 5.12 mostram, respectivamente, os arquivos de saída no formato padrão e para visualização gráfica, adotados pelo ROSI, sendo ambos também mantidos para o SIMSonda.

Além desta forma antiga de saída, os resultados de uma simulação também são armazenados numa estrutura de dados do tipo tabela, visando facilitar a sua manipulação posterior, como exibido na Figura 5.13. Usou-se também uma tabela para registrar os eventos da simulação como: mudanças de fases, apogeu e impacto do foguete, dentre

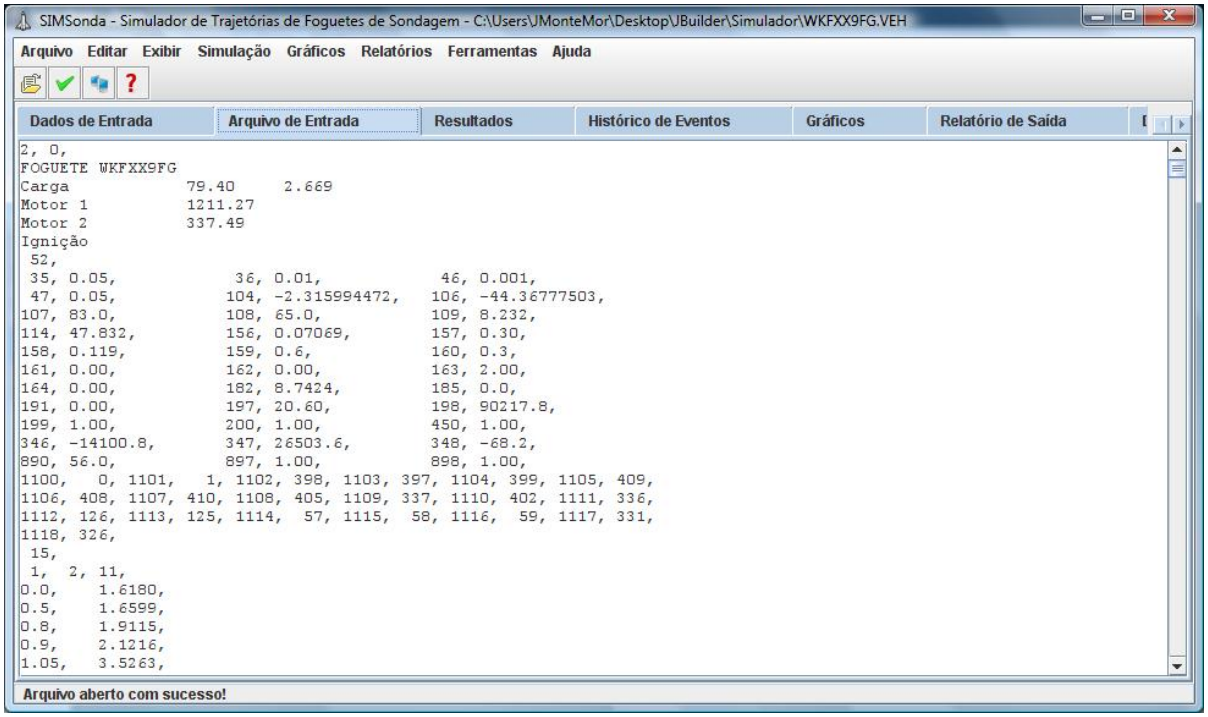

FIGURA 5.10 – Interface de visualização do arquivo de entrada

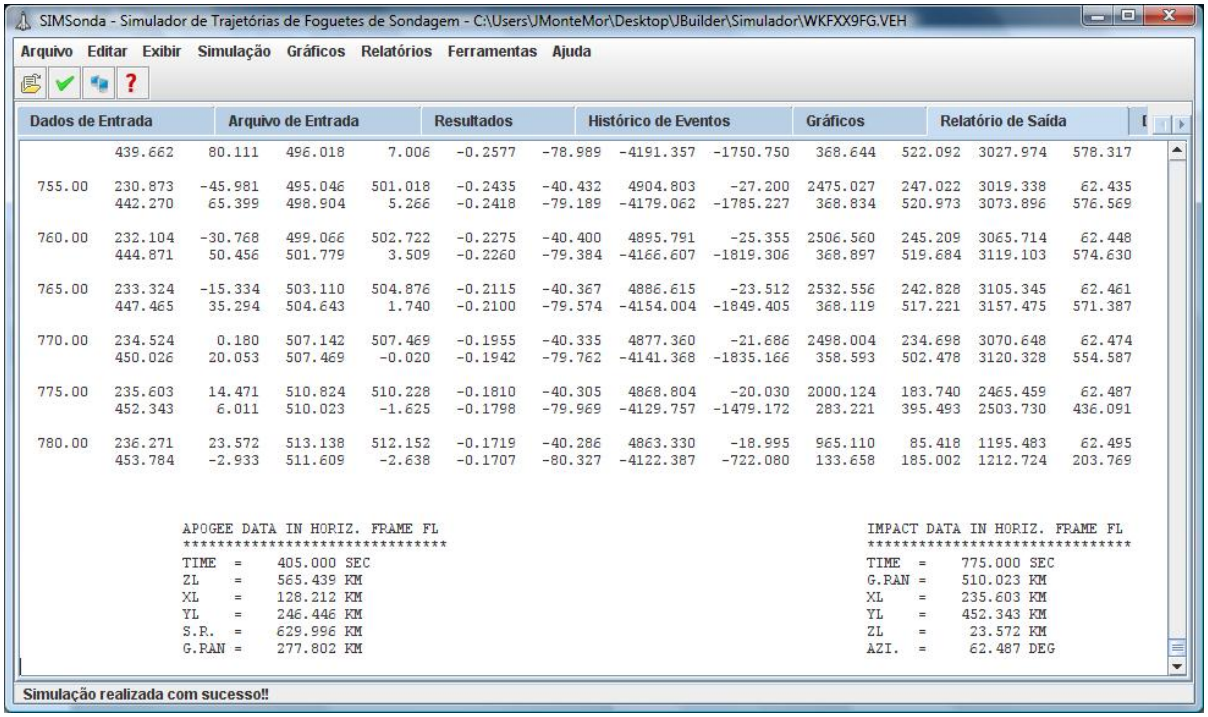

FIGURA 5.11 – Interface do arquivo de saída padrão

outros, como ilustra a Figura 5.14.

Para finalizar, com o objetivo de facilitar a interpretação dos resultados, o Simulador SIMSonda também produz gráficos com os valores de algumas grandezas em função do

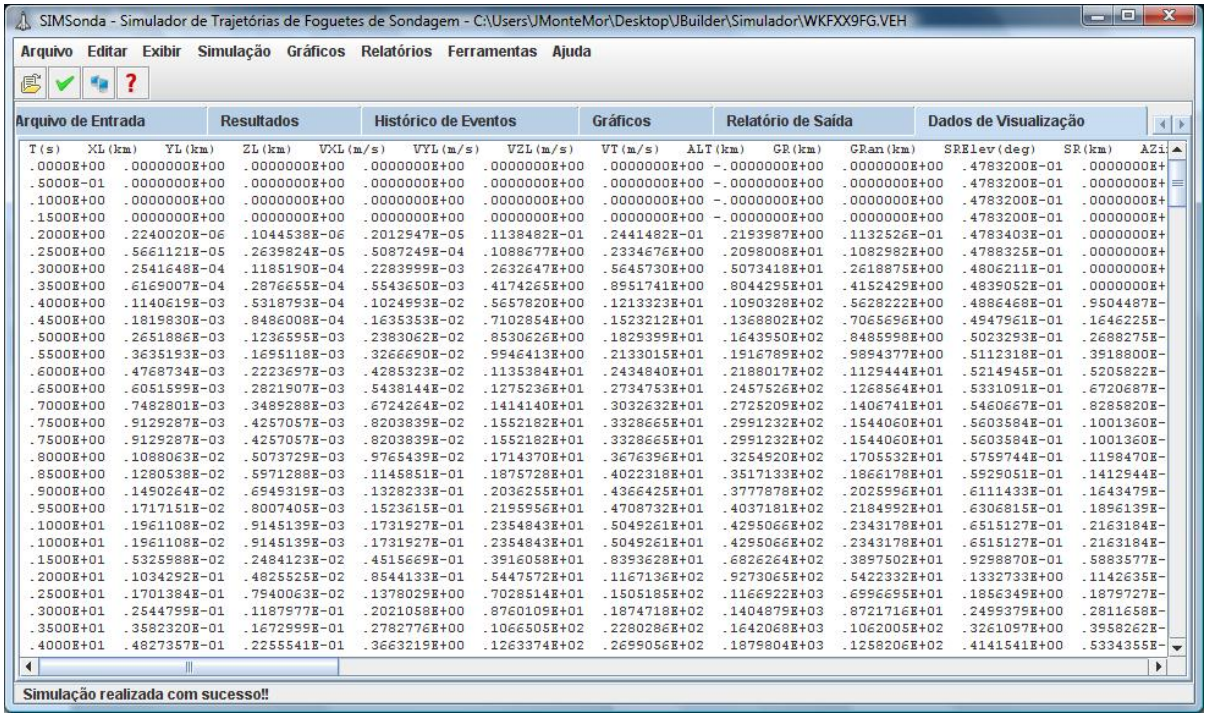

FIGURA 5.12 – Interface do arquivo de com dados para visualização

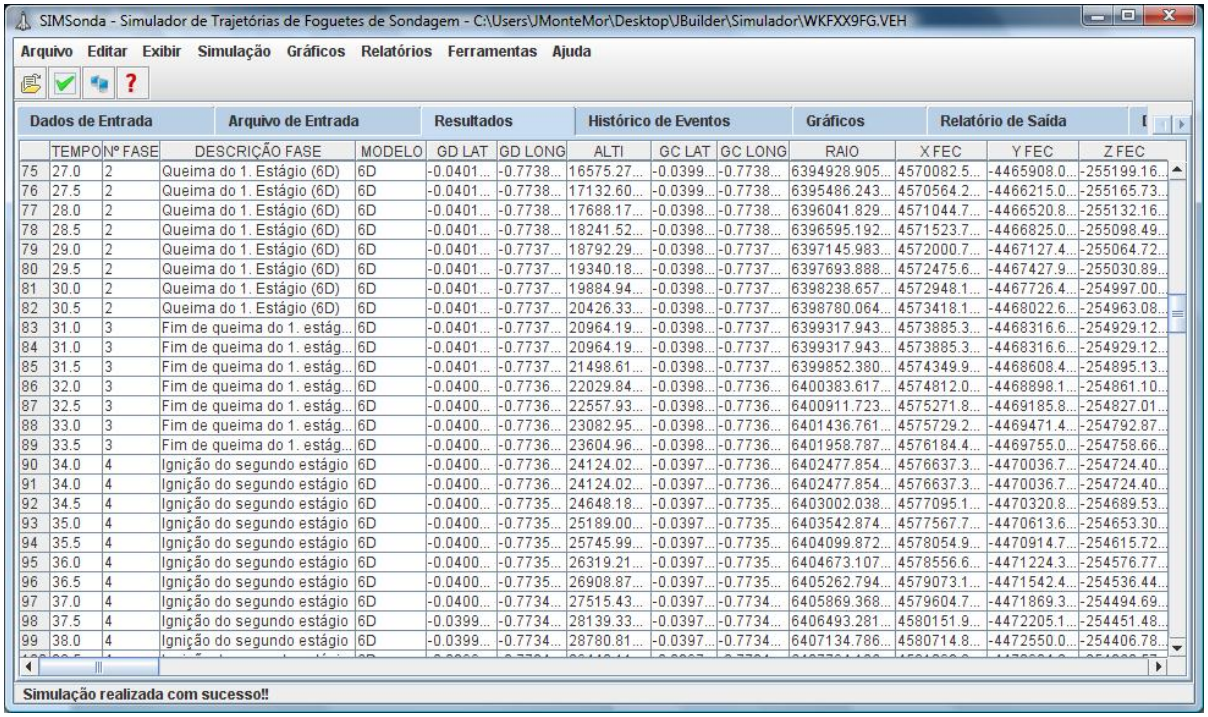

FIGURA 5.13 – Interface da tabela de resultados

tempo de vôo, conforme mostrado na Figura 5.15.

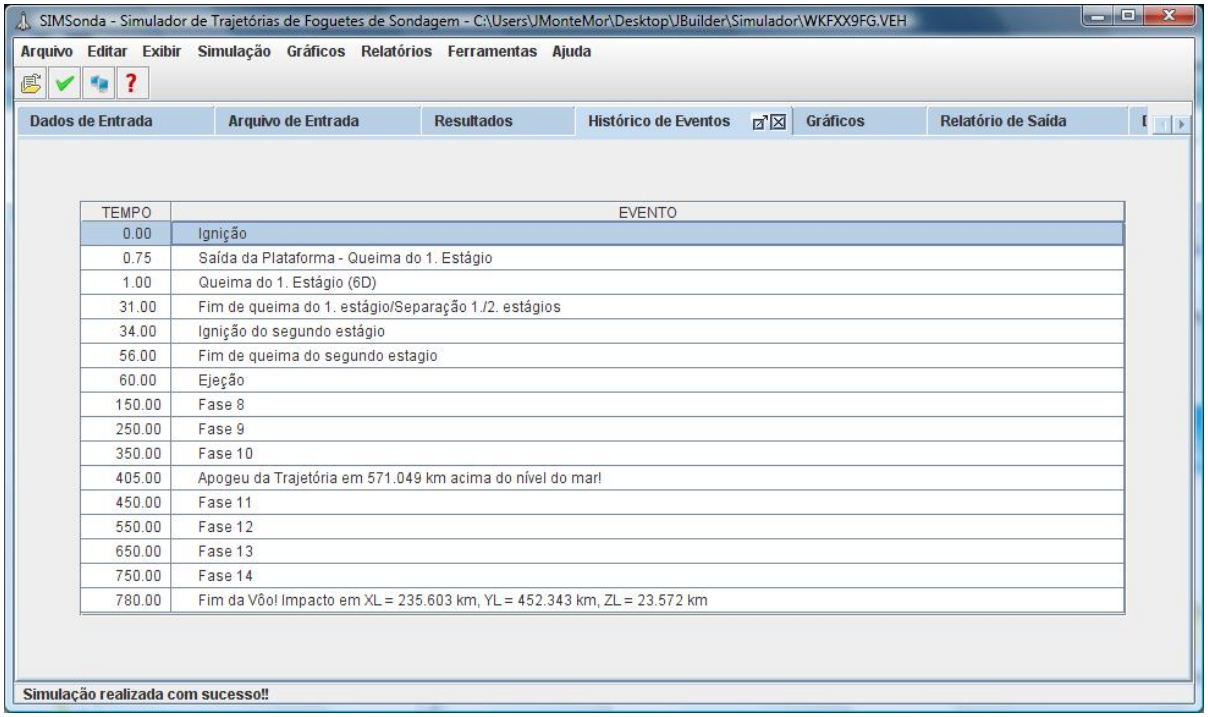

FIGURA 5.14 – Interface da tabela de eventos

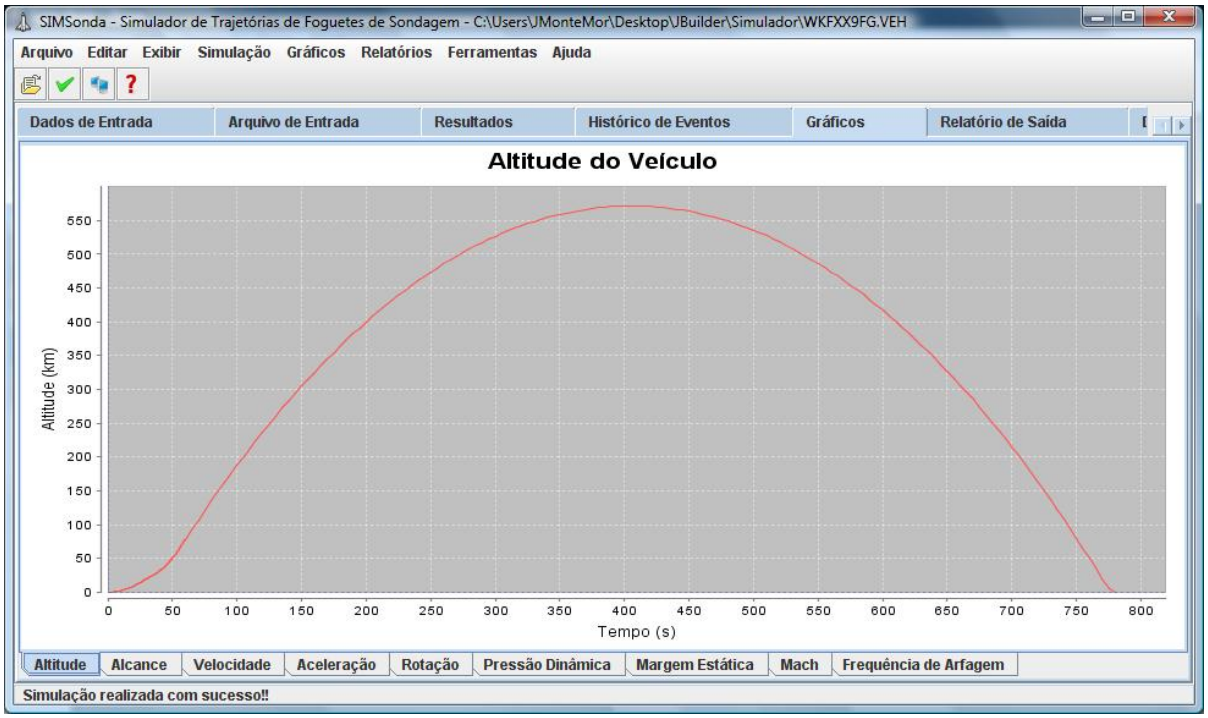

FIGURA 5.15 – Interface do gráfico de altitude

5.7.1 Infra-estrutura de Execução

O Protótipo do Simulador SIMSonda permite a execução de uma simulação através de uma Infra-estrutura de Execução simples, implementada na classe *Simulador*, do pacote

simsonda.infraestrutura. A Figura 5.16 apresenta os principais eventos executados pelo método *run()* desta classe, descritos detalhadamente no Apêndice H.

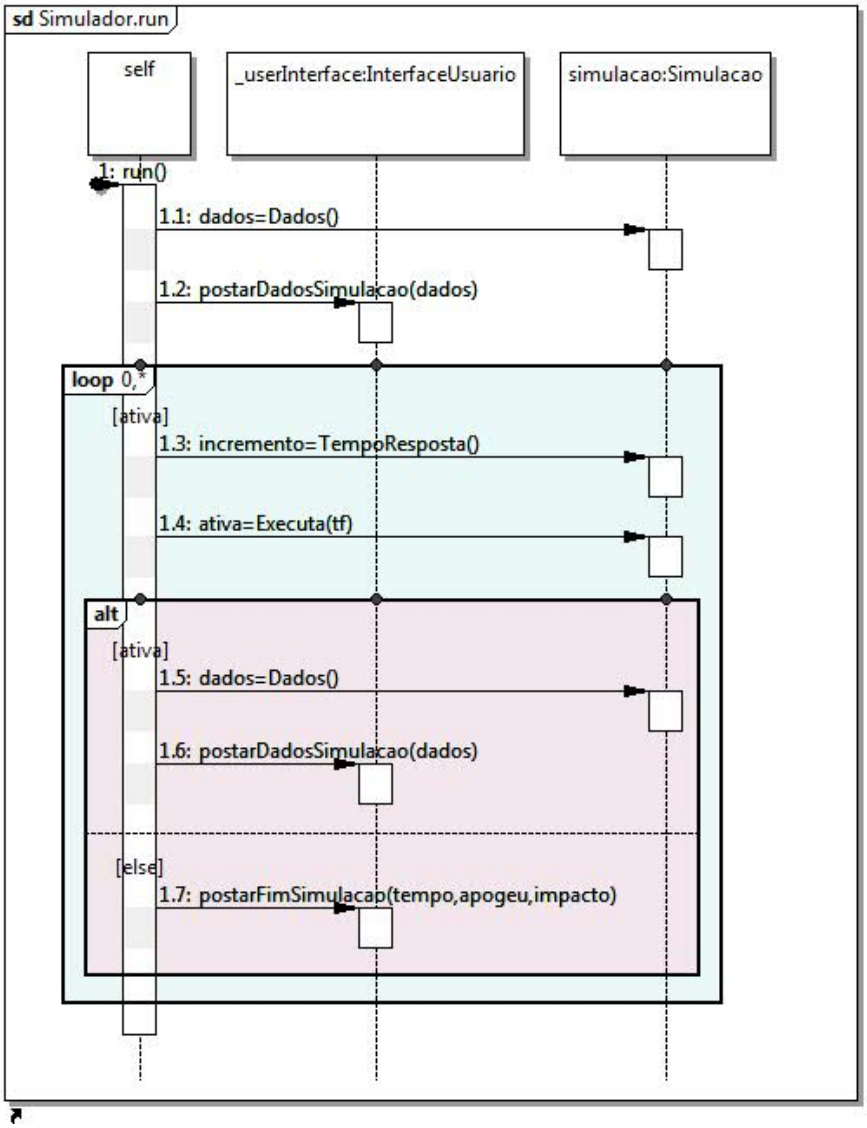

FIGURA 5.16 – Diagrama de sequência para infra-estrutura padrão

A Infra-estrutura solicita o avanço de tempo da simulação em intervalos de tempo fixos. Até ocorrer o fim da simulação, a cada avanço de tempo são apresentados na Interface com o Usuário os dados referentes a trajetória do foguete.

Além desta Infra-estrutura de Execução, o Simulador SIMSonda permite a execução de uma Federação para Visualização Remota de Trajetórias simuladas de um foguete de sondagem, através da Infra-estrutura de *Runtime* pRTI. Esta Federação possui três

Federados membros, chamados de Sistema Coordenador, Simulador SIMSonda e Sistema de Visualização, sendo discutida na Seção 5.7.2.

5.7.2 Federação de Visualização Remota da Trajetória de Foguetes

A Federação de Visualização Remota de Trajetórias de Foguetes, desenvolvida como Estudo de Caso, envolve três Federados, como mostrado na Figura 5.2. Construiu-se o Federado SIMSonda adicionando-se ao Simulador SIMSonda, um componente desenvolvido para comunicação com o *software* pRTI.

O Federado Coordenador tem como responsabilidade registrar os pontos de sincronismo da Federação e controlar a quantidade de Federados membros, além de apresentar estas informações em sua interface com o usuário, como apresentado na Figura 5.17.

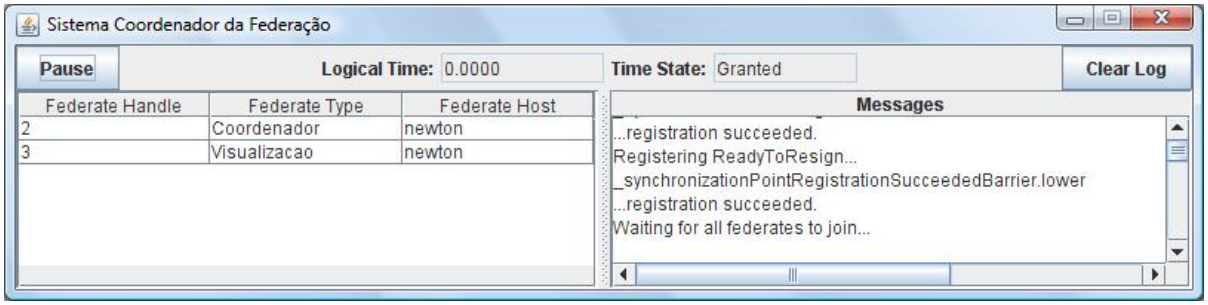

FIGURA 5.17 – Interface com o usuário do Sistema Coordenador

O Federado Sistema de Visualização recebe os dados que definem a trajetória de um foguete e apresenta, na sua interface com o usuário, estes dados através de uma tabela e a altitude do foguete através de um gráfico, semelhantes aos produzidos para o Simulador SIMSonda mostrados nas Figuras 5.13 e 5.15.

Criou-se um Sistema de Visualização com interface gráfica com o usuário bem simples, pois o objetivo deste Estudo de Caso foi avaliar o esforço para implementação do

padrão HLA neste Federado. Em seu trabalho de pesquisa, intitulado “Um Ambiente De Visualização da Evolução da Trajetória de Veículos Espaciais Brasileiros”, Silva (2007) desenvolveu um sistema para a visualização tridimensional da trajetória de foguete. O Simulador SIMSonda também poderia interagir com este Sistema de Visualização 3D através da arquitetura proposta pelo padrão HLA.

Para a construção da Federação torna-se necessário a definição do Modelo de Objetos da Federação (*Federation Object Model - FOM*), correspondendo aos dados compartilhados entre os Federados membros da Federação. A Figura 5.18 ilustra partes do arquivo que contém a declaração do FOM para esta Federação.

```
(FED
        (Federation TRAJETORIA)
        (FEDversion v1.3)
        (spaces)

        (objects
                (class ObjectRoot
                        (attribute privilegeToDeleteObject reliable timestamp)
                        (class Veiculo
                                (attribute T reliable timestamp)
                                (attribute XL reliable timestamp)
                                (attribute YL reliable timestamp)
                                (attribute ZL reliable timestamp)
                                (attribute VXL reliable timestamp)
                                (attribute VYL reliable timestamp)
                                (attribute VZL reliable timestamp)
                                (attribute ALT reliable timestamp)
                                        :
                        ) ;; fim classe Veiculo
                                :
                ) ;; fim classe ObjectRoot
        )  ;; fim da declaração de objetos

        (interactions
                (class InteractionRoot reliable timestamp
                        (class SimulationEnds reliable timestamp)
                                :
                )  ;; fim da classe InteractionRoot
        )  ;; fim da declaração de interações
)  ;; fim do modelo de objetos
```

FIGURA 5.18 – Modelo de Objetos da Federação

Este arquivo utiliza o formato adotado pelo padrão HLA para definir os Dados de

Execução da Federação (*Federation Execution Data - FED*). Estas informações são requeridas pela RTI para suportar a execução de uma Federação e seu formato é o mesmo para qualquer RTI. Além das informações compartilhadas pelos Federados (FOM), este arquivo contém informações mantidas pela RTI para suportar a execução da Federação. O Federado Coordenador faz uso destas informações para o controle da quantidade de Federados membros da Federação.

Na especificação do FOM declarou-se uma classe de objetos chamada *Veiculo*, cujos atributos correspondem aos dados do foguete compartilhados para a visualização remota. A Figura 5.18 apresentou apenas alguns atributos desta classe, sendo a maioria omitida para simplificar a estrutura do arquivo. Além desta classe de objetos, existe um classe de interação denominada *SimulationEnds* usada pra divulgação do fim da simulação aos Federados membros.

O Federado Simulador registra os atributos da classe *Veiculo*, sendo responsável pela atualização de seus valores a cada avanço de tempo da simulação. Ele também é responsável pelo envio da interação definida na classe *SimulationEnds*, quando ocorrer o fim da simulação. O Federado Visualização se inscreve nos atributos da classe *Veiculo* para receber os dados atualizados da trajetória simulada do foguete e na classe de interação informando o fim da simulação.

A classe de interação *SimulationEnds* e todos os atributos da classe *Veiculo* foram definidos com as características *reliable* e *timestamp*. A primeira característica solicita que a RTI utilize uma comunicação confiável, garantindo a entrega das informações enviadas. Já a segunda característica estabelece que as informações devem ser entregues ordenadas em função do tempo de envio.

A Figura 5.19 ilustra o ciclo de vida de um Federado membro da Federação para

Visualização Remota de Trajetórias da Figura 5.19. Este ciclo foi definido em (KUHL; WEATHERLY; DAHMANN, 1999) e adaptado para a Federação idealizada neste Estudo de Caso. Os diagramas de seqüência referentes ao ciclo de vida de cada Federado participando da Federação são apresentados e descritos no Apêndice H.

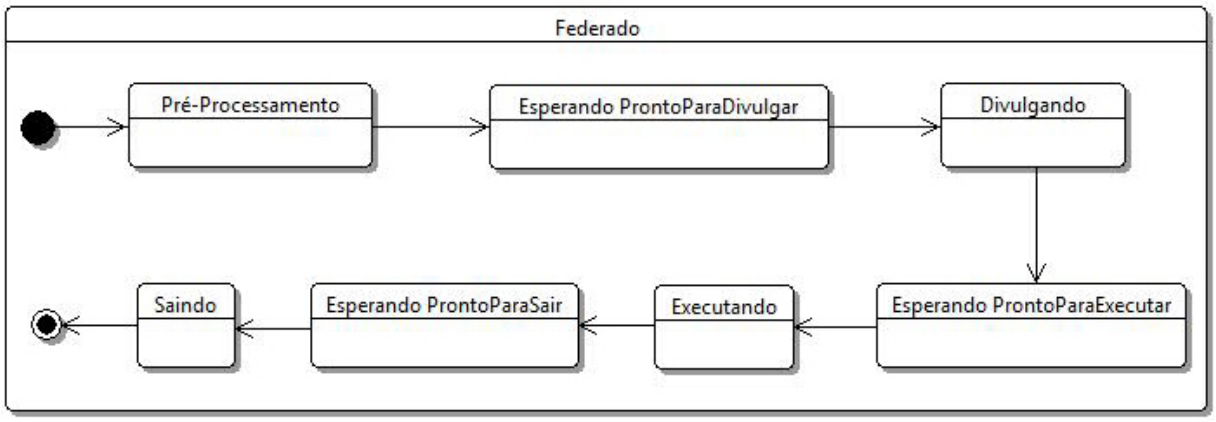

FIGURA 5.19 – Ciclo de vida de um federado

Após ingressar na Federação, um Federado assume o estado de *Pré-Processamento*, onde ele define para quais atributos de classes de objetos ele irá publicar novos valores e quais classe de interações ele irá enviar. Além disso, ele se inscreve nos atributos que deseja receber atualizações de valores e nas interações de interesse.

Em seguida, o Federado alcança o primeiro ponto de sincronismo da Federação, denominado *ProntoParaDivulgar*, e assume o estado *Esperando ProntoParaDivulgar*. Neste estado, o Federado já definiu quais informações do FOM ele tem interesse e deve aguardar que os demais Federados também alcancem este ponto de sincronismo. Isto garante que nenhum Federado comece a divulgar valores de atributos ou enviar interações antes que um Federado não tenha se inscrito nessas informações.

Quando a RTI informar que a Federação está sincronizada no ponto *ProntoParaDivulgar*, o Federado assume o estado *Divulgando*, onde ele registra as instâncias iniciais dos objetos que ele compartilha e define os valores iniciais para os atributos dessa instância.

Após isso, o Federado alcança o segundo ponto de sincronismo, chamado *ProntoParaExecutar*, e assume o estado *Esperando ProntoParaExecutar*. Este ponto garante que os objetos já foram registrados pelos Federados e os valores iniciais de seus atributos já foram publicados para os outros Federados interessados nestas informações.

Ao receber a confirmação da RTI que a Federação está sincronizada no ponto *ProntoParaExecutar*, o Federado conduz a simulação solicitando a RTI o avanço de seu tempo local até receber a interação informando o fim da simulação. A cada solicitação de avanço de tempo, o Federado deve processar as invocações da RTI em sua implementação da interface FederateAmbassador, até receber a permissão para o avanço de seu tempo lógico.

Ao receber a interação de fim da simulação, o Federado alcança o ponto de sincronismo *ProntoParaSair* e assume o estado *Esperando ProntoParaSair*. Este ponto garante que todos os Federados já estão prontos para deixar a Federação, permitindo que sua execução seja removida da RTI. Quando a Federação for sincronizada neste ponto, o Federado assume o estado *Saindo* da Federação e finaliza seu ciclo de vida.

O desenvolvimento dos Federados Sistema Coordenador, Simulador SIMSonda e Sistema de Visualização permitiu avaliar a adoção do padrão HLA na criação de uma simulação distribuída, sendo os resultados discutidos no Capítulo 6.

6 Discussão dos Resultados

6.1 Reusabilidade do *Framework* Fênix

Landin e Niklasson (1995) define um *framework* como um componente de *software*, provendo larga escala de reuso de análise, de projeto e de código, como já discutido no Capítulo 2. O reuso deste três elementos foi constatado durante o desenvolvimento do Simulador SIMSonda.

Quanto ao reuso de análise, verificou-se a reusabilidade de requisitos, casos de uso e classes do *Framework* Fênix, durante a fase de análise do Simulador SIMSonda. Os Apêndices A e E descrevem, respectivamente, os requisitos do *Framework* Fênix e do Simulador SIMSonda. Constatou-se que os requisitos do Fênix também devem ser atendidos pelos simuladores que derivam-se dele. Alguns requisitos do Fênix foram definidos de forma abstrata, por exemplo, a criação do modelo matemático representando a atmosfera terrestre. No SIMSonda este requisito foi reescrito, tornando concreto o modelo da atmosfera, ao especificar o modelo *U.S. Standard Atmosphere* 1962, e.g., (KRAMER; CRAUBNER; ZIEGLTRUM, 1976).

Os casos de uso do Fênix, descritos no Apêndice B, foram reutilizados na construção do modelo de casos de uso do SIMSonda, descrito no Apêndice F. Por exemplo, o fluxo

de eventos do caso de uso *FEN-CDU01 - Criar Simulação*, do *Framework* Fênix, foi reutilizado como um passo do fluxo de eventos definido para o caso de uso *SIM-CDU01 - Criar Simulação*, do Simulador SIMSonda.

Para a criação do modelo de classes do Simulador SIMSonda, apresentado no Apêndice G, reutilizou-se todas as classes do *Framework* Fênix, descritas no Apêndice C. Verificou-se a reusabilidade das classes, dos relacionamentos entre elas e da estrutura de pacotes definida para o *Framework* Fênix.

Já quanto ao reuso de projeto, o *Framework* Fênix englobou o projeto de algoritmos abstratos para a simulação de trajetórias de foguetes de sondagem. Ele definiu as interfaces e os obstáculos que o Simulador SIMSonda precisou satisfazer para tornar a implementação destes algoritmos concreta. Qualquer simulador criado com base no *Framework* Fênix deve implementar todos os métodos abstratos definidos em suas classes e interfaces.

Com relação ao reuso de código, para se medir o grau de reusabilidade do *Framework* Fênix, adotou-se como métrica a contagem de linhas de código. Analisou-se a quantidade de linhas escritas para o desenvolvimento do *Framework* Fênix, que foram reutilizadas para a construção do Protótipo do Simulador SIMSonda.

A Tabela 6.1 apresenta a quantidade de linhas de código para os pacotes *fenix*, *hla* e *simsonda*, desprezando-se as linhas em branco e de comentário.

Nome do Pacote	Linhas	%
fenix	1.652	13
hla	2.928	24
simsonda	7.715	63
Total	12.295	100

TABELA 6.1 – Quantidade de linhas de código do Protótipo do Simulador SIMSonda

Na Tabela 6.1 não foram consideradas as linhas de código das classes dos pacotes *hla.rti* e *simsonda.aplicacao.tabbedpane*, que correspondem a bibliotecas externas reusadas, res-

pectivamente, para construção dos Federados e da Interface Gráfica com o Usuário.

Em relação ao Protótipo SIMSonda, a quantidade de código do *Framework* Fênix corresponde a uma porcentagem muito pequena de código-fonte. De acordo com a Arquitetura de *Software* proposta na Seção 5.1, do Capítulo 5, o código do Protótipo abrange, além do Módulo dos Modelos de Simulação, os Módulos de Interface com o Usuário e de Infra-estrutura de Execução, como mostrado na Tabela 6.2.

Nome do Pacote	Linhas	%
aplicacao	3.591	47
simulacao	4.036	52
infraestrutura	88	1
Total	7.715	100

TABELA 6.2 – Quantidade de linhas de código do pacote *simsonda*

Os pacotes *simsonda.aplicacao* e *simsonda.infraestrutura* serão desprezados para a análise da reusabilidade do *Framework* Fênix, pois o Módulo de Interface com o Usuário é específico para o Simulador SIMSonda e o Módulo de Infra-estrutura de Execução é simples, com poucas linhas de código, sendo reutilizado para qualquer Simulador.

O enfoque na criação do *Framework* Fênix esteve apenas no Módulo do Modelo de Simulação. Assim, torna-se interessante comparar o *Framework* Fênix apenas com a implementação concreta dos três Modelos 1D, 3D e 6D do Simulador SIMSonda, que correspondem a 52% do código-fonte do pacote *simsonda*. A Tabela 6.3 apresenta a quantidade de código nos pacotes *fenix* e *simsonda.simulacao*.

Nome do Pacote	Linhas	%
fenix	1.652	29
simsonda.simulacao	4.036	71
Total	5.688	100

TABELA 6.3 – Quantidade de linhas de código para o Módulo dos Modelos de Simulação

Embora 29% do Modelo de Simulação do Simulador SIMSonda seja reaproveitado do

Framework Fênix, deve-se ressaltar que o Protótipo envolveu realmente a implementação de três Modelos de Simulação com base no Fênix, como mostrado na Figura 6.1. A Tabela 6.4 apresenta a quantidade de linhas de código de cada componente do pacote *simsonda.simulacao*.

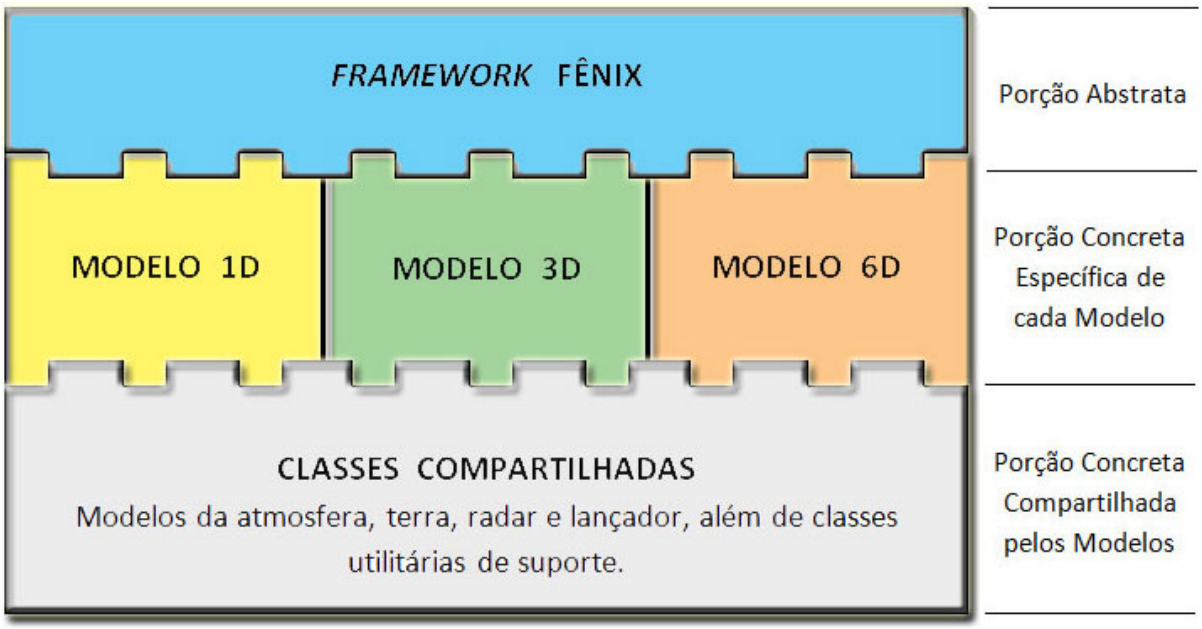

FIGURA 6.1 – Componentes dos Modelos de Simulação do SIMSonda

Nome do Pacote ou Classe	Linhas	%
Simulacao	79	2
fases	922	23
matematico.util	1.235	30
matematico.atmosfera	161	4
matematico.terra	106	2
matematico.lancador	31	1
matematico.radar	12	1
matematico.veiculo.modelo1D	360	9
matematico.veiculo.modelo3D	417	10
matematico.veiculo.modelo6D	713	18
Total	4.036	100

TABELA 6.4 – Quantidade de linhas de código do pacote *simsonda.simulacao*

Os modelos das entidades atmosfera, terra, radar e lançador não mudam nos Modelos 1D, 3D e 6D. Além disto, alguns pacotes que formam o pacote *simsonda.simulacao.util* também são compartilhados por estes três modelos.

A Figura 6.2 apresenta as classes que integram o Modelo de Simulação 1D. A classe *fases.Fase1D* associa-se ao modelo matemático do veículo definido na classe *Veiculo* do pacote *veiculo.modelo1D*. O mesmo ocorre em relação as classes *fases.Fase3D* e *veiculo.modelo3D.Veiculo* para o Modelo de Simulação 3D, como mostrado na Figura 6.3, e com as classes *fases.Fase6D* e *veiculo.modelo6D.Veiculo* para o Modelo de Simulação 6D, apresentado na Figura 6.4.

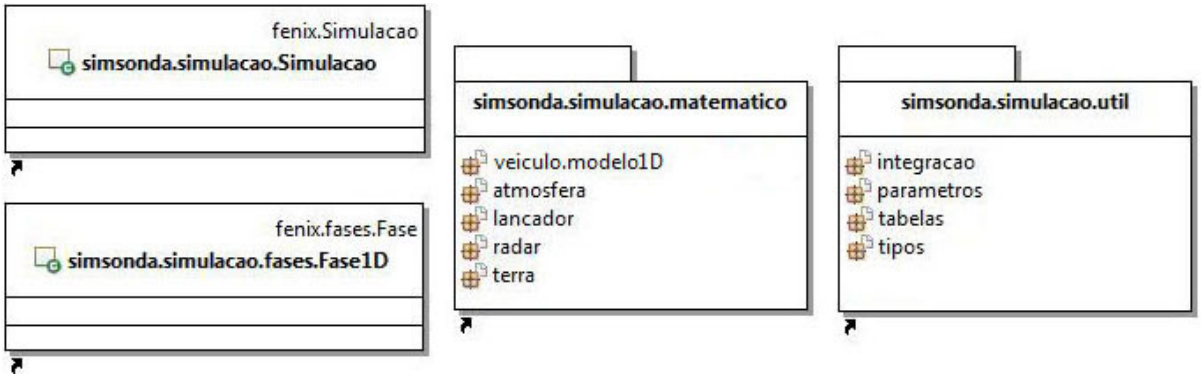

FIGURA 6.2 – Modelo de Simulação 1D

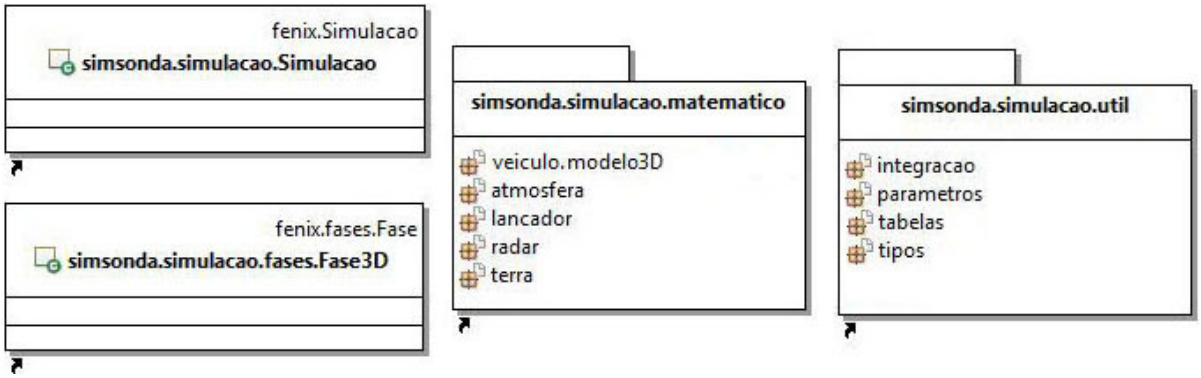

FIGURA 6.3 – Modelo de Simulação 3D

O Modelo de Simulação 6D poderia ser adotado para todas as fases de uma simulação, uma vez que a quantidade de cálculos realizados deixou de ser um empecilho para os processadores e dispositos de armazenamento atuais. Esta possibilidade não foi testada, uma vez que os arquivos de teste adquiridos não possuiam alguns parâmetros e tabelas exclusivos deste modelo, sendo recomendada a realização deste experimento.

FIGURA 6.4 – Modelo de Simulação 6D

Apesar dos Modelos 1D, 3D e 6D compartilharem algumas classes, eles são independentes e podem ser usados para construir simuladores individuais, como ilustrado na Figura 6.5. Dessa forma, as Tabelas 6.5, 6.6 e 6.7 mostram a quantidade de código-fonte dos Modelos de Simulação 1D, 3D e 6D.

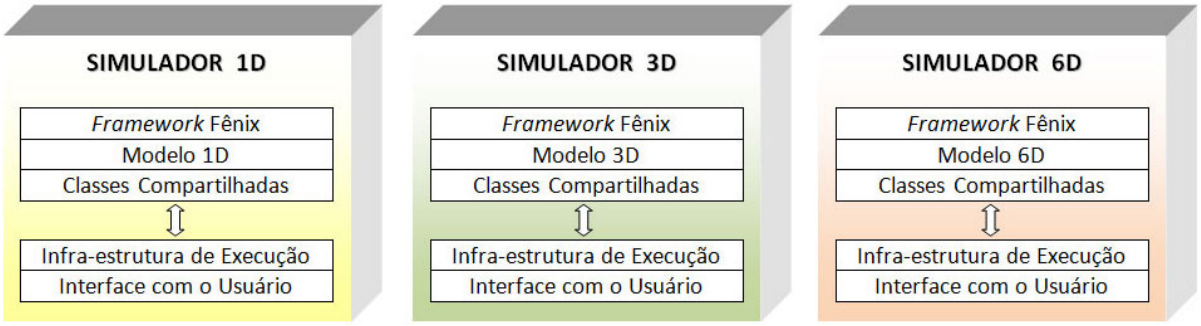

FIGURA 6.5 – Simuladores independentes

Para o Modelo 1D não foram contadas as linhas de código contidas no pacote *sislineares* do pacote *simsonda.simulacao.util*, as classes *Fase3D* e *Fase6D* do pacote *simsonda.simulacao.fases*, e os pacotes *simsonda.simulacao.matematico.veiculo.modelo3D* e *simsonda.simulacao.matematico.veiculo.modelo6D*.

Nos Modelos de Simulação 1D e 3D, não se utilizam alguns parâmetros e tabelas especificos do Modelo 6D. As linhas de código para a manipulação destas informações foram excluídas da contagem apresentada nas Tabelas 6.5 e 6.6.

No Modelo de Simulação 3D não se contou também as linhas contidas no pacote

Nome do Componente	Linhas	%
Fênix	1.652	45
Modelo 1D	2.010	55
Total	3.662	100

TABELA 6.5 – Quantidade de linhas de código para o Modelo de Simulação 1D

sislineares do pacote *simsonda.simulacao.util*, as classes *Fase1D* e *Fase6D* do pacote *simsonda.simulacao.fases*, além dos pacotes *modelo1D* e *modelo6D* contidos no pacote *simsonda.simulacao.matematico.veiculo*.

Nome do Componente	Linhas	%
Fênix	1.652	45
Modelo 3D	2.045	55
Total	3.697	100

TABELA 6.6 – Quantidade de linhas de código para o Modelo de Simulação 3D

Para a contagem das linhas de código do Modelo de Simulação 6D não se considerou as classes *Fase1D* e *Fase6D* do pacote *simsonda.simulacao.fases*, e os pacotes *modelo1D* e *modelo3D* do pacote *simsonda.simulacao.matematico.veiculo*.

Nome do Componente	Linhas	%
Fênix	1.652	38
Modelo 6D	2.655	62
Total	4.307	100

TABELA 6.7 – Quantidade de linhas de código para o Modelo de Simulação 6D

Assim, o *Framework* Fênix proporcionou em média 43% de reusabilidade de código para a construção dos três Modelos de Simulação do Simulador SIMSonda. Da mesma forma como ocorreu para os Modelos 1D, 3D e 6D, o *Framework* Fênix pode ser reusado para o desenvolvimento de outros Modelos de Simulação. Para criar uma simulação da trajetória de um foguete, um simulador precisa implementar o comportamento abstrato definido pelas classes do Fênix, e adicionar comportamento específico quando julgar necessário.

Cabe ressaltar que esta análise dos resultados parte do pressuposto que se pode utilizar

estes modelos para simulações independentes de trajetórias 1D, 3D ou 6D. Além disso, os valores obtidos referem-se aos Modelos de Simulação do Simulador SIMSonda, não sendo generalizados para outros simuladores. Não se garante cerca de 43% de reusabilidade de código na construção de Modelos de Simulação para outros simuladores de trajetórias de foguetes de sondagem.

6.2 Avaliação do Padrão HLA

O padrão HLA consiste numa especificação de como sistemas de *software* podem interagir a fim de realizar uma simulação num ambiente distribuído. O padrão estabelece os serviços que devem ser fornecidos por uma infra-estrutura de execução e as interfaces utilizadas pelos sistemas para se comunicarem, através desta infra-estrutura.

O esforço necessário para o aprendizado do padrão HLA é semelhante ao de qualquer tecnologia para o desenvolvimento de sistemas de computação distribuídos.

Como a infra-estrutura de suporte já estava disponível no *software* pRTI, o padrão proporcionou um componente de *software* que foi adicionado ao Simulador SIMSonda para participar de uma simulação distribuída. O padrão HLA padroniza a comunicação e o sincronismo necessário para a execução de simulações distribuídas, além de permitir que seus participantes ingressem e deixem a simulação a qualquer momento, sendo ideal para simulações semelhantes a jogos.

Neste sentido, torna-se uma opção interessante pois padroniza a interação entre os sistemas envolvidos, permitindo que simuladores implementados com tecnologias diferentes, executando em plataformas diferentes, consigam juntos realizar uma simulação num ambiente de simulação mais amplo. Este grau de interoperabilidade pode ser obtido se

a RTI for desenvolvida com base numa tecnologia para sistemas de computação distribuídos como a especificada para *Common Object Request Broker Architeture - CORBA* (MAHMOUD, 1999).

Recomenda-se o desenvolvimento de uma Infra-estrutura para a Execução de Federações direcionada para o domínio aero-espacial, pois pode facilitar a sua atualização no que se refere a novas tecnologias de sistemas distribuídos, proporcionar um maior controle da execução de Federações, além de garantir maior segurança das informações compartilhadas entre os Federados, sendo este, geralmente, um requisito essencial para aplicações deste domínio.

O padrão HLA estabelece a forma como cada Federado deve interagir com os demais Federados numa Federação. Em termos de esforço para o desenvolvimento de uma simulação distribuída, a diferença em relação a um simulador *stand-alone* refere-se ao ciclo de vida dos Federados na Federação, sendo implementado o sincronismo necessário para o progresso da simulação e o compartilhamento dos dados.

Com o objetivo de avaliar o esforço de desenvolvimento para a adoção do padrão HLA na construção de sistemas de simulação distribuídos, utilizou-se como métrica a contagem de linhas de código. Analisou-se a quantidade de linhas escritas para a construção da Federação para Visualização Remota da Trajetória, descrita no Capítulo 5.

Desenvolveu-se os Federados Coordenador, Visualização e SIMSonda, membros desta Federação de exemplo, no pacote *hla.federacao*. A Tabela 6.8 apresenta a quantidade de linhas de código deste pacote.

Esta quantidade de linhas de código para a criação da Federação corresponde a 24% de todas as linhas escritas no desenvolvimento do Estudo de Caso. Este valor refere-se ao

Nome do Pacote	Linhas	%
util	99	4
coordenador	972	33
visualizacao	971	33
simulador	886	30
Total	2.928	100

TABELA 6.8 – Quantidade de linhas de código para construção da Federação

esforço necessário para o desenvolvimento de toda a Federação.

Analisando as quantidades de linhas para o desenvolvimento de cada Federado, apresentadas na Tabela 6.8, percebe-se que a maior difereça ocorre entre os pacotes *coordenador* e *simulador*, correspondendo a 86 linhas de código. Esta diferença entre os Federados refere-se basicamente as linhas escritas para o desenvolvimento da Interface com o Usuário, uma vez que a interface do Federado SIMSonda foi criada no pacote *simsonda.aplicacao*.

Como discutido no Capitulo 5, o Federado Sistema de Visualização somente recebe os valores das atualizações de atributos publicados pelo Federado Simulador SIMSonda e os exibe na Interface com o Usuário, não realizando praticamente nenhum outro tipo de processamento. O Federado Sistema Coordenador também não realiza nenhum processamento interno muito significativo, sendo responsável basicamente por sincronizar a Federação. A maior parte do código escrito para estes Federados refere-se a implementação do padrão HLA.

A estutura do código-fonte organizada em classes permitiu que a implementação do padrão HLA ocorre-se em módulos separados, como ilustra a Figura 6.6.

As Figuras 6.7, 6.8 e 6.9 apresentam as classes dos pacotes *hla.federacao.simulador*, *hla.federacao.coordenador* e *hla.federacao.visualizacao* que, juntamente com o pacote util, implementam o Módulo HLA, respectivamente, dos Federados SIMSonda, Coordenador e Sistema de Visualização, membros da Federação para Visualização Remota de Trajetórias.

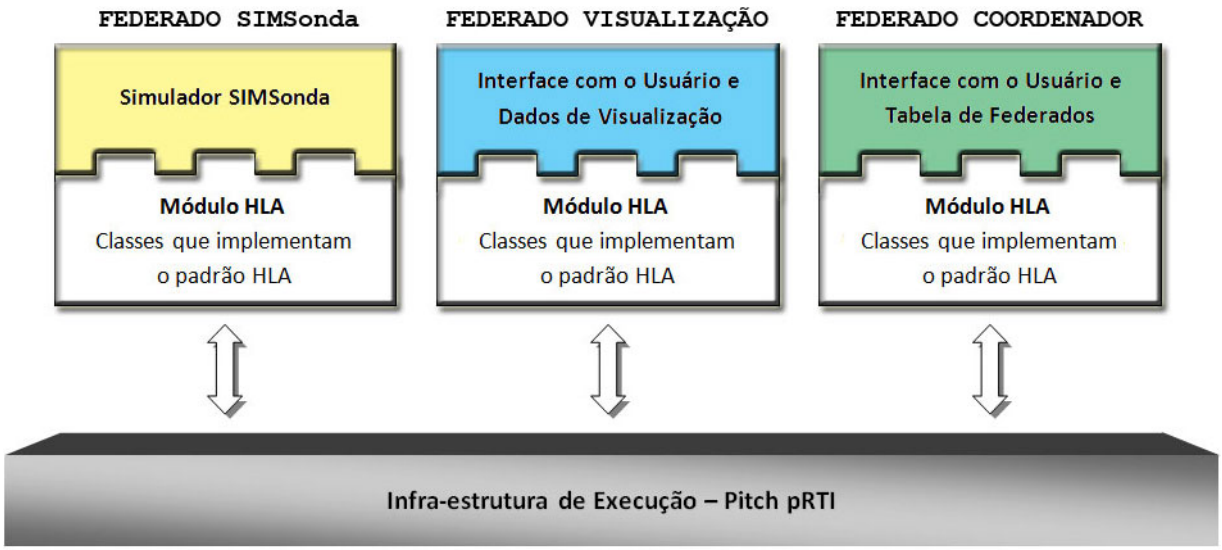

FIGURA 6.6 – Componentes dos Federados

As demais classes destes pacotes, omitidas nestas Figuras, não estão envolvidas com o padrão HLA.

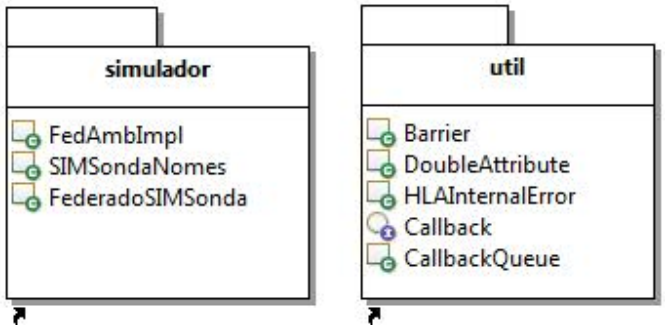

FIGURA 6.7 – Classes do Federado SIMSonda que implementam o padrão HLA

Para construir o Federado SIMSonda, acrescentou-se um Módulo HLA ao Simulador SIMSonda, permitindo sua interação com a RTI. Buscou-se organizar o código deste Federado segundo o paradigma da orientação a objetos, resultando num módulo adicional do Simulador SIMSonda, implementado no pacote *hla.federacao.simulador*.

A implementação do Federado SIMSonda como um módulo independente do Simulador SIMSonda torna-se uma vantagem, permitindo também a utilização da infra-estrutura de suporte para uma simulação *stand-alone*. O padrão HLA não é aconselhável para este tipo de simulação, pois sua implementação torna-se um esforço desnecessário já que foi

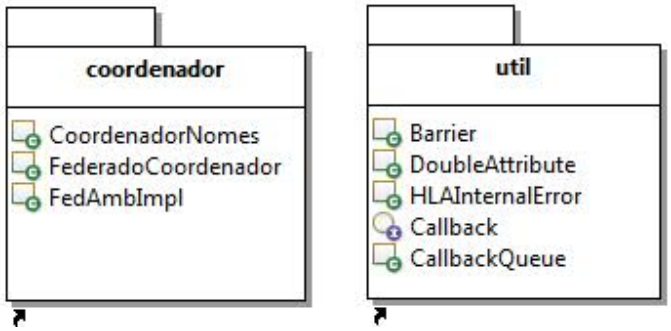

FIGURA 6.8 – Classes do Federado Coordenador que implementam o padrão HLA projetado para a simulação distribuída.

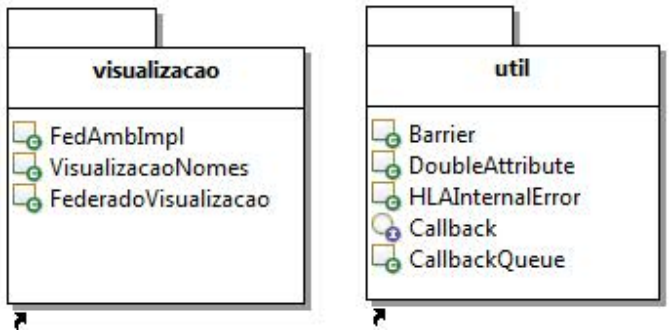

FIGURA 6.9 – Classes do Federado Sistema de Visualização que implementam o padrão HLA

A Tabela 6.9 apresenta a contagem das linhas de código do Módulo HLA dos Federados. Nesta contagem foram excluídas as linhas das classes *FederadoSIMSonda*, *FederadoVisualizacao* e *FederadoCoordenador* que não estão relacionadas ao padrão HLA.

Para o Federado Simulador SIMSonda foram consideradas todas as classes do pacote *hla.federacao.simulador*. Quanto ao Federado Sistema Coordenador considerou-se todas as classes do pacote *hla.federacao.coordenador*, exceto as classes *CoordenadorFrame* e *FederateTable*. Já quanto ao Federado Sistema de Visualização considerou-se todas as classes do pacote *hla.federacao.visualizacao*, excluindo as classes *VisualizacaoFrame* e *Dados*.

O esforço médio para adotar o padrão HLA na simulação distribuída para Visualização Remota da Trajetória, desenvolvida como Estudo de Caso, correspondeu a 728 linhas de código. Este valor refere-se apenas a esta Federação de exemplo, não sendo generali-

Nome do Federado	Linhas	%
SIMSonda	832	38
Coordenador	638	29
Visualização	713	33
Total	2.183	100
Média	728	

TABELA 6.9 – Quantidade de linhas de código para o Módulo HLA

zado para outras Federações ou Federados. Como a implementação do padrão HLA foi realizada como um módulo separado do Simulador SIMSonda, pode-se ainda comparar a adoção deste padrão em relação ao esforço de desenvolvimento de todo o Protótipo, como mostrado na Tabela 6.10.

Nome do Componente	Linhas	%
Simulador SIMSonda	7.715	90
Módulo HLA	832	10
Total	8.547	100

TABELA 6.10 – Quantidade de linhas de código para o Federado SIMSonda

Na Tabela 6.10 considerou-se para a contagem de linhas do Simulador SIMSonda as classes dos pacotes *fenix* e *simsonda*. Dessa forma, a quantidade de linhas necessárias para permitir que o Protótipo do Simulador SIMSonda participe de um ambiente de simulação distribuída corresponde a 10% de todo o código escrito para o Protótipo. Conforme já ressaltado, este valor refere-se apenas a esta Federação de exemplo, não sendo generalizado para outras aplicações com base no padrão HLA.

As linhas de código a serem acrescentadas no desenvolvimento de todas as funcionalidades da interface gráfica com o usuário, numa versão completa do Simulador SIMSonda, irão aumentar a porcentagem de linhas do componente *Simulador SIMSonda*, provocando a redução da porcentagem de linhas referentes ao componente *Módulo HLA*.

Assumindo que a Infra-estrutura de Execução encontra-se disponível, e com base no fato de que o padrão HLA suporta um conjunto limitado de serviços, especificados nas

interfaces *Federate-Ambassador* e *RTI-Ambassador*, torna-se aceitável afirmar que será pequeno o esforço necessário para incorporar um Simulador da proporção e complexidade do SIMSonda numa Federação, comparando-se com o esforço total dispendido no desenvolvimento de todo o Simulador.

Os resultados desta análise podem ser considerados num *benchmark*, para auxiliar na seleção de arquiteturas para o desenvolvimento de sistemas simulação distribuídos. Pode-se adotar, como um dos critérios de comparação das arquiteturas, a quantidade de linhas de código para construir uma aplicação de teste, como a Federação para Visualização Remota da Trajetória.

6.3 Verificação e Validação do Simulador SIMSonda

Para o *Framework* Fênix e o Simulador SIMSonda serem realmente utilizados no Instituto de Aeronáutica e Espaço (IAE), eles devem ter os seus modelos e códigos-fonte completamente verificados por uma equipe de engenheiros, sendo realizados diversos testes para a sua validação, conforme ocorre para sistema de *software* de emprego militar. Esta verificação pode melhor qualificar os resultados do Simulador SIMSonda.

O Protótipo do Simulador SIMSonda foi construído com o intuito de verificar e validar o modelo proposto no *Framework* Fênix, sendo sua implementação uma confirmação de que o *Framework* pode ser utilizado para a construção deste tipos de simuladores. Por outro lado, o Simulador SIMSonda pode ser validado comparando-se seus resultados com os produzidos pelo programa ROSI, usando a mesma base de dados.

Como o ROSI já foi validado para utilização no IAE, os valores produzidos por ele foram considerados como os valores corretos, sendo analisada a discrepância entre estes e

os resultados do SIMSonda.

Para a comparação dos resultados dos programas considerou-se o apogeu da trajetória e o ponto de impacto do foguete com a terra. Realizou-se cinco experimentos com base em cinco arquivos de entrada de exemplo, fornecidos pelo IAE, para quatro foguetes de sondagem diferentes, sendo dois mono-estágio e dois biestágio. Cada arquivo foi submetido aos dois simuladores, sendo os resultados armazenados num arquivo texto no formato usado para a visualização gráfica.

Desenvolveu-se um programa utilitário para a análise dos resultados contidos nos dois arquivos textos. Este programa processa o conteúdo de cada arquivo, armazenando em tabelas os dados de apogeu e de ponto de impacto da trajetória, e produzindo dois gráficos com curvas de altitude e velocidade do foguete.

As Tabelas 6.11, 6.12, 6.13, 6.14 e 6.15 apresentam os dados de tempo (T), posição ($XL,\ YL,\ ZL$), distância radial (*Slant Range - S.R.*) e alcance (*Ground Range - G.RAN*) em relação ao apogeu da trajetória para cada arquivo de entrada. Os valores foram arredondados considerando-se seis casas decimais, sendo segundos (s) a unidade de tempo e quilômetros (km) a unidade das demais informações. Estes dados do apogeu foram calculados para o Sistema de Referência do Lançador (F_L), descrito em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976).

	T	XL	YL	ZL	S.R.	G.RAN
ROSI	135,0	25,089430	26,265760	79,972930	87,835300	36,323130
SIMSonda	135,0	25,089430	26,265760	79,972930	87,835300	36,323130
Discrepância	0,0	0,000000	0,000000	0,000000	0,000000	0,000000
Percentual	0,0	0,000000	0,000000	0,000000	0,000000	0,000000

TABELA 6.11 – Apogeu para o arquivo de teste 1

Na Tabela 6.16 apresenta a discrepância média em relação ao apogeu da trajetória comparando-se os resultados dos Simuladores SIMSonda e ROSI, sendo considerados os

	T	XL	YL	ZL	S.R.	G.RAN
ROSI	403,0	254,780300	0,522039	572,616100	626,739600	254,780800
SIMSonda	403,0	254,779600	0,522002	572,613000	626,736500	254,780200
Discrepância	0,0	0,000700	0,000038	0,003100	0,003100	0,000600
Percentual	0,0	0,000275	0,007203	0,000541	0,000495	0,000235

TABELA 6.12 – Apogeu para o arquivo de teste 2

	T	XL	YL	ZL	S.R.	G.RAN
ROSI	405,0	246,445900	128,211900	565,439700	629,996700	277,801800
SIMSonda	405,0	246,445700	128,211700	565,438700	629,995800	277,801600
Discrepância	0,0	0,000200	0,000200	0,001000	0,000900	0,000200
Percentual	0,0	0,000081	0,000156	0,000177	0,000143	0,000072

TABELA 6.13 – Apogeu para o arquivo de teste 3

	T	XL	YL	ZL	S.R.	G.RAN
ROSI	201,0	75,901120	-6,960391	163,531000	180,421200	76,219600
SIMSonda	202,0	76,144480	-6,969097	163,620600	180,605200	76,462740
Discrepância	-1,0	-0,243360	0,008706	-0,089600	-0,184000	-0,243140
Percentual	-0,5	-0,320628	-0,125079	-0,054791	-0,101984	-0,318999

TABELA 6.14 – Apogeu para o arquivo de teste 4

	T	XL	YL	ZL	S.R.	G.RAN
ROSI	260,0	64,961750	72,953090	253,007600	271,210300	97,684100
SIMSonda	260,0	65,148120	72,985620	252,752500	271,025900	97,832400
Discrepância	0,0	-0,186370	-0,032530	0,255100	0,184400	-0,148300
Percentual	0,0	-0,286892	-0,044590	0,100827	0,067992	-0,151816

TABELA 6.15 – Apogeu para o arquivo de teste 5

valores absolutos das diferenças encontradas nos cinco arquivos de exemplo.

Arquivo	T (s)	XL (km)	YL (km)	ZL (km)	S.R. (km)	G.RAN (km)
1	0,0	0,000000	0,000000	0,000000	0,000000	0,000000
2	0,0	0,000700	0,000038	0,003100	0,003100	0,000600
3	0,0	0.000200	0.000200	0.001000	0.000900	0,000200
4	1,0	0,243360	0,008706	0,089600	0,184000	0,243140
5	0,0	0,186370	0,032530	0,255100	0,184400	0,148300
Média	0,2	0,086126	0,008295	0,069760	0,074480	0,078448

TABELA 6.16 – Discrepância média para o apogeu da trajetória

Da mesma forma, as Tabelas 6.17, 6.18, 6.19, 6.20 e 6.21 apresentam para os cinco arquivos de entrada os dados de tempo (T), posição ($XL,\ YL,\ ZL$), azimute (AZI) e alcance ($G.RAN$) do ponto de impacto do foguete com a superfície da terra em relação

ao Sistema F_L. Estes valores também foram arredondados considerando-se seis casas decimais, sendo segundos (s) a unidade de tempo, quilômetros (km) a unidade de posição e alcance e graus a unidade de azimute.

	T	XL	YL	ZL	AZI	G.RAN
ROSI	275,0	49,693650	52,072990	0,544050	43,660000	71,979540
SIMSonda	275,0	49,693650	52,072990	0,544050	43,660000	71,979540
Discrepância	0,0	0,000000	0,000000	0,000000	0,000000	0,000000
Percentual	0,0	0,000000	0,000000	0,000000	0,000000	0,000000

TABELA 6.17 – Ponto de impacto para o arquivo de teste 1

	T	XL	YL	ZL	AZI	G.RAN
ROSI	778,0	468,947200	2,330076	19,625130	89,720000	468,953000
SIMSonda	778,0	468,945500	2,329995	19,627930	89,720000	468,951300
Discrepânci	0,0	0,001700	0,000081	-0,002800	0,000000	0,001700
Percentual	0,0	0,000363	0,003476	-0,014267	0,000000	0,000363

TABELA 6.18 – Ponto de impacto para o arquivo de teste 2

	T	XL	YL	ZL	AZI	G.RAN
ROSI	775,0	452,343600	235,603600	23,570950	62,490000	468,953000
SIMSonda	775,0	452,343300	235,603300	23,571730	62,490000	510,022900
Discrepância	0,0	0,000300	0,000300	-0,000780	0,000000	0,000400
Percentual	0,0	0,000066	0,000127	-0,003309	0,000000	0,000078

TABELA 6.19 – Ponto de impacto para o arquivo de teste 3

	T	XL	YL	ZL	AZI	G.RAN
ROSI	396,0	148,245800	-13,532750	2,070873	95,220000	148,862200
SIMSonda	396,0	148,251600	-13,500870	1,837526	95,200000	148,865100
Discrepância	0,0	-0,005800	-0,031880	0,233347	0,020000	-0,002900
Percentual	0,0	-0,003912	0,235577	11,268050	0,021004	-0,001948

TABELA 6.20 – Ponto de impacto para o arquivo de teste 4

	T	XL	YL	ZL	AZI	G.RAN
ROSI	498,0	124,887200	140,347400	2,983611	41,660000	187,867500
SIMSonda	498,0	125,278300	140,393100	3,275346	41,740000	188,161800
Discrepância	0,0	-0,391100	-0,045700	-0,291735	-0,080000	-0,294300
Percentual	0,0	-0,313163	-0,032562	-9,777917	-0,192031	-0,156653

TABELA 6.21 – Ponto de impacto para o arquivo de teste 5

Na Tabela 6.22, para o cálculo da discrepância média do ponto de impacto, considerou-se os valores absolutos das diferenças obtidas nos cinco experimentos.

Arquivo	T (s)	XL (km)	YL (km)	ZL (km)	AZI (grau)	G.RAN (km)
1	0,0	0,000000	0,000000	0,000000	0,000000	0,000000
2	0,0	0,001700	0,000081	0,002800	0,000000	0,001700
3	0,0	0,000300	0,000300	0,000780	0,000000	0,000400
4	0,0	0,005800	0,031880	0,233347	0,020000	0,002900
5	0,0	0,391100	0,045700	0,291735	0,080000	0,294300
Média	0,0	0,079780	0,015592	0,105732	0,020000	0,059860

TABELA 6.22 – Discrepância média para o ponto de impacto com a terra

Além da análise dos dados de apogeu e ponto de impacto, comparou-se a altitude e a velocidade do foguete, no Sistema F_L, em função do tempo de vôo, para cada arquivo de teste através dos gráficos apresentados nas Figuras 6.10, 6.11, 6.12, 6.13, 6.14. De acordo com a legenda destes gráficos, o Arquivo 1 corresponde ao arquivo produzido pelo programa ROSI, enquanto que o Arquivo 2 refere-se ao arquivo gerado pelo Simulador SIMSonda.

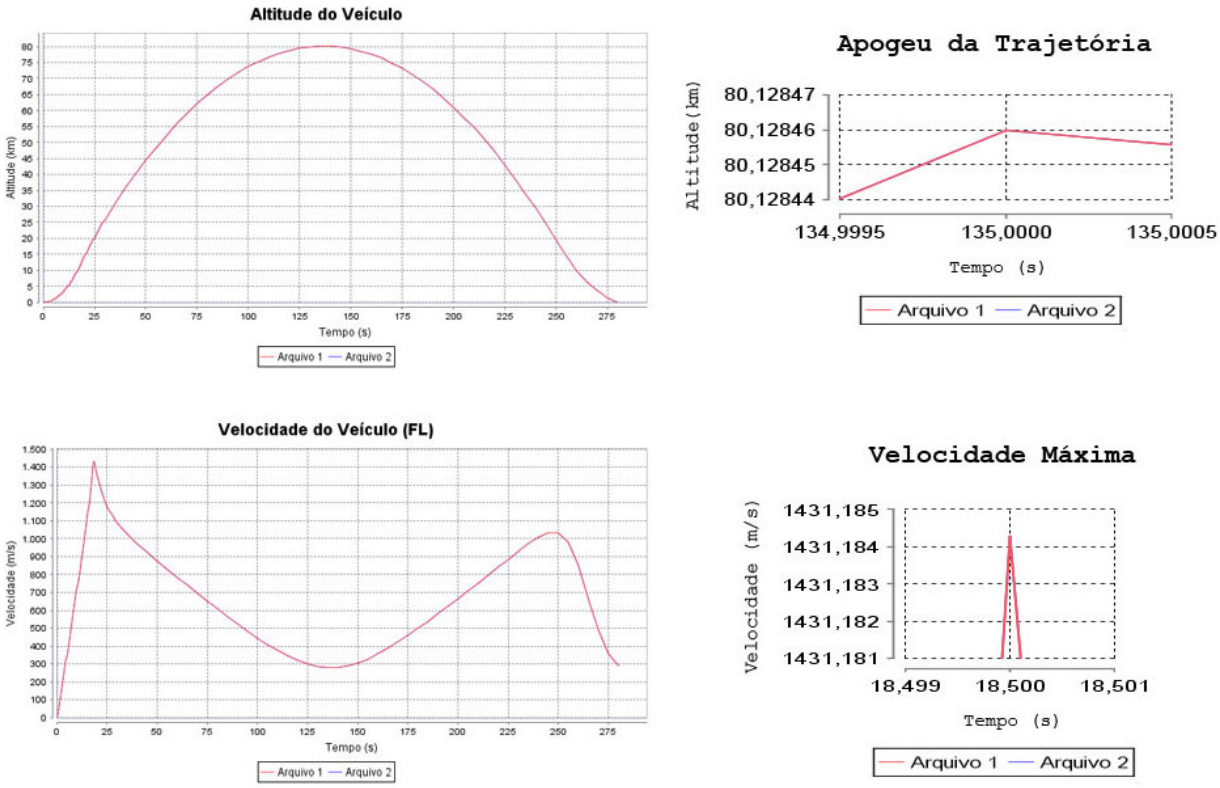

FIGURA 6.10 – Gráficos de altitude e velocidade do foguete para o arquivo 1

Os resultados produzidos pelo Simulador SIMSonda se aproximaram de tal forma dos

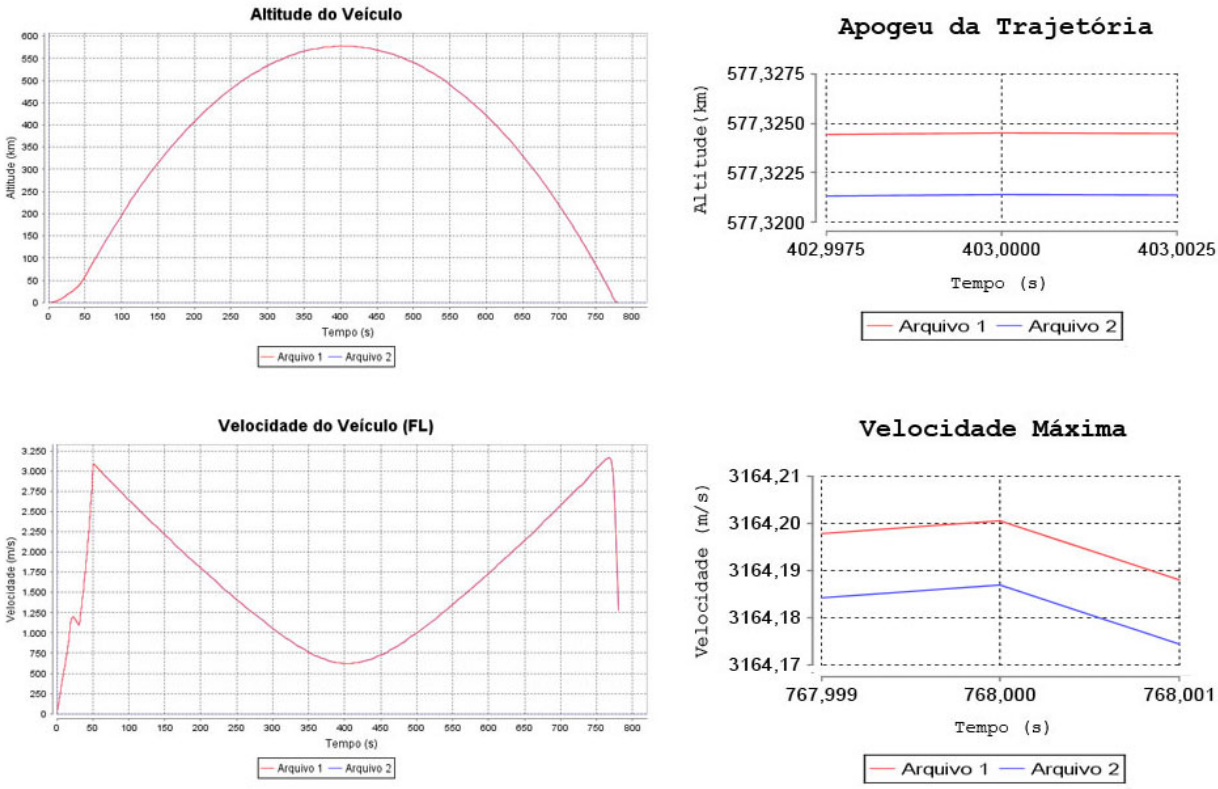

FIGURA 6.11 – Gráfico de altitude e velocidade do foguete para o arquivo 2

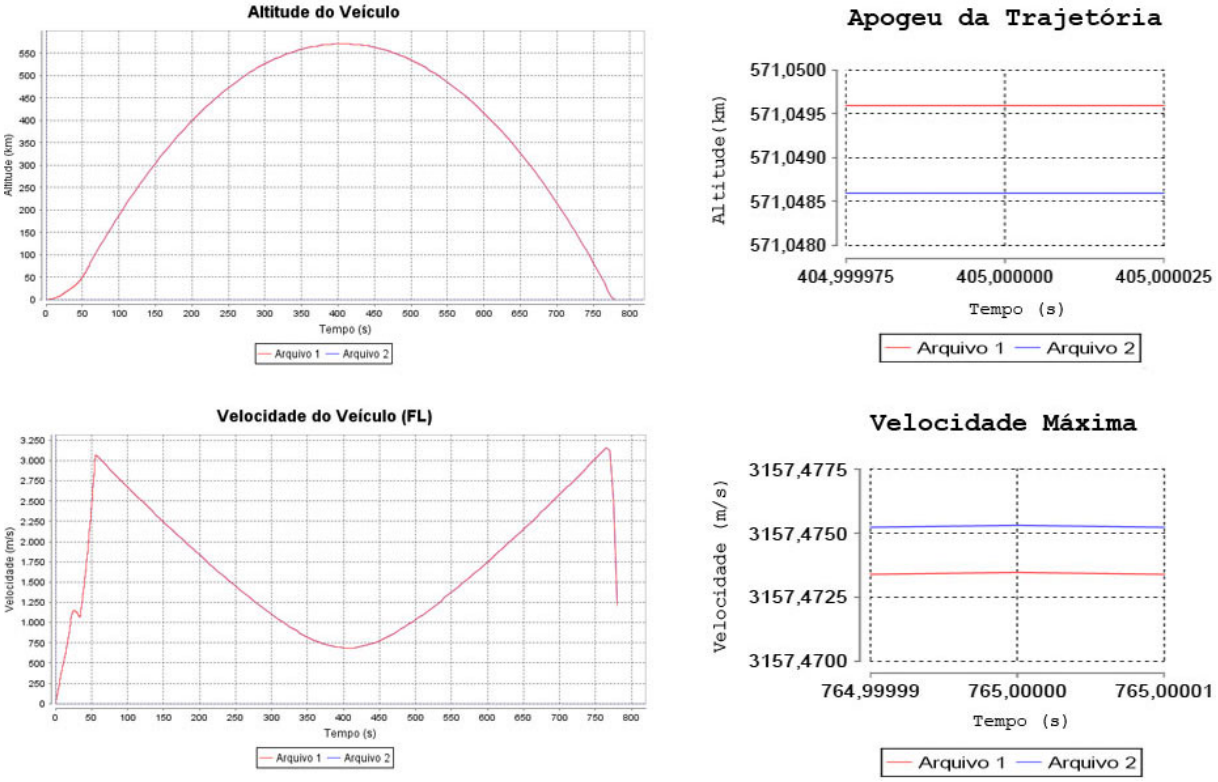

FIGURA 6.12 – Gráfico de altitude e velocidade do foguete para o arquivo 3

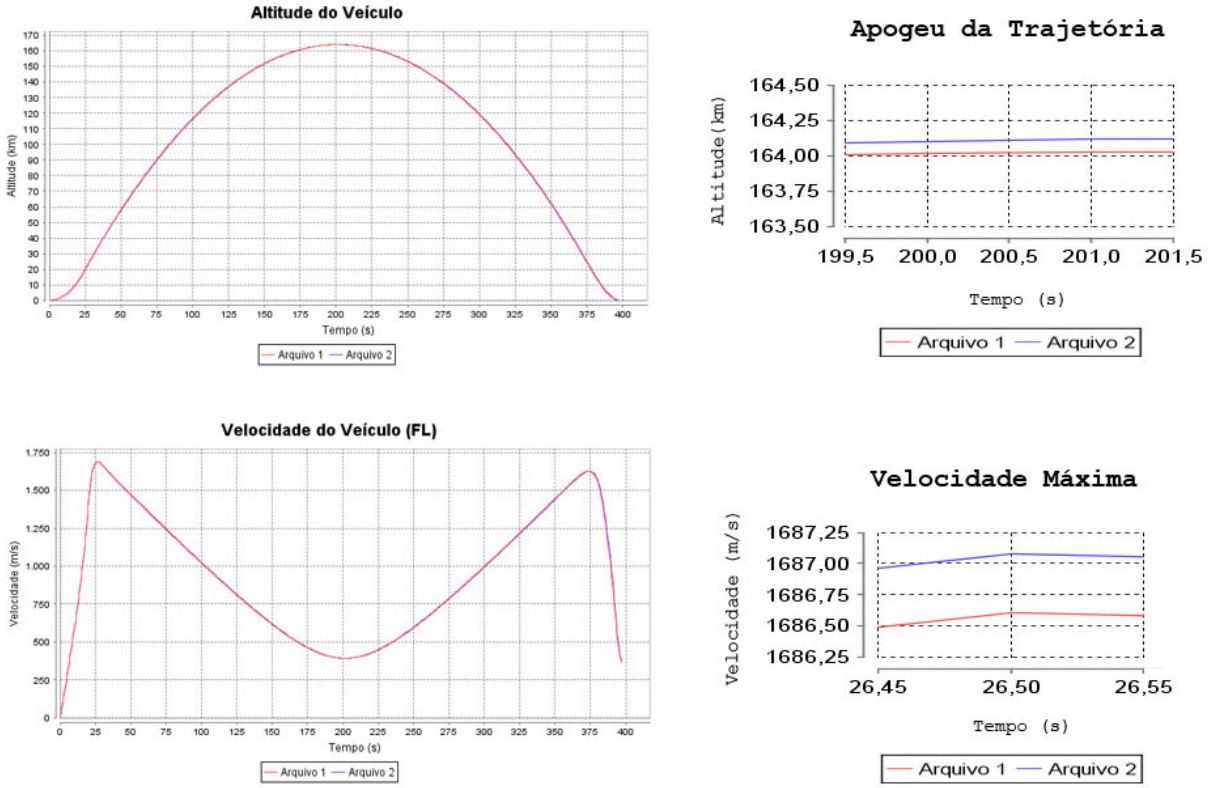

FIGURA 6.13 – Gráfico de altitude e velocidade do foguete para o arquivo 4

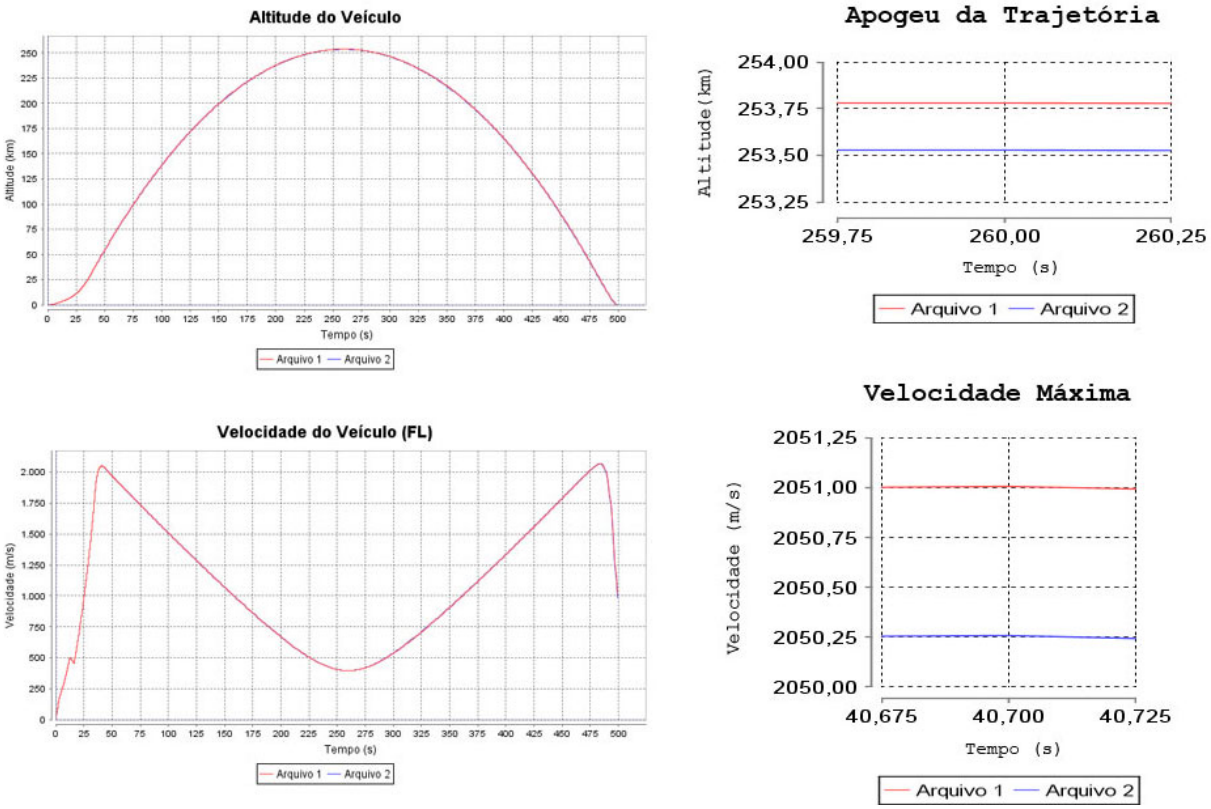

FIGURA 6.14 – Gráfico de altitude e velocidade do foguete para o arquivo 5

resultados obtidos no programa ROSI, que as curvas de altitude e velocidade apresentam-se praticamente sobrepostas para os cinco arquivos de exemplo. Por esta razão, apresentou-se o apogeu da trajetória e a velocidade máxima do foguete utilizando uma escala adequada.

Com base nas discrepâncias médias encontradas para apogeu e ponto de impacto e nos gráficos de altitude e velocidade, foram considerados significativos os resultados produzidos pelo Protótipo do Simulador SIMSonda. Constatou-se que a qualidade dos resultados foi superior a alcançada por Louis (2006), no desenvolvimento do *software* jRosi, comparando os valores das discrepâncias obtidas.

O Simulador SIMSonda mostrou-se robusto e confiável em todos os testes realizados, sendo seus resultados consistentes nas plataformas utilizadas. Constatou-se, experimentalmente, que o compilador da linguagem FORTRAN produziu resultados com qualidade superior comparados aos produzidos pelo compilador da linguagem Java. Entretanto, as diferenças observadas foram consideradas desprezíveis.

Constatou-se nesta análise dos resultados produzidos pelo Simulador SIMSonda e pelo programa ROSI, que a utilização de Processos de Desenvolvimento de *Software*, conforme apresentados nos Capítulos 4 e 5, proporciona um produto de *software* de qualidade.

Com relação a análise dos resultados obtidos quanto a reusabilidade do *Framework* Fênix e a avaliação da construção da Federação de Visualização Remota de Trajetórias, verificou-se as vantagens de utilização de *frameworks* OO e do padrão HLA, discutidas no Capítulo 2.

Para finalizar, o Capítulo 7 apresenta as considerações finais desta pesquisa, bem como recomendações do autor e sugestões de trabalhos futuros.

7 Conclusão

7.1 Considerações Finais

Para facilitar a construção, a manutenção e a adaptação de simuladores de trajetórias de foguetes e reduzir o desperdício dos recursos envolvidos nessas atividades, este projeto de pesquisa propôs um *Framework* para a Simulação de Trajetórias de Foguetes de Sondagem, denominado Fênix.

Este *Framework* possibilita o reuso de código dos Modelos de Simulação de Simuladores, definindo o modelo das entidades envolvidas, e a estrutura e o fluxo para execução das fases de uma simulação. Neste trabalho, obteve-se uma taxa média de 43% de reusabilidade na criação de Modelos de Simulação, demonstrando que sua utilização torna-se viável para a construção de novos modelos de simuladores.

Além do *Framework* Fênix, também foi idealizada uma Arquitetura de *Sofware* para um Simulador de Trajetória de Foguetes. Esta arquitetura permitiu o desenvolvimento de um Protótipo de Simulador de Trajetórias de Foguetes de Sondagem, chamado SIMSonda, como um Estudo de Caso, para demonstrar como o *Framework* Fênix pode ser reutilizado na criação dos simuladores de trajetórias. A documentação disponibilizada irá fornecer o entendimento necessário para o desenvolvimento de novos Modelos de Simulação.

Constatou-se, durante o desenvolvimento do Protótipo do SIMSonda, que realmente os *frameworks*, além de possibilitarem o reuso de código, também proporcionam o reuso de análise e projeto. Diversos artefatos produzidos para o *Framework* Fênix foram reusados no processo de desenvolvimento do Simulador SIMSonda.

O Protótipo construído mostrou como uma interface mais amigável com o usuário pode facilitar o trabalho dos profissionais envolvidos na definição de uma trajetória nominal. As opções de entrada e saída mais amigáveis permitiram compreender melhor a estrutura de fases da simulação e facilitaram a interpretação dos resultados. Nos testes realizados, os resultados foram muito próximos dos resultados produzidos pelo programa ROSI, já validado pelo IAE, sendo que uma verificação mais detalhada do modelo matemático implementado pode qualificar mais ainda estes resultados.

Neste trabalho também foi avaliada a adoção do padrão *IEEE STD 1516 - High Level Architecture*. Em termos de esforço de aprendizado, o padrão possui a mesma complexidade de outras tecnologias para sistemas de computação distribuídos.

Quanto ao esforço de implementação, verificou-se que a quantidade de linhas de código para criar um Federado corresponde a uma pequena parcela de seu tamanho, quando refere-se a sistemas com modelos mais complexos que o Simulador SIMSonda. Foram necessárias em média 728 linhas de código para incorporar cada Federado na Federação para Visualização Remota da Trajetória, desenvolvida como Estudo De Caso. Para o Federado Simulador SIMSonda, isto correspondeu a um acréscimo de 10% na quantidade total de linhas de código escritas.

Com o desenvolvimento do Estudo de Caso, constatou-se que o padrão HLA pode ser implementado como um componente independente nos simuladores ou aplicativos que integram uma simulação distribuída. Assim, os diversos simuladores envolvidos em cam-

panhas de lançamento de foguetes, desenvolvidos em versões *stand-alone*, podem ser estendidos com poucas linhas de código para tornar-se membros de Federações.

7.2 Contribuições

As principais contribuições deste trabalho para a comunidade acadêmica referem-se a Arquitetura de Simuladores de Trajetórias de Foguetes, e o Framework Fênix, concebidos no Capítulo 4. A arquitetura define a estrutura dos simuladores dividida nos seguintes módulos: Modelo de Simulação, Infra-estrutura de Execução e Interface com o Usuário. O Framework *Fênix* define um modelo abstrato para a Simulação de Trajetórias de Foguetes de Sondagem, correspodendo ao Módulo do Modelo de Simulação.

O *Framework* Fênix, juntamente com esta Arquitetura de *Software*, proporcionam reuso de análise porque descrevem os elementos de importância numa simulação de trajetória de foguete, os relacionamentos entre estes elementos e como um simulador de trajetórias pode ser dividido em módulos menores. Eles proporcionam reuso do projeto porque, para a construção do Módulo do Modelo de Simulação, desenvolveu-se algoritmos abstratos e definiu-se as interfaces e os obstáculos que uma simulador de trajetórias específico precisa satisfazer.

Para a comunidade tecnológica, o *Framework* Fênix também proporciona reuso de código, correspondendo a 1.652 linhas de código que podem ser reutilizadas no desenvolvimento de novos modelos de simulação dos simuladores de trajetórias de foguetes de sondagem. Este trabalho mostrou como aplicar e avaliar a utilização de *Frameworks* Orientados a Objetos (OO) e do Padrão *High Level Archicteture (HLA)*, num Estudo de Caso, direcionado ao setor aeroespacial brasileiro.

Os pesquisadores envolvidos com a simulações de foguetes de sondagem podem utilizar o Protótipo de Simulador de Trajetórias de Foguetes de Sondagem, denominado SIMSonda, como uma ferramenta de auxílio em suas pesquisas, abrangendo modelos matemáticos considerando um, três e seis graus de liberdade para o movimento dos foguetes.

Este trabalho também mostrou como aplicar Processos da Engenharia de *Software* na construção de um *Framework* e um Simulador, específicos para o setor aeroespacial, produzindo artefatos como modelos de requisitos, de casos de uso e de classes, além de diagramas de seqüência e código-fonte documentado.

7.3 Recomendações

O autor recomenda que o Instituto de Aeronáutica e Espaço (IAE) realize as atividades de verificação e validação do *Framework* Fênix, para ser reusado na construção de simuladores aplicados às pesquisa envolvendo trajetórias espaciais.

É aconselhável a utilização da técnica de Orientação a Objetos (OO), incluindo *frameworks*, no desenvolvimento de simuladores espaciais para emprego no IAE, uma vez que a OO possibilita o reuso de componentes e facilita a compreensão e a manutenção de sistemas de *software*.

Recomenda-se também a aplicação de outras técnicas, métodos e ferramentas já consagrados da Engenharia de Software, para documentar todo o processo de desenvolvimento, conforme demonstrado neste trabalho, proporcionando uma melhor compreensão do modelo construído e resultando num produto de melhor qualidade.

A adoção do padrão HLA na criação de sistemas de simulação distribuídos torna-se interessante, pois este padrão define uma arquitetura que padroniza a comunicação e o

sincronismo necessário para a execução de simulações distribuídas.

Recomenda-se uma implementação da *Runtime Infrastructure* definida no padrão HLA, utilizando tecnologias para sistemas de computação distribuídos, como CORBA, que possibilitem um alto grau de interoperabilidade entre os sistemas.

7.4 Propostas de Trabalhos Futuros

Como propostas para a continuação deste trabalho, sugere-se:

1. Realizar a evolução do *Framework* Fênix, considerando os mecanismos de controle usados nos veículos espaciais, para permitir a simulação de foguetes como o Veículo Lançador de Satélites (VLS);

2. Integrar o Simulador SIMSonda com o Ambiente de Visualização da Evolução da Trajetória de Foguetes desenvolvido por Silva (2007), utilizando o padrão HLA, para propiciar a visualização em três dimensões de trajetórias simuladas remotamente;

3. Desenvolver uma interface de comunicação para o teste integrado do sistema embarcado de um veículo controlado sob diversas situações de vôo; e

4. Desenvolver um ambiente de simulação para o treinamento de equipes responsáveis pelas operações de lançamento, envolvendo contagens regressivas e rastreios simulados.

Referências Bibliográficas

AGÊNCIA ESPACIAL BRASILEIRA. **Programa nacional de atividades espaciais**. Brasília: AEB, 2005. Disponível em: <http://www.aeb.gov.br/area/download/pnae_web.pdf>. Acesso em: 12 nov. 2007.

AGÊNCIA ESPACIAL BRASILEIRA. **Foguetes de sondagem**. Brasília: AEB, 2007. Disponível em: <http://www.aeb.gov.br/conteudo.php?ida=23&idc=124>. Acesso em: 12 nov. 2007.

ANDRADÓTTIR, S. Simulation optimization. In: BANKS, J. **Handbook of simulation**. New York: Wiley-Interscience and Engineering & Management Press, 1998. Cap.9. p.307–334.

BANKS, J. Principles of simulation. In: BANKS, J. **Handbook of simulation**. New York: Wiley-Interscience and Engineering & Management Press, 1998. Cap.1. p.3–30.

BOSCH, J. et al. Obstacles in object-oriented framework-based software development. In: SYMPOSIA ON OBJECT ORIENTED APPLICATION FRAMEWORKS, 1998. **Proceedings...** [S. l.: s. n.], 1998.

BUDD, T. A. **An introduction to object-oriented programming**. 3.ed. Boston: Addison-Wesley, 2002. 648p.

BUSS, A.; JACKSON, L. Distributed simulation modeling: A comparison of HLA, CORBA, and RMI. In: WINTER SIMULATION CONFERENCE, 1998, Washington. **Proceedings...**. Washington: [s. n.], 1998. p.819–825.

CENTRO DE COMPUTAÇÃO DA UNIVERSIDADE ESTADUAL DE CAMPINAS. **Frameworks orientados a objetos**. Campinas: CCUEC/UNICAMP, 1998. Informativo técnico n. 52. Disponível em: <http://www.ccuec.unicamp.br/revista/infotec/informacao/informativo.html>. Acesso em: 12 nov. 2007.

CENTRO DE COMPUTAÇÃO DA UNIVERSIDADE ESTADUAL DE CAMPINAS. **Orientação a objetos**: da teoria à prática em Java. Campinas: CCUEC/UNICAMP, 1999. Disponível em: <http://www.micropic.com.br/noronha/Informatica/PD/JAVA/Orientação a objetos - da teoria a prática em java .pdf> Acesso em: 12 nov. 2007.

CENTRO DE LANÇAMENTO DE ALCÂNTARA. **Campanhas**. Alcântara: CLA, 2007. Disponível em: <http://www.cla.aer.mil.br>. Acesso em: 12 nov. 2007.

DEUTSCH, L. P. Design reuse and frameworks in the smalltalk-80 system. In: BIGGERSTAFF, T. J.; PERLIS, A. J. **Software reusability**: applications and experience. New York: Addison-Wesley, 1989. v.2, p.57–71.

GUEDES, G. T. A. **UML 2**: guia de consulta rápida. 2.ed. São Paulo: Novatec Editora, 2005. 109p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE STD 1516**: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). Framework and Rules. New York: IEEE, 2000a. 22p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE STD 1516.1**: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). Federate Interface Specification. New York: IEEE, 2000b. 467p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE STD 1516.2**: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). Object Model Template (OMT) Specification. New York: IEEE, 2000c. 130p.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE STD 1516.3**: Recommended Practice for High Level Architecture (HLA). Federation Development and Execution Process (FEDEP). New York: IEEE, 2003. 32p.

INSTITUTO DE AERONÁUTICA E ESPAÇO. **Especificação e concepção do sistema de controle do VLS**. São José dos Campos: IAE, 1993.

INSTITUTO DE AERONÁUTICA E ESPAÇO. **Código-fonte do programa ROcket SImulation - ROSI**. São José dos Campos: IAE, 2005.

INSTITUTO DE AERONÁUTICA E ESPAÇO. **Benefícios gerados pelo desenvolvimento de foguetes de sondagem**. São José dos Campos: IAE, 2007. Disponível em: <http://www.iae.cta.br/FoguetesdeSondagem-/foguetesdesondagem_importanciaparapais.htm>. Acesso em: 12 nov. 2007.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The unified software development process**. Massachusetts: Addison-Wesley, 1999. 512p.

JAVAWORLD.COM. **CloseAndMaxTabbedPane**: an enhanced JTabbedPane. Massachusetts, 2004. Disponível em: <http://www.javaworld.com/javaworld/jw-09-2004/jw-0906-tabbedpane.html>. Acesso em: 12 nov. 2007.

JFREE.ORG. **JFreeChart**: a free Java chart library. United Kingdom, 2000. Disponível em: <http://www.jfree.org/jfreechart/>. Acesso em: 12 nov. 2007.

JOHNSON, R. E. How to design frameworks. In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, 8., 1993, Washington. **Proceedings...** Washington: [s. n.], 1993.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. **Journal of Object-Oriented Programming**, v.1, n.2, p.22–35, 1988.

KOMPETENZZENTRUM HLA. **Über HLA**. Magdeburg, 2003. Disponível em: <http://www.kompetenzzentrum-hla.de/modules-.php?name=Content&pa=showpage&pid=89&index=1>. Acesso em: 12 nov. 2007.

KRAMER, H. J.; CRAUBNER, A.; ZIEGLTRUM, W. Analysis and specification of trajectory program ROSI. In: WORKSHOP ON SOUNDING ROCKETS, 2., 1976, São José dos Campos. **Proceedings...** São José dos Campos: CTA/DFVLR, 1976.

KUHL, F.; WEATHERLY, R.; DAHMANN, J. **Creating computer simulation systems**: an introduction to the High Level Architecture. New Jersey: Prentice-Hall, 1999. 224p.

LANDIN, N.; NIKLASSON, A. **Development of object-oriented frameworks**. 146f. 1995. Dissertação (Mestrado) — Lund University, Department of Communication Systems, Sweden, 1995.

LOUIS, J. E. **Monitoração da trajetória em tempo-real de veículos espaciais**: suporte na tomada de decisão pela segurança de vôo. 135f. 2006. Dissertação (Mestrado) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2006.

MACKULAK, G. T.; COCHRAN, J. K. The generic-specific modeling approach: an application of artificial intelligence to simulation. In: IIE INTEGRATED SYSTEMS CONFERENCE & SOCIETY FOR INTEGRATED MANUFACTURING CONFERENCE, 1990, San Antonio. **Proceedings...** San Antonio: [s. n.], 1990. p.82–87.

MAHMOUD, Q. H. **Distributed programming with Java**. Boston: Manning Publications, 1999. 320p.

MATTSSON, M. **Object-oriented frameworks**: a survey of methodological issues. 130f. 1996. Tese (Licenciatura) — Lund University, Department of Computer Science, Sweden, 1996.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Jama**: a Java matrix package. Gaithersburg, 2005. Disponível em: <http://math.nist.gov/javanumerics/jama/>. Acesso em: 12 nov. 2007.

NORDSIECK, A. On the numerical integration of ordinary differential equations. In: **Mathematics of Computation**, v.16, n.77, p.22–49. 1962.

PEREIRA, F. C. V.; YAMANAKA, S. S. C. **Validação do programa ROSI 2005**. São José dos Campos: IAE, 2006.

PIDD, M. **Computer simulation in management science**. 3.ed. New York: John Wiley, 1997. 351p.

PITCH TECHNOLOGIES. **Pitch pRTI 1.3**. Sweden, 2005. Disponível em: <http://www.pitch.se/products/hla-13-products/pitch-prti-13.html>. Acesso em: 12 nov. 2007.

PRESSMAN, R. S. **Engenharia de software**. 5.ed. Rio de Janeiro: McGraw-Hill, 2002. 843p.

REID, M. R. An evaluation of the High Level Architecture (HLA) as a framework for NASA modeling and simulation. In: NASA SOFTWARE ENGINEERING WORKSHOP, 25., 2000, Greenbelt. **Proceedings...** Maryland: NASA, 2000.

RESENDE, A. R. M. L. **Um modelo de processo para seleção de componentes de software**. 219f. 2006. Dissertação (Mestrado) — Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, 2006.

RIBEIRO, T. da S. The brazilian space program sounding rockets and satellite launching vehicles. In: THE UNITED NATIONS REGIONAL MEETING ON SPACE TECHNOLOGY AND APPLICATIONS FOR DEVELOPMENT, 1998, Concepcíon. **Proceedings...** Chile: [s. n.], 1998.

SHANNON, R. E. **Systems simulation**: the art and science. New Jersey: Prentice-Hall, 1975. 368p.

SHAW, M.; GARLAN, D. **Software architecture**: perspectives on an emerging discipline. New Jersey: Prentice-Hall, 1996. 242p.

SILVA, G. B. **Ambiente de visualização da evolução da trajetória de veículos espaciais brasileiros**. 119f. 2007. Dissertação (Mestrado) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2007.

SILVA, R. P.; PRICE, R. T. O uso de técnicas de modelagem no projeto de frameworks orientados a objetos. In: ARGENTINE SYMPOSIUM ON OBJECT ORIENTATION, 1., 1997, Buenos Aires. **Proceedings...** Argentina: Sadio, 1997. p.87–94.

SOMMERVILLE, I. **Engenharia de software**. 6.ed. São Paulo: Addison-Wesley, 2003. 592p.

STEELE, M. J. et al. Generic simulation models of reusable launch vehicles. In: WINTER SIMULATION CONFERENCE, 2002, San Diego. **Proceedings...** California: [s. n.], 2002. p.747–753.

TOLK, A. HLA-OMT versus traditional data and object modeling. In: COMMAND AND CONTROL RESEARCH AND TECHNOLOGY SYMPOSIUM. 2001, Annapolis. **Proceedings...** Maryland: [s. n.], 2001.

TRIVELATO, G. C. **Simulação distribuída**: o que é potencial da “HLA” e “Web Simulation”. São José dos Campos: INPE, 2003. (INPE 9664-NTC/357).

WILSON, D. A.; WILSON, S. D. Writing frameworks - capturing your expertise about a problem domain. In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS, 8., 1993, Washington. **Proceedings...** Washington: [s. n.], 1993.

Apêndice A - Requisitos do *Framework* Fênix

A.1 Introdução

Esta especificação descreve os requisitos do *Framework* para a Simulação de Trajetórias de Veículos Espaciais de Sondagem, denominado Fênix, incluindo seus Requisitos Funcionais (RF) e Não-Funcionais (RNF), suas restrições e outros fatores necessários à sua completa descrição. Utilizou-se as siglas FEN-RF, FEN-RNF e FEN-RS, seguidas de uma numeração, para identificar, respectivamente, cada Requisito Funcional, Não-Funcional, e Restrição do Fênix.

A.2 Definição da Solução Proposta

Este Projeto de Pesquisa tem como objetivo desenvolver um *Framework* para a Simulação de Trajetórias de Veículos Espaciais de Sondagem, a fim de facilitar a construção, a manutenção e a adaptação de aplicações envolvendo trajetórias espaciais e reduzir o desperdício dos recursos envolvidos nessas atividades.

A.3 Escopo

Este documento refere-se ao levantamento de requisitos do *Framework* Fênix e se aplica a todas as fases e atividades do seu desenvolvimento.

A.4 Visão Geral do Sistema

O *Framework* Fênix irá prover a simulação de trajetórias de foguete de sondagem, considerando os modelos abstratos das entidades envolvidas e a estrutura da simulação, dividida em fases abstratas caracterizadas pela a ocorrência de eventos no vôo. Os requisitos do *Framework* foram especificados com base na Arquitetura de *Software* idealizada na Seção 4.1, do Capítulo 4.

A.5 Lista de Requisitos

As Seções A.5.1 e A.5.2 apresentam os Requisitos Funcionais (RF) e Não-Funcionais (RNF) especificados para o *Framework* Fênix.

A.5.1 Requisitos Funcionais (RF)

Quanto ao Módulo do Modelo Matemático dos simuladores de trajetórias, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RF01 - Criação de modelos representando as entidades envolvidas: terra, atmosfera terrestre, lançador, foguete e radar; e

- FEN-RF02 - Cálculo do movimento do veículo espacial considerando diferentes graus de liberdade.

Quanto ao Módulo do Modelo de Fases da simulação de lançamento, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RF03 - Divisão em fases de vôo, caracterizando eventos como: descontinuidade de parâmetros de vôo ou liberação de estágios; e
- FEN-RF04 - Gerenciamento de execução da simulação, possibilitando o controle do tempo de vôo e a mudança das fases para permitir o progresso da simulação.

A.5.2 Requisitos Não-Funcionais (RNF)

Quanto à qualidade, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RNF01 - Criação de modelos no formato definido pela Linguagem de Modelagem Unificada (*Unified Modeling Language - UML)*, utilizando ferramentas da Engenharia de *Software* Ajudada por Computador (*Computer-Aided Software Engineering - CASE)*; e
- FEN-RNF02 - Aplicação de métodos, técnicas e ferramentas da Engenharia de Software, em conformidade com o Processo Geral de Desenvolvimento de *Frameworks*, conforme descrito na Seção 4.2, do Capítulo 4.

Quanto à confiabilidade, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RNF03 - Precisão numérica adequada dos dados durante a simulação.

Quanto à implementação, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RNF04 - Utilização da linguagem orientada a objetos Java e de *Integrated Development Environments - IDEs.*

Quanto à documentação, o *Framework* Fênix deverá ser capaz de propiciar:

- FEN-RNF05 - Produção de artefatos durante o desenvolvimento do protótipo, como especificação de requisitos e modelos conceituais.

A.6 Restrições do Modelo

Quanto às limitações do modelo, o *Framework* Fênix:

- FEN-RES01 - Tem permissões irrestritas para reuso; e
- FEN-RES02 - Pode ser reutilizado apenas para o desenvolvimento de simuladores na linguagem Java.

Apêndice B - Modelo de Casos de Uso do *Framework* Fênix

B.1 Introdução

Este documento descreve o modelo de casos de uso referente ao *Framework* para a Simulação de Trajetórias de Veículos Espaciais de Sondagem, denominado Fênix. Utilizou-se a sigla FEN-CDU, seguida de uma numeração, para identificar cada caso de uso do Fênix.

B.2 Definição do Papel Simulador

O ator Simulador consiste num sistema de *software* que reusa o *Framework* Fênix para construir e executar simulações de trajetórias de foguetes, conforme mostrado Figura B.1.

B.3 Diagrama de Casos de Uso (DCU)

A Figura B.1 mostra o diagrama de caso de uso do Fênix, onde o ator Simulador interage com o *Framework* para criar a estrutura de fases da simulação e o modelo ma-

temático das entidades envolvidas. Além disso, o *Framework* possibilita que o Simulador avance o tempo da simulação até o impacto do veículo espacial com a superfície da terra.

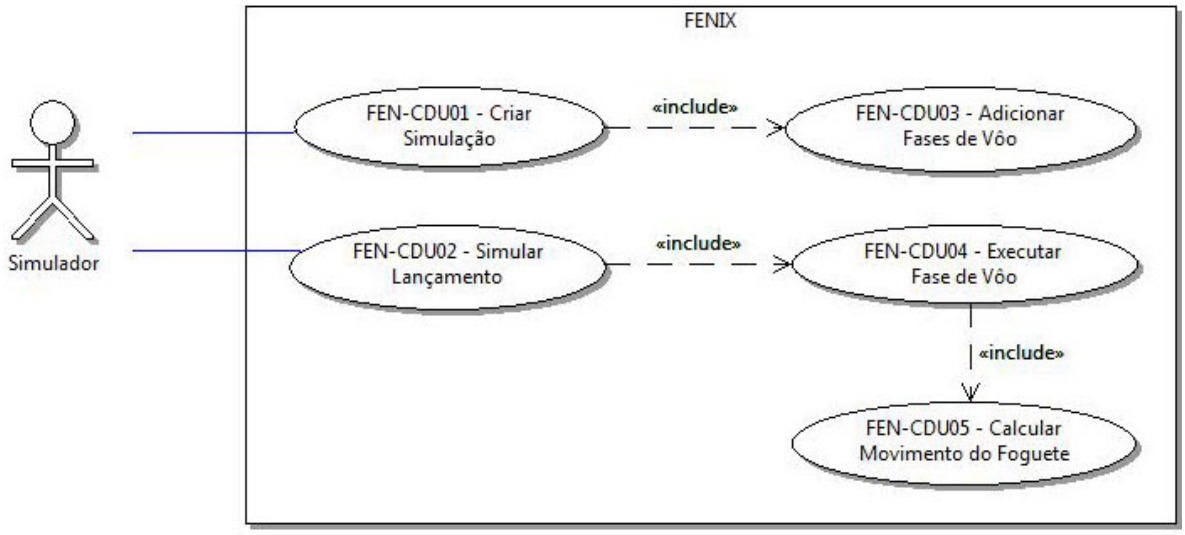

FIGURA B.1 – DCU - *Framework* para Simulação de Trajetórias de Foguetes

B.4 Documentação dos Casos de Uso

Esta Seção descreve os fluxos de eventos dos casos de uso definidos para o Fênix.

Nome: FEN-CDU01 - Criar Simulação **Sumário**: O Simulador cria uma simulação de lançamento de foguete.
Ator Primário: Simulador
Pré-Condições: Parâmetros e tabelas das fases disponíveis.
Fluxo Principal: 1) O Simulador cria uma nova simulação; 2) O Simulador define o modelo da terra; 3) O Simulador determina o modelo da atmosfera; 4) O Simulador fornece o modelo do radar; 5) O Simulador implementa o modelo do veículo; 6) O Simulador estabelece o modelo do lançador; 7) O *Framework* cria uma estrutura do tipo fila para armazenar as fases da simulação; 8) O Simulador adiciona as fases de vôo. Este Evento é descrito no FEN-CDU02; e 9) Fim do caso de uso.
Pós-Condições: Modelo da simulação de lançamento do foguete criado.

Nome: FEN-CDU02 - Adicionar fases de vôo **Sumário**: O Simulador adiciona os parâmetros e tabelas necessários para execução das fases de lançamento.
Ator Primário: Simulador
Pré-Condições: Parâmetros e tabelas de cada fase disponíveis.
Fluxo Principal: 1) O Simulador cria uma nova fase; 2) O Simulador especifica o tempo de início da fase; 3) O Simulador determina a condição necessária para o fim da fase, geralmente, ocorrendo quando um parâmetro de interesse (por exemplo: tempo ou altitude) atinge um valor pré-determinado; 4) O Simulador implementa os comportamentos de inicialização e finalização da fase, que permitem manipular dos dados entre as fases de vôo; 5) O Simulador implementa os comportamentos de pré e pós-integração da fase, que permitem manipular os dados antes e depois da integração das equações de movimento a cada avanço de tempo da fase; 6) O Simulador define o valor do incremento de tempo usado pela rotina de integração; 7) O Simulador informa o valor do incremento de tempo para saída de dados; 8) O Simulador fornece os valores dos demais parâmetros da fase; 9) O Simulador adiciona as tabelas necessárias para execução da fase; 10) O Simulador adiciona a nova fase no final da fila de fases da simulação; 11) Enquanto existerem mais fases, retorna ao Evento 1; e 12) Fim do caso de uso.
Pós-Condições: Todas as fases adicionadas à fila de fases da simulação.

Nome: FEN-CDU03 - Simular Lançamento **Sumário**: O Simulador solicita a execução da simulação de lançamento do foguete.
Ator Primário: Simulador
Pré-Condições: Realização do FEN-CDU01 - Criar Simulação.
Fluxo Principal: 1) O *Framework* carrega os parâmetros e tabelas da próxima fase da simulação; 2) O Simulador requisita a execução da fase atual, solicitando continuamente o avanço do tempo da simulação. Este Evento é descrito no FEN-CDU04 - Executar Fase de Vôo; 3) Enquanto existirem mais fases, retorna ao Evento 1; e 4) Fim do caso de uso.
Pós-Condições: O lançamento do foguete foi simulado.

Nome: FEN-CDU04 - Executar Fase de Vôo **Sumário**: O *Framework* executa uma fase de simulação.
Ator Primário: Simulador
Pré-Condições: Realização do FEN-CDU03 - Simular Lançamento.
Fluxo Principal: 1) O *Framework* inicializa a fase atual; 2) O Simulador solicita avanço do tempo de vôo. 3) O *Framework* calcula a variação de movimento do veículo para o intervalo de tempo considerado. Este Evento é descrito no FEN-CDU05 - Calcular Movimento do Foguete; 4) O Simulador consulta dados de interesse como: posição, velocidade, atitude, dentre outros, em diferentes sistemas de referência; 5) Enquanto a fase não chegar ao fim, retorna ao Evento 2. 6) O *Framework* finaliza a fase atual; e 7) Fim do caso de uso.
Pós-Condições: Uma fase de vôo foi executada.

Nome: FEN-CDU05 - Calcular Movimento do Foguete **Sumário**: Calcula o movimento do foguete de acordo com o modelo matemático estabelecido para a fase.
Pré-Condições: Realização do FEN-CDU04 - Executar Fase de Vôo
Fluxo Principal: 1) O *Framework* inicializa a integração (comportamento de pré-integração); 2) O *Framework* calcula a trajetória do veículo, integrando as equações diferenciais que definem o movimento do veículo no intervalo de tempo considerado; 3) O *Framework* finaliza a integração (comportamento de pós-integração); e 4) Fim do caso de uso.
Pós-Condições: A trajetória foi calculada para o intervalo de tempo considerado.

Apêndice C - Modelo de Classes do *Framework* Fênix

C.1 Introdução

Este documento descreve o modelo de classes que definem o *Framework* para a Simulação de Trajetórias de Veículos Espaciais de Sondagem, denominado Fênix.

C.2 Pacote *fenix*

O *Framework* foi definido no pacote *fenix* e possui sua estrutura de classes divididas nos pacotes *matematico*, *simulacao* e *util*, como apresentado na Figura C.1. A classe *fenix.Simulacao* especifica como a simulação será organizada em fases de vôo e executada, sendo detalhada na Figura C.2. As Seções deste Capítulo abordam maiores detalhes sobre as demais classes que integram estes pacotes.

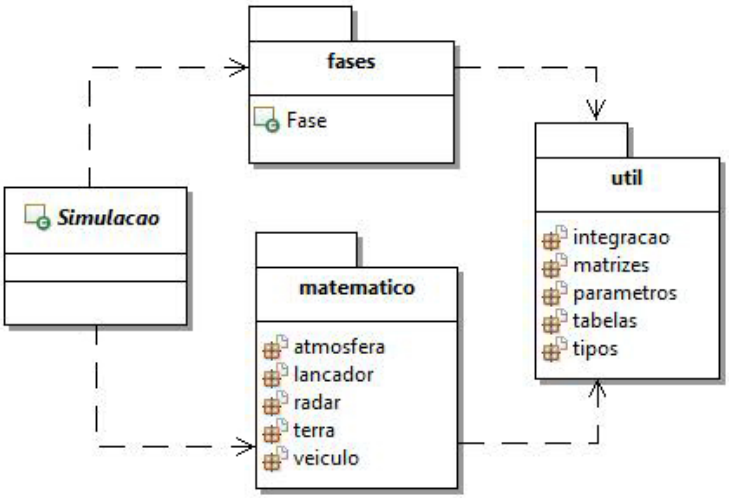

FIGURA C.1 – Diagrama de pacotes do *Framework* Fênix

C.2.1 Pacote *fenix.fases*

Para se criar uma fase de vôo deve-se implementar os métodos abstratos definidos na classe *fenix.fases.Fase*, como ilustrado na Figura C.2. Uma mudança de fase na execução da simulação, relaciona-se a ocorrência de algum evento como: saída do trilho do lançador, liberação de estágios, separação da coifa, saída da atmosfera, apogeu ou impacto.

C.2.2 Pacote *fenix.matematico*

A Figura C.3 mostra o conteúdo do pacote *matematico*, que engloba o modelo matemático das entidades envolvidas na simulação.

Para o cálculo da trajetória de um foguete, o pacote *veiculo* utiliza informações disponibilizadas pelas entidades terra, lançador, radar e atmosfera, modeladas nas classes apresentadas na Figura C.4.

A classe *fenix.matematico.atmosfera.Atmosfera* define um modelo abstrato da atmosfera terrestre, com informações como velocidade do vento, pressão atmosférica, densidade, dentre outras. A classe *fenix.matematico.lancador.Lancador* estabelece, de forma

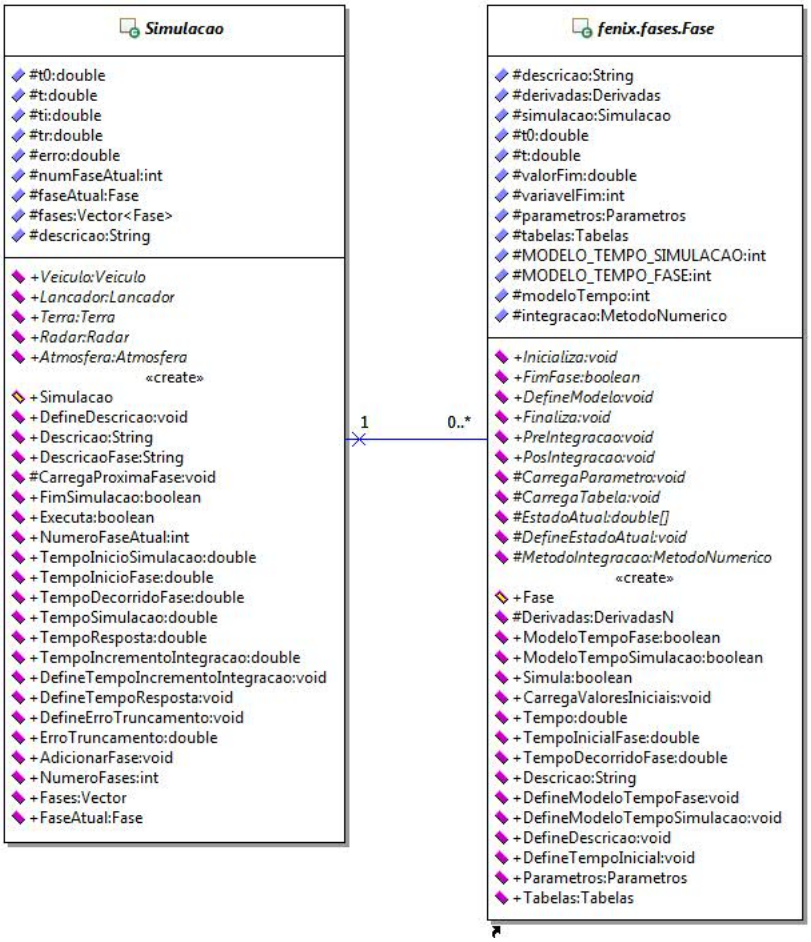

FIGURA C.2 – Classes do pacote *fenix.fases*

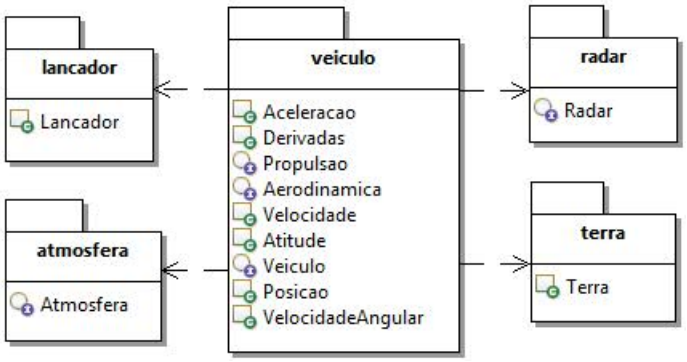

FIGURA C.3 – Estrutura do pacote *fenix.matematico*

abstrata, as características necessárias da plataforma de lançamento, como posição, elevação, azimute, dentre outras. A classe *fenix.matematico.radar.Radar* determina, por meio de abstração, a posição do radar responsável pelo rastreio do veículo. Por fim, a classe *fenix.matematico.terra.Terra* especifica um modelo abstrato do planeta terra, de acordo

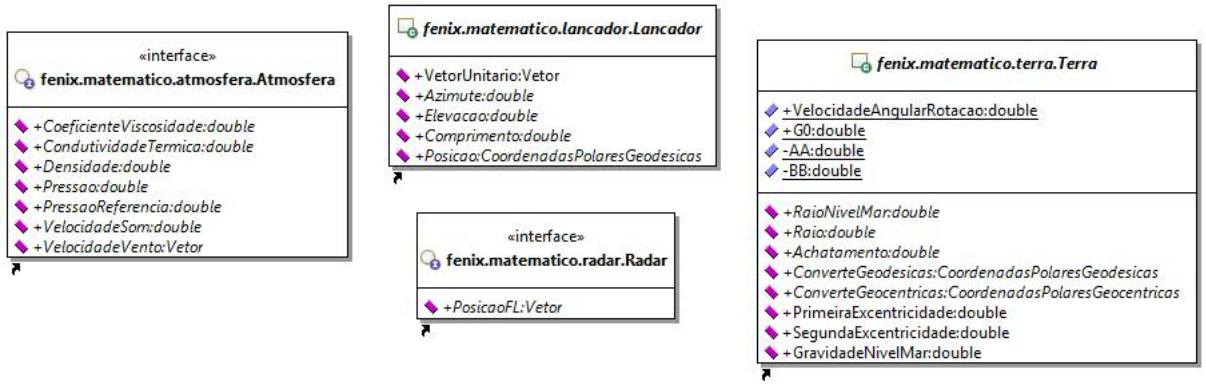

FIGURA C.4 – Classes que representam as entidades terra, atmosfera, lançador e radar

com características como raio e velocidade angular.

O pacote *fenix.matematico.veiculo* possui classes abstratas que definem o comportamento de um veículo espacial, independente da quantidade de graus de liberdade considerados. Suas classes incorporam todas as funcionalidades comuns aos diferentes modelos matemáticos de um foguete. A Figura C.5 mostra o relacionamento entre as classes que formam este pacote.

Escolheu-se manipular as características de um veículo espacial nas classes *Propulsao* e *Aerodinamica*, e as grandezas relativas ao seu movimento do veículo nas classes: *Posicao*, *Velocidade*, *VelocidadeAngular*, *Aceleracao* e *Atitude*. Para o cálculo da variação dessas grandezas em função do tempo criou-se a classe *Derivadas*.

Estas classes apenas especificam o comportamento abstrato de um foguete na simulação. Para a construção de um modelo concreto, torna-se necessário herdá-las, implementando o comportamento específico considerado.

Assim, o desenvolvimento dos modelos matemáticos das entidades da simulação ocorre através da implementação dos métodos abstratos definidos pelas classes do pacote *fenix.matematico*. A estrutura proposta pelo Fênix não varia em diferentes modelos matemáticos e possibilita flexibilidade na construção dos modelos destas entidades.

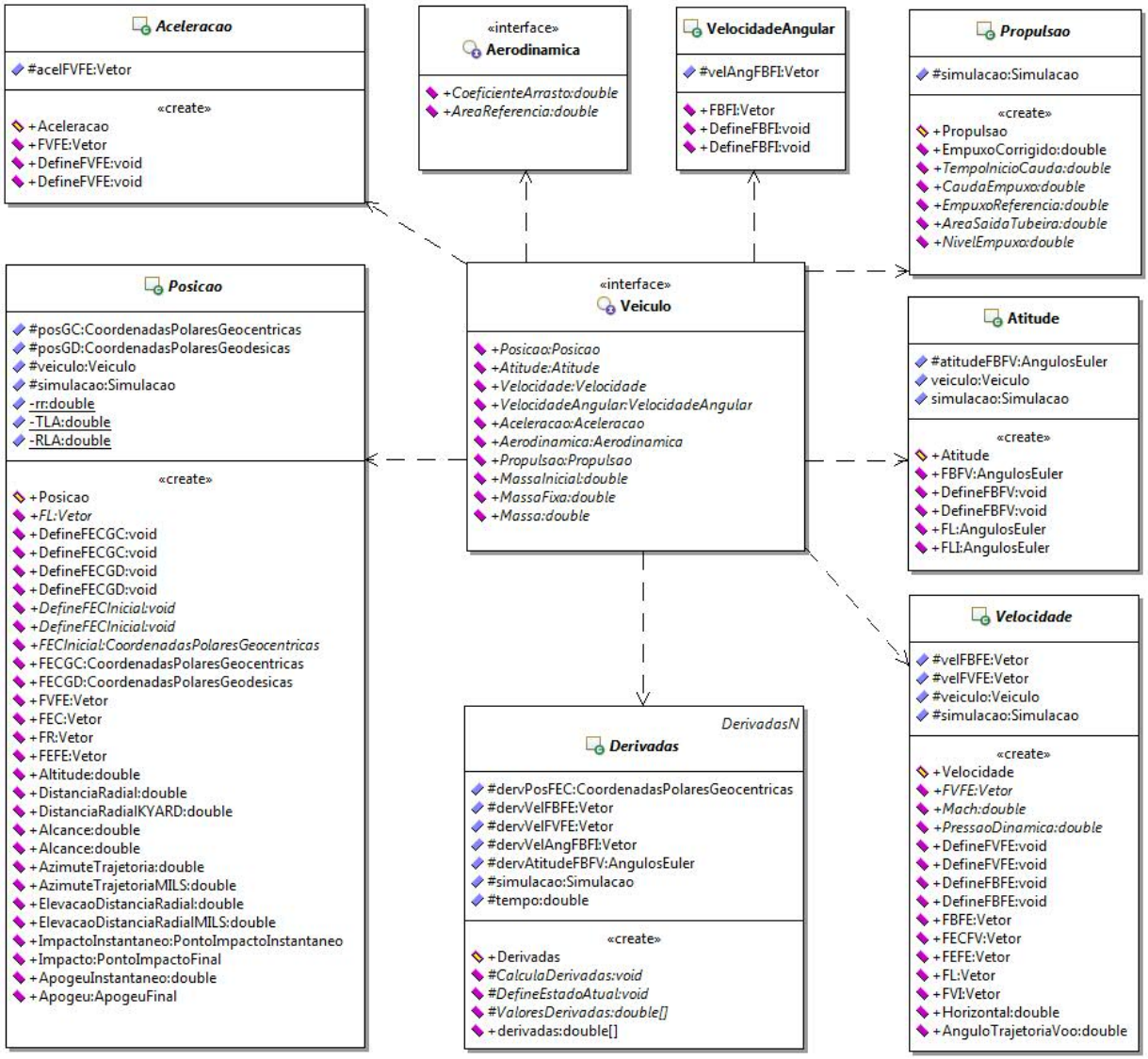

FIGURA C.5 – Modelo abstrato de um veículo espacial

C.2.3 Pacote *fenix.util*

O pacote *util* contém classes que definem estruturas e tipos de dados necessários para a execução da simulação e o cálculo da trajetória do foguete. Ele abrange classes de suporte necessárias aos modelos matemático e da simulação. A Figura C.6 apresenta sua estrutura interna de pacotes.

A Figura C.7 mostra as classes que formam os pacotes *fenix.util.integracao* e *fenix.util.matrizes*. O pacote *integracao* define um método numérico abstrato para a in-

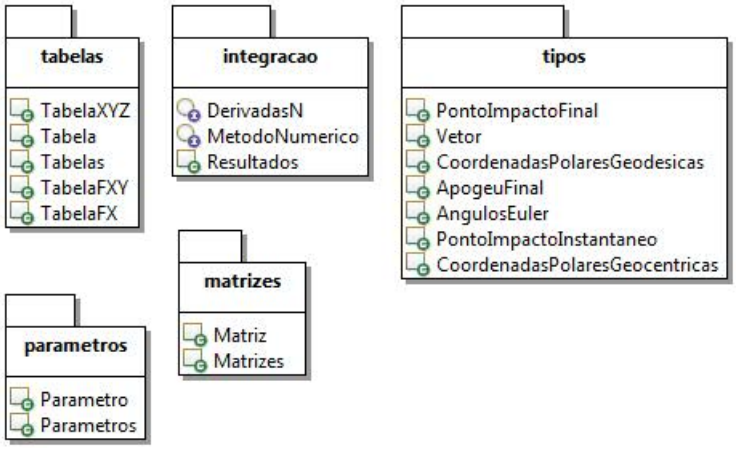

FIGURA C.6 – Estrutura do pacote *fenix.util*

tegração das equações de movimento do foguete. Assim, a escolha do método utilizado fica a critério do desenvolvedor de cada simulador. Mudar o método escolhido não se torna um problema, desde que seja implementado conforme a assinatura especificada.

Já o pacote *matrizes* define estruturas do tipo matriz, usadas para converter grandezas como posição, velocidade, dentre outras, entre sistemas de referência. Para isto reusou-se a biblioteca *Jama*, disponível em (NIST, 2005), que disponibiliza alguns métodos para o cálculo de operações envolvendo matrizes e também para a resolução de sistemas lineares.

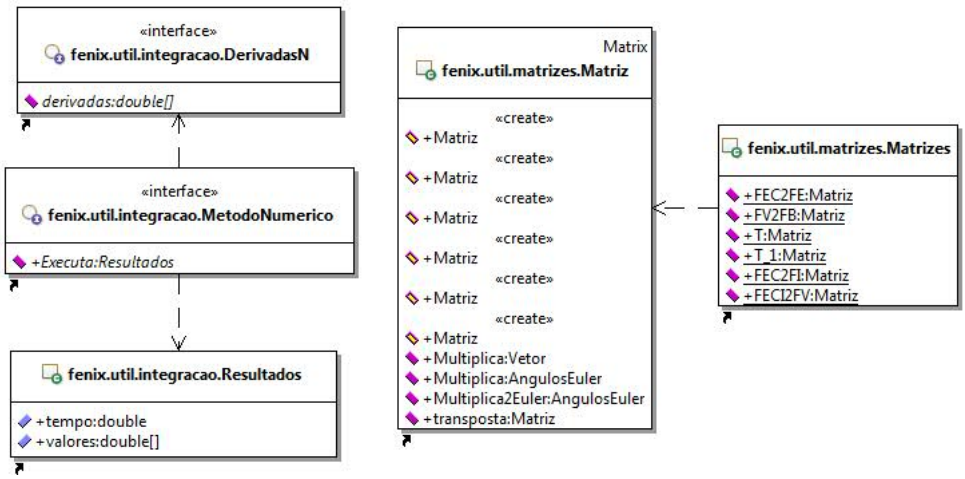

FIGURA C.7 – Classes dos pacotes *fenix.util.integracao* e *fenix.util.matrizes*

Para determinar o formato dos dados de entrada, criou-se estruturas de dados para armazenar parâmetros e tabelas nos pacotes *fenix.util.parametros* e *fenix.util.tabelas*. Estes

pacotes permitem manipular os dados de entrada de forma abstrata, pois o *Framework* não pretende restringir a forma como são adquiridos pelos simuladores. A Figura C.8 demonstra as classes destes pacotes.

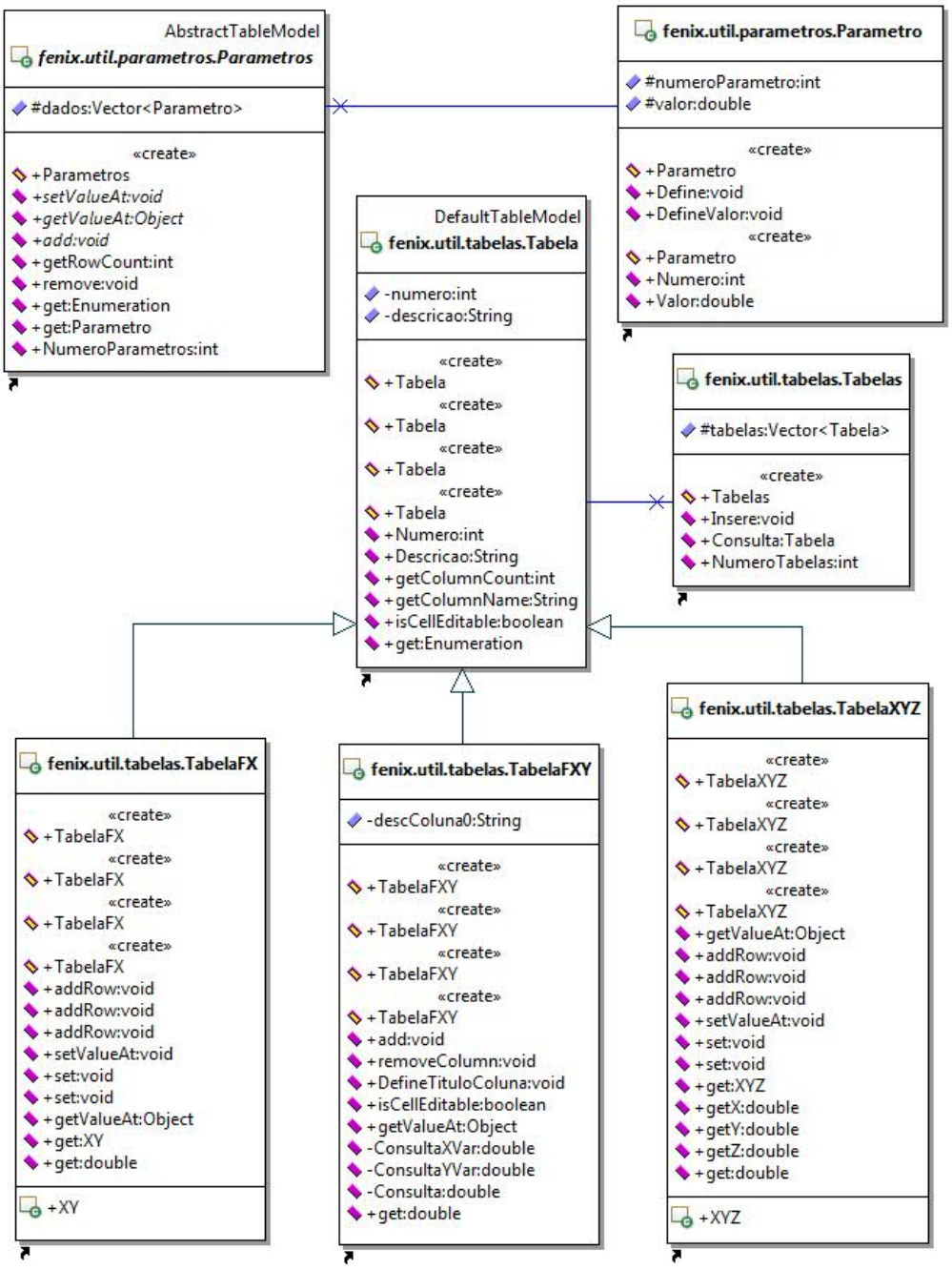

FIGURA C.8 – Classes dos pacotes *fenix.util.parametros* e *fenix.util.tabelas*

Para finalizar, o pacote *fenix.util.tipos* define novas classes de tipos de dados usados na simulação como: vetores, coordenadas polares geodésicas e geocêntricas, ângulos de

euler, dentre outros. Estes novos tipos armazenam os dados da trajetória de um foguete, conforme visualizado no diagrama da Figura C.9.

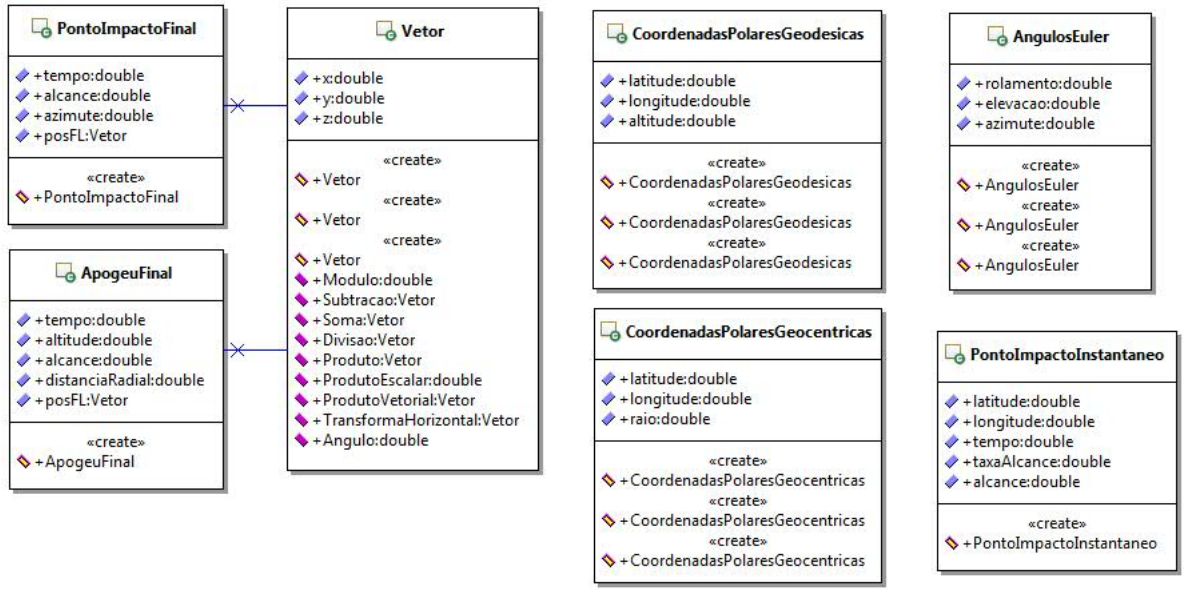

FIGURA C.9 – Classes do pacote *fenix.util.tipos*

C.3 Modelo de Classes por Caso de Uso

A Figura C.10 demonstra o relacionamento entre as principais classes do *Framework* Fênix para permitir a realização do caso de uso FEN-CDU01 - Criar Simulação, incluindo a realização do caso FEN-CDU03 - Adicionar Fases de Vôo.

Já para a realização do caso de uso FEN-CDU02 - Simular Lançamento, englobando os casos FEN-CDU04 - Executar Fase de Vôo e FEN-CDU05 - Calcular Movimento do foguete, relacionam-se principalmente as classes apresentadas na Figura C.11.

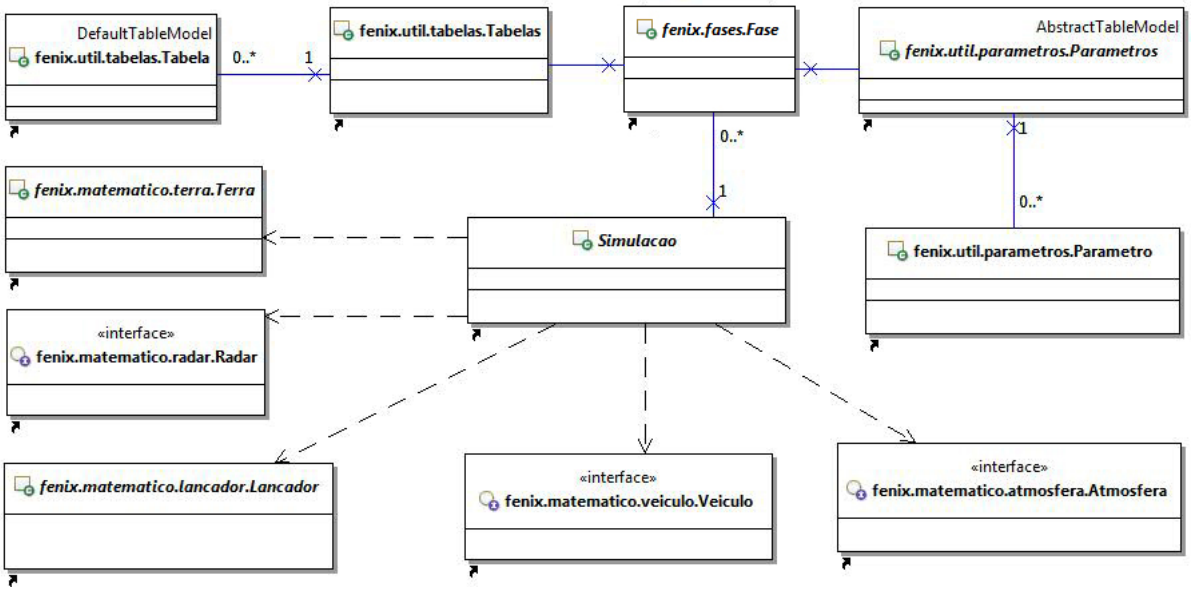

FIGURA C.10 – Classes para a realização do FEN-CDU01

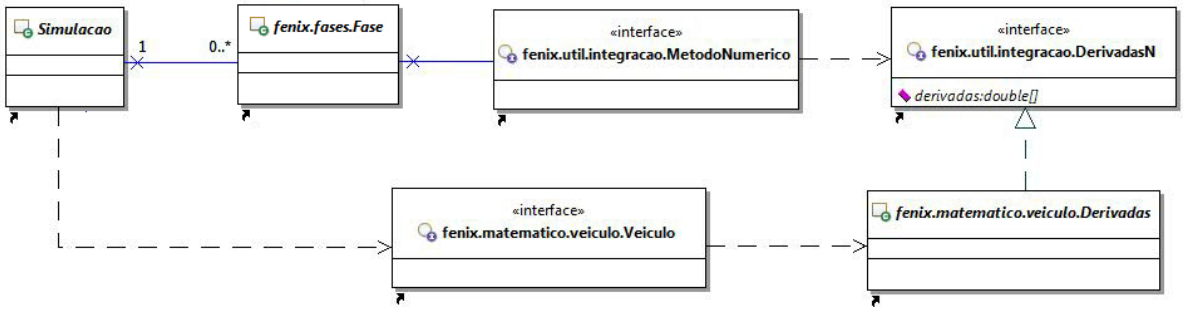

FIGURA C.11 – Classes para a realização do FEN-CDU02

Apêndice D - Diagrama de Sequência do *Framework* Fênix

D.1 Introdução

Este documento apresenta um diagrama de seqüência do *Framework* Fênix, referente ao fluxo de execução de uma simulação, como especificado pelo caso de uso FEN-CDU 02 - Simular Lançamento.

D.2 Diagrama de Sequência

O diagrama de seqüência observado na Figura D.1 mostra os principais eventos ocorridos no método *Executa()*, da classe *fenix.Simulacao*. Para o progresso da simulação, um Simulador solicita o avanço de tempo através invocação deste método, fornecendo o tempo desejado como parâmetro. Assume-se que a primeira fase foi previamente carregada e inicializada (o que ocorre após ela ser adicionada à simulação). Os métodos *Inicializa()*, *Finaliza()*, *PreIntegracao()*, *PosInteracao()* e *FimFase()* são abstratos. Deve-se implementar estes métodos conforme o significado expresso para cada fase de vôo num simulador específico.

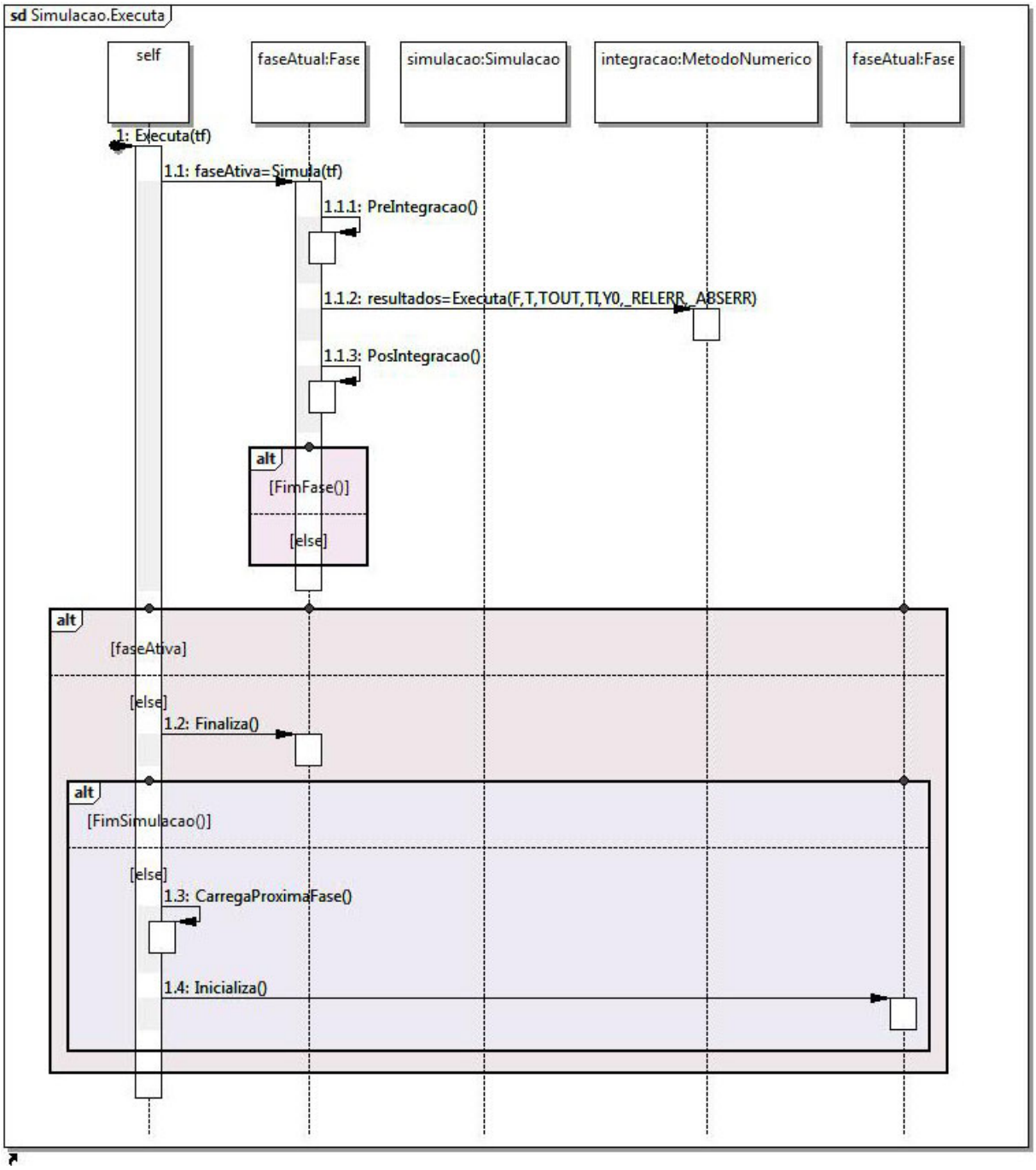

FIGURA D.1 – Método *Executa()* da classe *Simulacao*.

D.2.1 Descrição dos Eventos

Os eventos que ocorrem no método *Executa()*, conforme mostrados na Figura D.1, são descritos a seguir:

1.1 - Ao receber o novo tempo desejado para a simulação, o método solicita o avanço da

fase atual até este tempo, invocando o método *Simula()* da classe *Fase*, através do objeto faseAtual;

1.1.1 - Para realizar isto, inicialmente, o método *Simula()* processa o comportamento de pré-integração definido para a fase atual, que permite manipular os dados do foguete antes da integração;

1.1.2 - Em seguida, executa-se o método de integração, fornecendo como parâmetros as equações diferenciais (F), o tempo inicial (T) e final (TOUT) do intervalo para integração, os dados definindo o estado atual (Y0) do foguete, e os valores de erro relativo (_RELERR) e absoluto (_ABSERR) aceitos para o cálculo. Este método retorna um objeto contendo o tempo alcançado ao final da integração, e os dados definindo o novo estado do foguete;

1.1.3 - Após o cálculo do movimento do foguete, processa-se o comportamento de pós-integração da fase atual, permintindo manipular os dados do foguete após a integração;

Condição - Para finalizar a execução do método *Simula()*, é verificado se o fim da fase foi alcançado. Caso afirmativo, este método retorna falso para a variável faseAtiva; caso contrário, retorna verdadeiro;

Condição - O método *Executa()* verifica se a fase continua ativa. Caso afirmativo, este método retorna verdadeiro para o simulador; caso contrário, executa o Evento 1.2;

1.2 - Com o fim da fase atual, executa-se o método *Finaliza()* associado ao objeto *faseAtual*, que possibilita a manipulação de dados do foguete ou da simulação para concluir a fase;

Condição - O método *Executa()* precisa verificar se com o término da fase também

não atingiu-se o fim da simulação. Caso afirmativo, o método retorna falso para o simulador; caso contrário, executa o Evento 1.3;

1.3 - Como a simulação ainda não chegou ao fim, o objeto *faseAtual* passa apontar para a próxima fase. Em seguida, carrega-se os parâmetros e as tabelas desta fase; e

1.4 - Após carregar a nova fase, executa-se o método *Inicializa()*, que também permite manipular os dados dos foguete ou da simulação antes de começar a execução da fase. Para encerrar, o método *Executa()* retorna verdadeiro para o simulador, indicando que a simulação ainda encontra-se ativa.

Apêndice E - Requisitos do Protótipo SIMSonda

E.1 Introdução

Esta especificação de requisitos descreve o comportamento de um Simulador de Trajetórias de Foguetes de Sondagem, denominada SIMSonda, incluindo seus Requisitos Funcionais (RF), Não-Funcionais (RNF), restrições e outros fatores necessários à sua completa descrição. O documento procura controlar todos os requisitos de software durante o seu ciclo de desenvolvimento assegurando que o sistema atenda às necessidades do usuário final.

Os Requisitos Funcionais, Não-Funcionais e Restrições definidos para o *Framework* Fênix, descritos no Apêndice A, e identificados, respectivamente, como FEN-RF, FEN-RNF e FEN-RS, também devem ser atendidos pelo Simulador SIMSonda. Utilizou-se as siglas SIM-RF, SIM-RNF e SIM-RS, seguidas de uma numeração, para identificar os requisitos e restrições do SIMSonda.

E.1.1 Definição da Solução Proposta

Este Estudo de Caso tem como objetivo desenvolver um Protótipo de Simulador de Trajetórias de Veículos Espaciais de Sondagem, a fim de verificar o modelo abstrato definido pelo *Framework* Fênix, reduzindo a ocorrência de erros de projeto ou implementação.

E.1.2 Escopo

Este documento refere-se à especificação dos requisitos do Simulador SIMSonda e se aplica a todas as fases e atividades do seu desenvolvimento, com o intuito de auxiliar analistas e desenvolvedores na modelagem e construção de um sistema com qualidade e que atenda às necessidades dos envolvidos.

E.2 Visão Geral do Sistema

O Simulador SIMSonda irá prover a simulação de trajetórias de foguetes de sondagem, considerando modelos matemáticos para cálculo do movimento do veículo com um, três ou seis graus de liberdade. Além disso, o Simulador irá disponibilizar os dados da trajetória simulada de um foguete para outras aplicações, num ambiente de simulação distribuída. Os requisitos do Simulador foram especificados com base na Arquitetura de *Software* idealizada na Seção 5.1, do Capítulo 5.

E.3 Lista de Requisitos

As Seções E.3.1 e E.3.2 apresentam os Requisitos Funcionais (RF) e Não-Funcionais (RNF) especificados para o Simulador SIMSonda.

E.3.1 Requisitos Funcionais (RF)

Quanto ao Módulo do Modelo Matemático, o Simulador SIMSonda deverá ser capaz de propiciar os mesmos requisitos do *Framework* Fênix. Entretanto, o simulador deverá considerar os modelos matemáticos utilizados pelo programa ROSI, descritos em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Assim, os requisitos FEN-RF01 e FEN-RF02 especificados para o *Framework* Fênix, no Apêndice A, serão atendidos através dos requisitos SIM-RF01 e SIM-RF02, descritos a seguir:

- SIM-RF01 - Criação de modelos das entidades envolvidas, considerando como modelo da terra os modelos *Geoid Internacional* ou IAU 1964 e como modelo da atmosfera o modelo *U.S. Standard Atmosphere* 1962. Além disso, considera-se os modelos de veículo, radar e lançador utilizados no programa ROSI; e

- SIM-RF02 - Cálculo do movimento do veículo espacial, considerando três modelos utilizados no programa ROSI: modelo da rampa de lançamento, para movimento unidimensional; modelo atmosférico, para movimento em seis dimensões; e modelo atmosférico mais simplificado, para movimento tridimensional.

Quanto ao Módulo do Modelo de Fases, além dos requisitos FEN-RF03 e FEN-RF04 especificados para o *Framework* Fênix, no Anexo A, o Simulador SIMSonda também deverá ser capaz de propiciar:

- SIM-RF03 - Aquisição dos dados de entrada necessários para execução da simulação no formato utilizado pelo programa ROSI. Esses dados são parâmetros e tabelas utilizados pelos modelos das entidades terra, lançador, radar, atmosfera e veículo, além de parâmetros para controle da simulação;

- SIM-RF04 - Disponibilização dos dados que registram o vôo do veículo através de tabelas e gráficos e nos formatos utilizados pelo programa ROSI. Esses dados referem-se às grandezas relativas ao veículo em função do seu tempo de vôo, para diferentes sistemas de referência, além do apogeu da trajetória e do ponto de impacto do veículo com a superfície terrestre; e

- SIM-RF05 - Interação com outras aplicações ou simuladores geograficamente distribuídos, compartilhando informações referentes a trajetória simulada do veículo espacial.

E.3.2 Requisitos Não-Funcionais (RNF)

Além dos Requisitos Não-Funcionais especificados para o *Framework* Fênix, quanto à implementação, o Simulador SIMSonda deverá ser capaz de propiciar:

- SIM-RNF01 - Utilização da infra-estrutura de *runtime* pRTI1.3 (PITCH, 2005), que implementa o padrão *IEEE STD 1516 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)* (IEEE, 2000a), proporcionando o sincronismo e a comunicação necessários para a execução de uma simulação distribuída.

E.4 Restrições do Sistema

Quanto às limitações do sistema, o Simulador SIMSonda:

- SIM-RES01 - Tem permissões irrestritas para uso;

- SIM-RES02 - Pode ser utilizado apenas em computadores que possuam a *Java Virtual Machine (JVM)* 1.6 ou superior instalada; e

- SIM-RES03 - Pode ser utilizado para uma simulação distribuída apenas em computadores que disponham do *hardware* de interface de rede.

Apêndice F - Modelo de Casos de Uso do Protótipo SIMSonda

F.1 Introdução

Este documento descreve o modelo de casos de uso referente ao *Simulador* de Trajetórias de Veículos Espaciais de Sondagem denominado SIMSonda. Os casos de uso do *Framework Fênix*, identificados pela sigla FEN-CDU, encontram-se descritos no Apêndice B. Utilizou-se a sigla SIM-CDU, seguida de uma numeração, para identificar cada caso de uso do SIMSonda.

F.2 Definição de Papéis

O ator Usuário refere-se a pessoa que opera o Simulador SIMSonda, conforme mostrado Figura F.1. Já o ator Sistema de Visualização Remoto consiste num *software* que recebe as informações compartilhadas pelo SIMSonda num ambiente de simulação distribuído.

F.3 Diagrama de Casos de Uso (DCU)

A Figura F.1 apresenta o diagrama de caso de uso do Protótipo do Simulador SIMSonda. Neste diagrama, observa-se a interação dos atores Usuário e Sistema de Visualização Remoto com o Simulador SIMSonda, além da sua interação com o *Framework* Fênix. Os casos de uso FEN-CDU01 e FEN-CDU02 encontram-se descritos no Apêndice B.

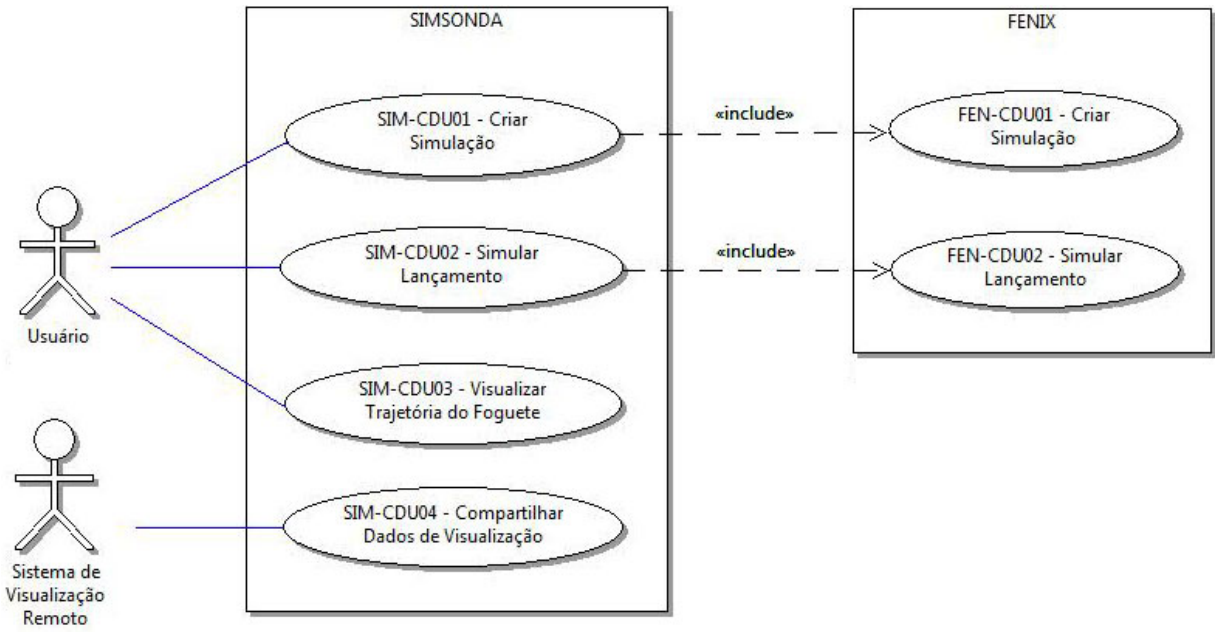

FIGURA F.1 – DCU - Simulador de Trajetórias de Foguetes

Na Figura F.1, o Usuário interage com o SIMSonda para criar a estrutura de fases da simulação e o modelo matemático das entidades envolvidas. Ele também utiliza o SIMSonda para executar uma simulação, avançando o tempo de vôo até o impacto do veículo espacial com a superfície da terra, enquanto visualiza os dados da trajetória simulada através da interface gráfica. Além disso, o Simulador SIMSonda também compartilha estes dados com um Sistema de Visualização Remoto, durante a execução de uma simulação distribuída.

F.4 Documentação dos Casos de Uso

Nome: SIM-CDU01 - Criar Simulação **Sumário**: O Usuário cria uma simulação de lançamento de foguetes de sondagem.
Ator Primário: Usuário
Pré-Condições: Arquivo de entrada disponível.
Fluxo Principal: 1) O Usuário seleciona a opção para criar uma nova simulação; 2) O Simulador SIMSonda solicita o arquivo de entrada, contendo os parâmetros e tabelas das fases de simulação; 3) O Simulador SIMSonda cria a simulação com base no arquivo entrada, conforme especificado pelo caso de uso FEN-CDU01 - Criar Simulação, descrito no Apêndice B; 4) O Simulador SIMSonda exibe a configuração das fases da simulação; e 5) Fim do caso de uso.
Pós-Condições: Configuração definida para a simulação de lançamento do foguete.

Nome: SIM-CDU02 - Simular Lançamento **Sumário**: O Usuário solicita a execução da simulação de lançamento do foguete.
Ator Primário: Simulador
Pré-Condições: Realização do SIM-CDU01 - Criar Simulação.
Fluxo Principal: 1) O Usuário solicita a execução da simulação; 2) O Simulador SIMSonda solicita o avanço de tempo da simulação, repetidamente, até o impacto do veículo com a superfície da terra, conforme especificado no FEN-CDU02, disponível no Apêndice B; 3) O Simulador SIMSonda informa que a simulação foi realizada com sucesso; e 4) Fim do caso de uso.
Pós-Condições: O lançamento do foguete foi simulado e os dados da sua trajetória encontram-se disponíveis.

Nome: SIM-CDU03 - Visualizar Trajetória do Foguete **Sumário**: O usuário visualiza os dados que definem a trajetória do foguete.
Ator Primário: Usuário
Pré-Condições: Realização do SIM-CDU02 - Simular Lançamento.
Fluxo Principal: 1) O Usuário seleciona a opção de visualizar a tabela de resultados da simulação; 2) O Simulador SIMSonda exibe uma tabela com os dados simulado; e 3) Fim do caso de uso. **Fluxo Alternativo**: **(1) Histórico de Eventos** a) O Usuário escolhe a opção de visualizar o histórico de eventos da simulação; b) O Simulador SIMSonda exibe uma tabela contendo a descrição e a estampa de tempo dos eventos; e c) Fim do caso de uso. **Fluxo Alternativo**: **(1) Visualizar Arquivo de Saída** a) O Usuário seleciona a opção de visualizar o arquivo com os resultados da simulação, no formato de saída usado pelo Instituto de Aeronáutica e Espaço (IAE); b) O Simulador SIMSonda mostra o conteúdo do arquivo de saída; e c) Fim do caso de uso. **Fluxo Alternativo**: **(1) Visualizar Arquivo de Visualização** a) O Usuário escolhe a opção de visualizar o arquivo com os resultados da simulação, no formato usado pelo IAE para visualização gráfica de algumas grandezas; b) O Simulador SIMSonda exibe o conteúdo do arquivo de visualização; e c) Fim do caso de uso. **Fluxo Alternativo**: **(1) Visualizar Gráficos** a) O Usuário seleciona a opção de visualizar os gráficos de algumas grandezas referentes ao movimento do foguete; b) O Usuário seleciona a grandeza de interesse; c) O Simulador SIMSonda apresenta o gráfico da grandeza selecionada em função do tempo de vôo; e d) Fim do caso de uso.
Pós-Condições: Os resultados da simulação foram apresentados ao Usuário.

Nome: SIM-CDU04 - Compartilhar Dados de Visualização **Sumário**: Compartilha dados da trajetória do foguete com um Sistema de Visualização Remoto.
Ator Primário: Sistema de Visualização Remoto
Pré-Condições: Realização do SIM-CDU02 - Simular Lançamento.
Fluxo Principal: 1) O Simulador SIMSonda especifica quais informações referentes a trajetória do foguete serão compartilhadas com os outros sistemas; 2) O Sistema de Visualização Remoto se inscreve nestas informações compartilhadas; 3) O Simulador SIMSonda divulga os novos valores calculados para estas informações a cada avanço do tempo de vôo, durante a execução de uma simulação distribuída; 4) O Sistema de Visualização Remoto recebe cada atualização de valores das informações compartilhadas pelo SIMSonda; e 5) Fim do caso de uso.
Pós-Condições: Os resultados da simulação foram compartilhados com o Sistema de Visualização Remoto.

Apêndice G - Modelo de Classes do Protótipo SIMSonda

G.1 Introdução

Este documento descreve o modelo de classes que definem o Protótipo do Simulador de Trajetórias de Foguetes de Sondagem, denominado SIMSonda, e também a Federação de Visualização Remota de Trajetórias.

G.2 Pacote *simsonda*

O Simulador SIMSonda foi definido no pacote *simsonda* e possui sua estrutura de classes divididas nos pacotes *aplicacao*, *infraestrutura* e *simulacao*, como mostrado na Figura G.1. As seções deste Capítulo apresentam maiores detalhes sobre as classes que integram estes pacotes.

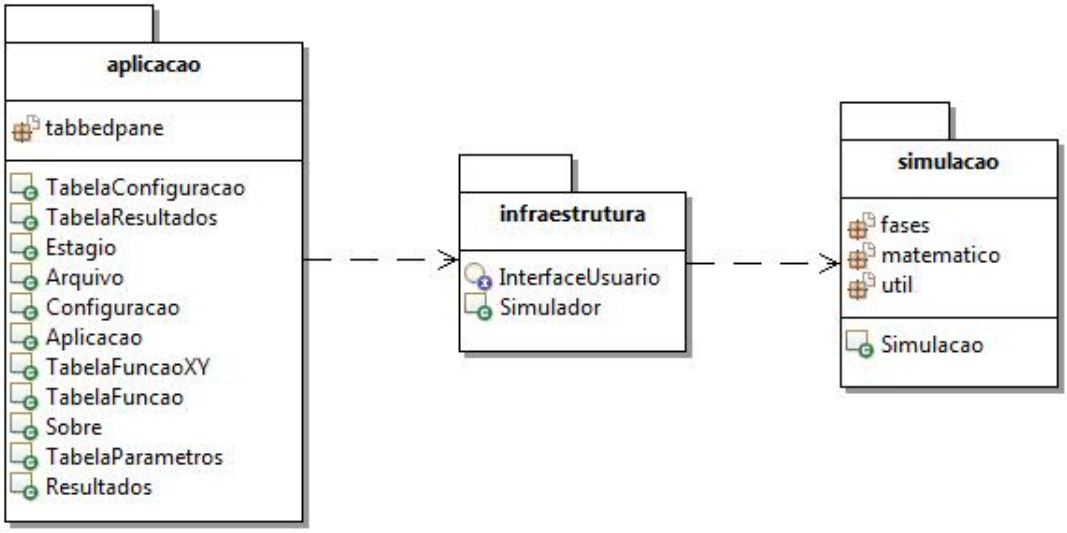

FIGURA G.1 – Diagrama de pacotes do Simulador SIMSonda

G.2.1 Pacote *simsonda.simulacao*

No pacote *simulacao* demonstrou-se como reutilizar o *Framework* Fênix para construir três modelos de simulação de trajetórias de foguetes. Este pacote foi sub-dividido nos pacotes *matematico*, *fases* e *util*, como ilustrado na Figura G.2.

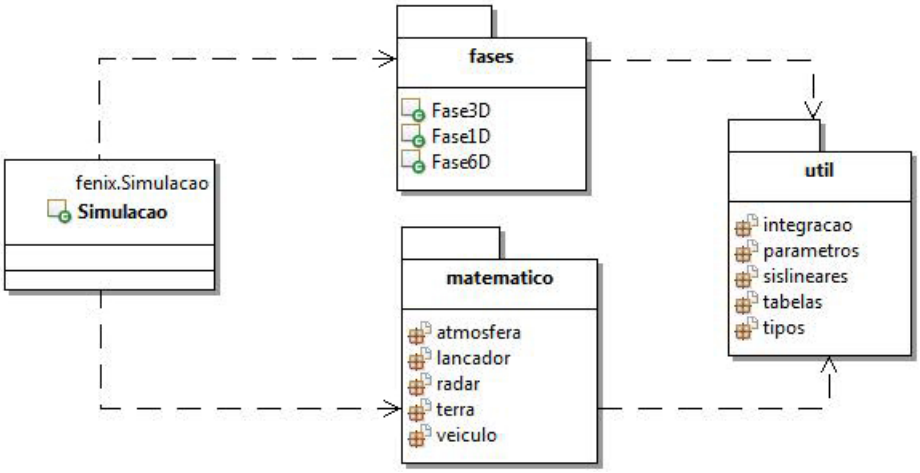

FIGURA G.2 – Estrutura do pacote *simsonda.simulacao*

A classe *simsonda.simulacao.Simulacao*, mostrada na Figura G.3, torna concretos os métodos abstratos da classe *fenix.Simulacao*, que determinam as instâncias dos modelos utilizados para terra, radar, atmosfera e lançador, de acordo como implementados no pacote *simsonda.simulacao.matematico*.

G.2.1.1 Pacote *simsonda.simulacao.fases*

Para a definição dos tipos de fase do Simulador SIMSonda, herdou-se a classe *Fase*, do pacote *fenix.fases*, implementando seus métodos abstratos, como apresenta a Figura G.3. Criou-se três tipos de fases distintos, associados aos modelos matemáticos de veículo espacial, definidos no pacote *simsonda.simulacao.matematico.veiculo*.

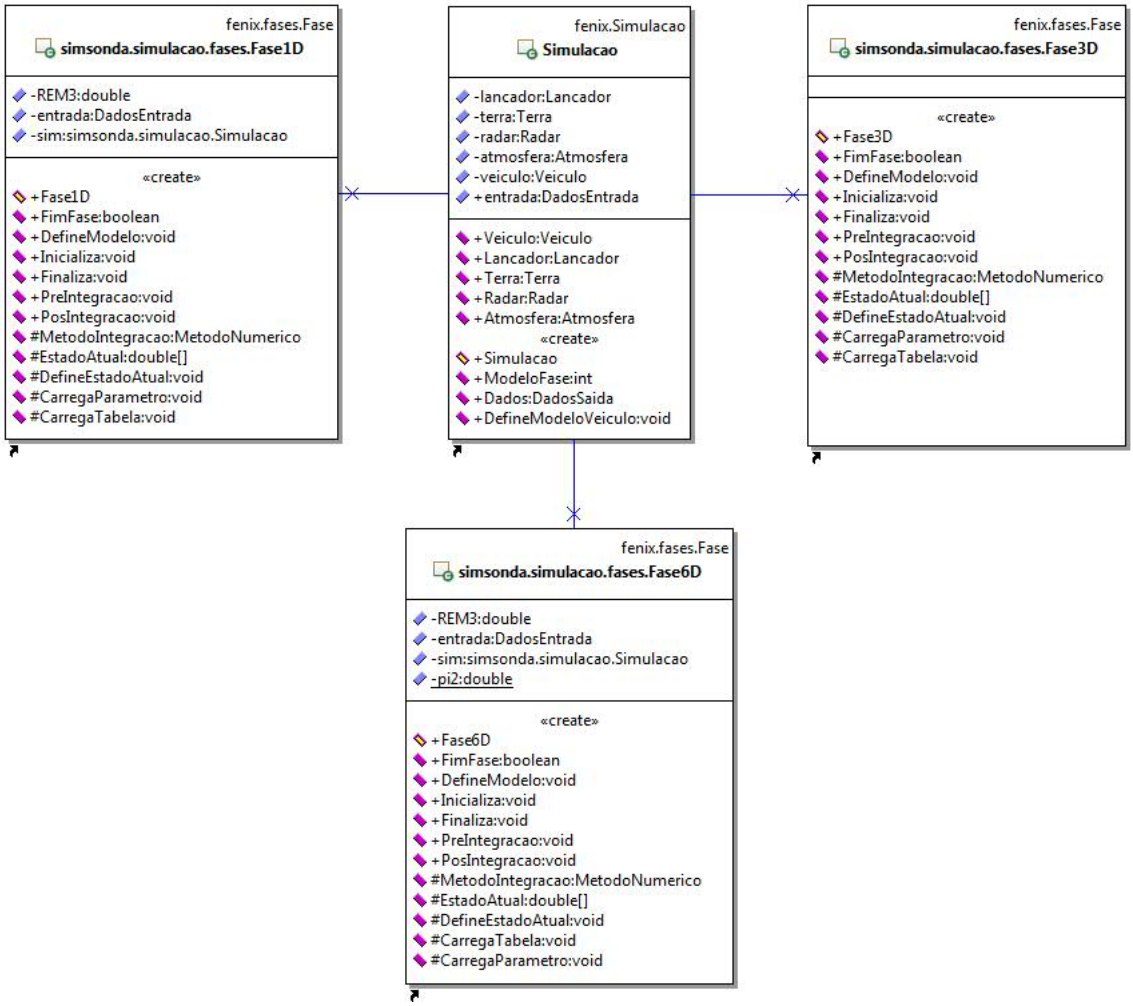

FIGURA G.3 – Classes que formam o pacote simsonda.modelo.simulacao

Na classe *Fase1D*, o método *DefineModelo()* garante a utilização do Modelo 1D de veículo. espacial O término da fase, implementado no método *FimFase()*, ocorre quando o veículo atingir o fim do trilho do lançador. O método *Inicializa()* realiza as conversões iniciais de algumas grandezas para o início da simulação e estabelece que o método de

integração deve utilizar as derivadas do Modelo 1D para o cálculo de movimento do veículo. Os métodos *Finaliza()*, *PreIntegracao()* e *PosIntegracao()* não foram utilizados neste tipo de fase.

O método *DefineModelo()*, da classe *Fase3D*, determina a utilização do Modelo 3D de veículo. Já o método *FimFase()* estabelece como término da fase quando um parâmetro selecionado atingir um valor pré-determinado, sendo este parâmetro o tempo de simulação ou a altitude do foguete. O método *Inicializa()* define a utilização das derivadas do Modelo 3D pelo método de integração. Os métodos *Finaliza()*, *PreIntegracao()* e *PosIntegracao()* também não foram utilizados neste tipo de fase.

A classe *Fase6D* estabelece a utilização do Modelo 6D de veículo, no método *DefineModelo()*. O término da fase foi definido da mesma forma como ocorreu na classe *Fase3D*. O método *Inicializa()* determina a utilização das derivadas do Modelo 6D pelo método de integração. O método *Finaliza()* converte a velocidade do veículo para diferentes sistemas de referência. O método *PosIntegracao()* permite ajustar o valor de rolamento do veículo a cada rotação e também converter a sua velocidade entre sistemas de referência. O método *PreIntegracao()* não foi utilizado neste tipo de fase.

G.2.1.2 Pacote *simsonda.simulacao.matematico*

No pacote *matematico* foram desenvolvidos os modelos concretos das entidades especificadas no *Framework* Fênix, de acordo com os modelos de equações utilizados pelo programa ROSI, descritos em (KRAMER; CRAUBNER; ZIEGLTRUM, 1976). Este pacote é sub-dividido nos pacotes apresentados na Figura G.4.

As classes que definem as entidades terra, radar, lançador e atmosfera podem ser

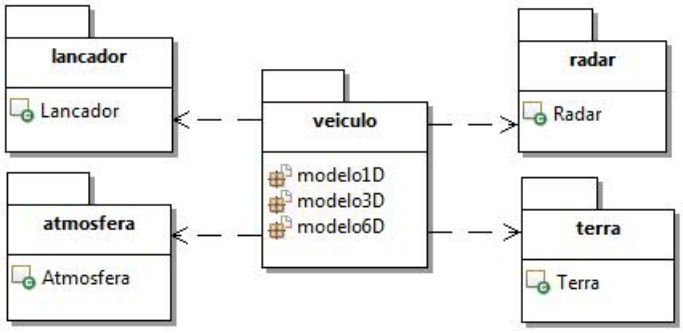

FIGURA G.4 – Estrutura do pacote *simsonda.simulacao.matematico*

visualizadas na Figura G.5.

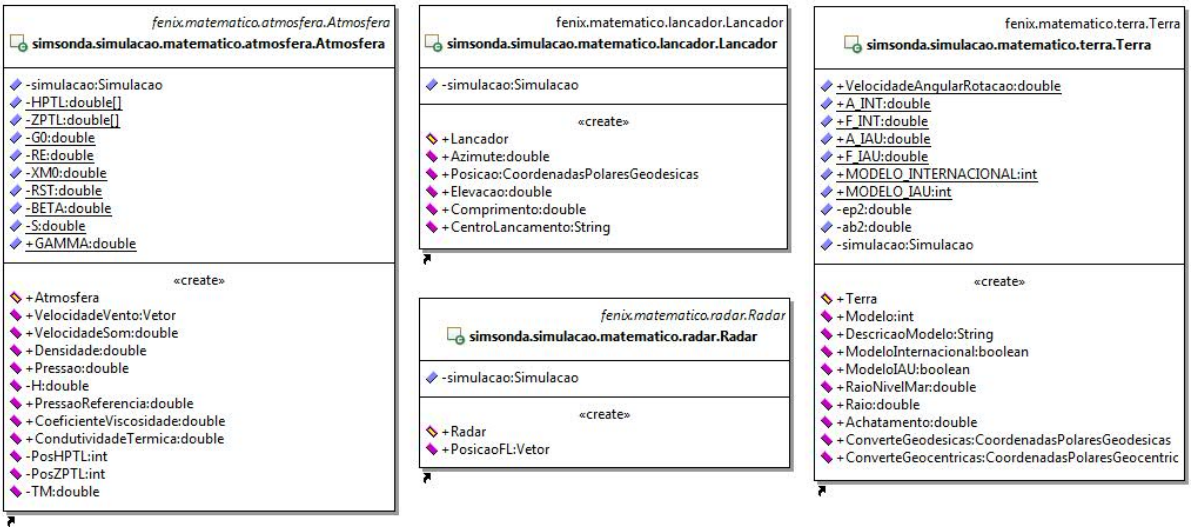

FIGURA G.5 – Classes que definem as entidades envolvidas com a simulação de trajetória

No pacote *simsonda.simulacao.matematico.veiculo* desenvolveu-se três modelos distintos da entidade veículo espacial. Os pacotes *modelo1D*, *modelo3D* e *modelo6D* permitem, respectivamente, calcular o movimento de um foguete considerando um, três ou seis graus de liberdade, como apresentado nas Figuras G.6, G.7 e G.8.

O Modelo 1D determina o movimento de translação do foguete em uma dimensão (no trilho do lançador). A única classe não herdada do *Framework* Fênix foi a classe *Forca*, que fornece o cálculo das forças de gravidade, aerodinâmica e de propulsão do veículo. Apesar da classe *fenix.matematico.veiculo.VelocidadeAngular* ser instanciada neste modelo, a velocidade angular, juntamento com os momentos do veículo são nulos.

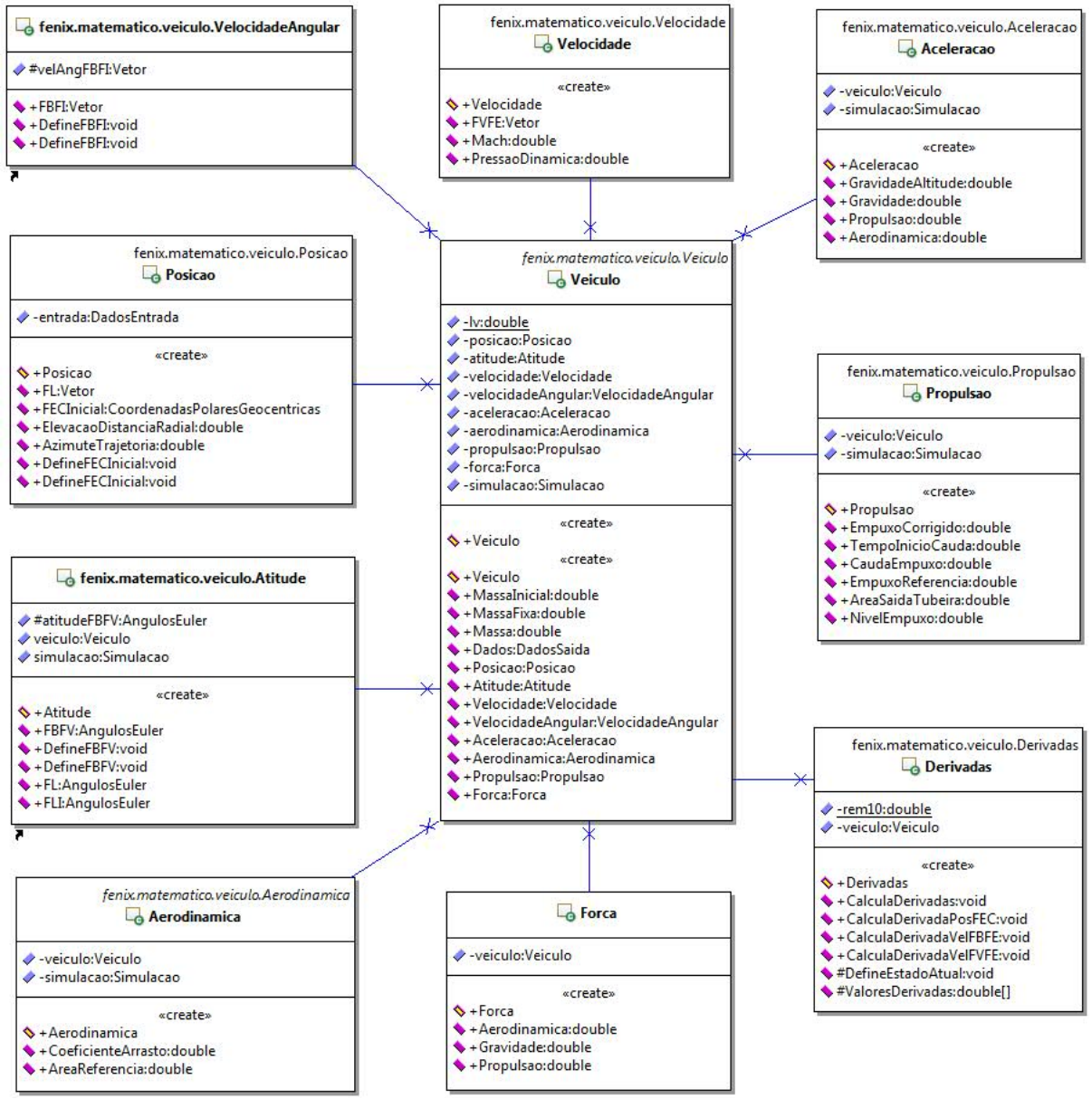

FIGURA G.6 – Classes que definem o Modelo 1D da entidade veículo

No Modelo 1D, a atitude do veículo é determinada pelo lançador, e não varia durante seu movimento preso ao trilho. A conversão da atitude do veículo em diferentes sistemas de referência foi completamente definida na classe *fenix.matematico.veiculo.Atitude*. Os três modelos do Simulador SIMSonda instanciam esta classe sem adicionar nenhum comportamento ou atributo.

O Modelo 3D define o movimento de translação do foguete em três dimensões. Os momentos do veículo também não são considerados neste modelo. Novamente, a única

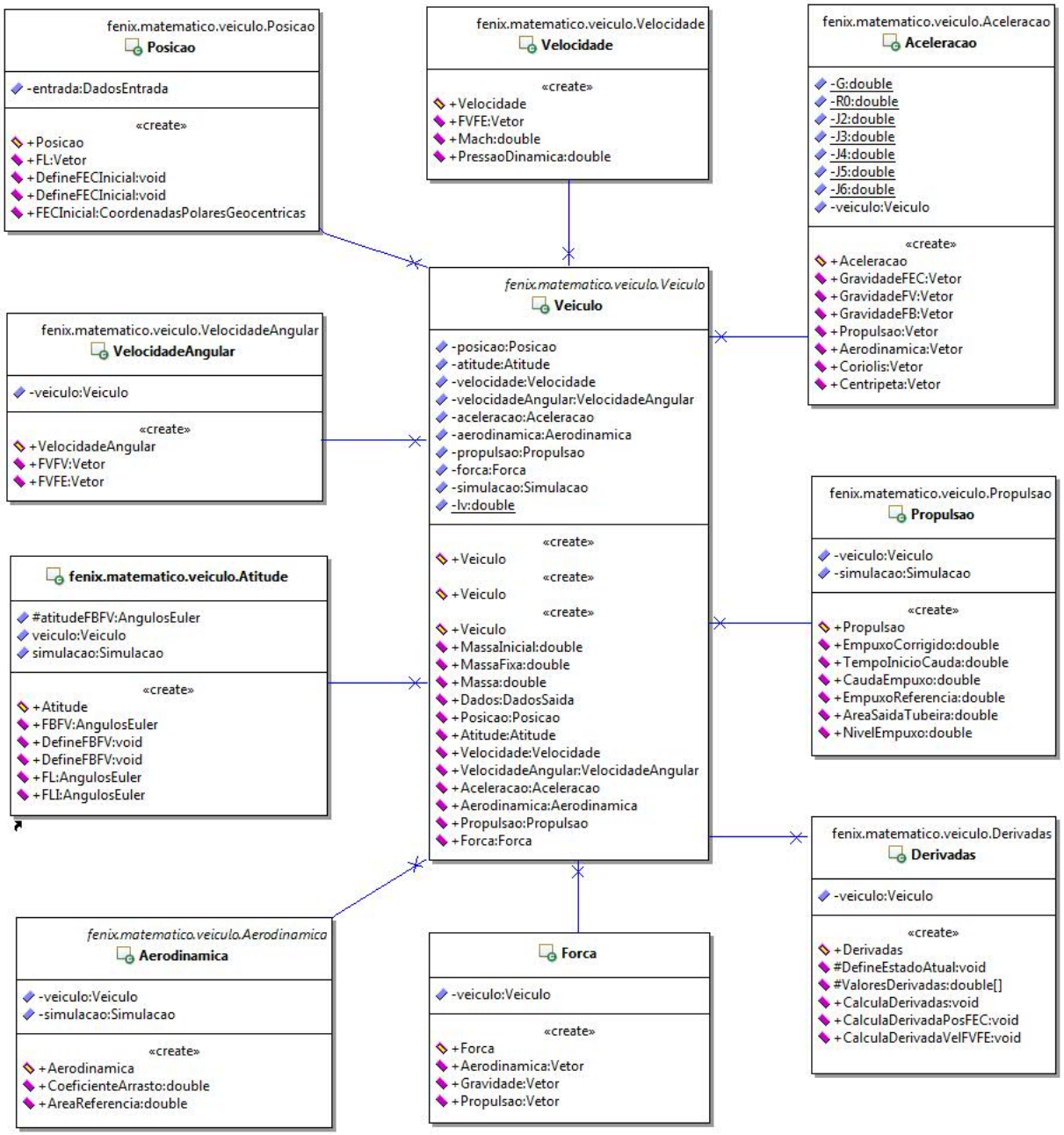

FIGURA G.7 – Classes que definem o Modelo 3D da entidade veículo

classe não herdada do *Framework* Fênix foi a classe *Forca*.

No Modelo 6D, além da classe *Forca*, a classe *Momento* também não foi herdada do *Framework* Fênix, fornecendo o cálculo dos momentos aerodinâmicos, de propulsão e de controle do veículo. Este modelo permite o cálculo dos movimentos de translação e rotação do foguete em três dimensões.

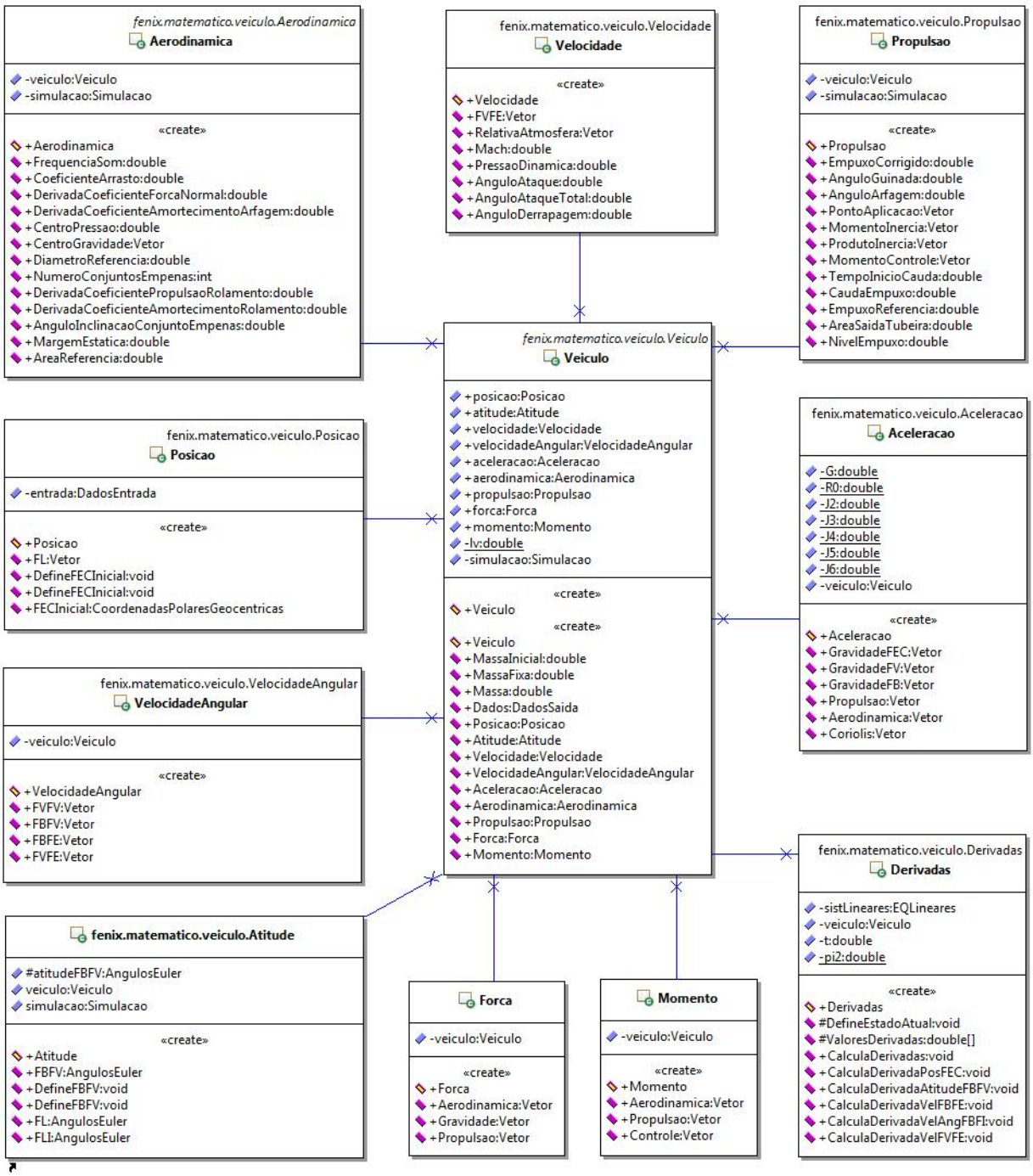

FIGURA G.8 – Classes que definem o Modelo 6D da entidade veículo

G.2.1.3 Pacote *simsonda.simulacao.util*

O pacote *util* foi sub-dividido nos pacotes mostrados na Figura G.9. As classes que formam este pacote podem ser visualizadas na Figura G.10.

O pacote *tipos* define novos tipos de dados para armazenar os dados de entrada e de

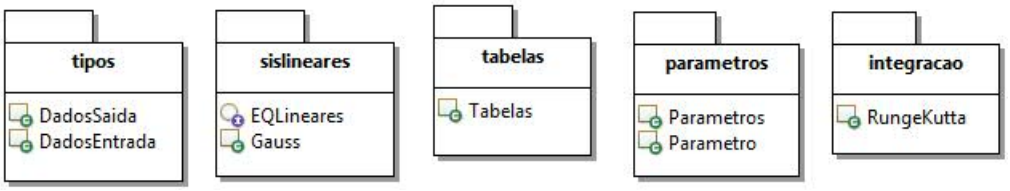

FIGURA G.9 – Estrutura do pacote *simsonda.simulacao.util*

saída da simulação. Como muitos dados de entrada persistem durante as mudanças de fase, definiu-se a classe estática *DadosEntrada* para armazenar os parâmetros e as tabelas atuais da simulação. Os métodos abstratos, definidos no *Framework* Fênix, para acessar os dados de entrada, foram implementados, no pacote *simulacao*, fazendo referência a atributos dessa classe estática.

O pacote *sislineares* contém uma interface, chamada *EQLineares*, que especifica um método para resolução de sistemas lineares. Este método foi utilizado no cálculo das derivadas do movimento do veículo considerando seis graus de liberdade. A classe *Gauss* implementa esta interface considerando o método de eliminação de Gauss com pivotação na diagonal principal. A utilização desta interface permite a implementação de outros métodos sem alterar o restante do modelo matemático.

Os pacotes *parametros* e *tabelas* definem estruturas de dados concretas para armazenar, respectivamente, parâmetros e tabelas de entrada. No *Framework* Fênix estabeleceu-se estas estruturas de forma abstrata. Já no Simulador SIMSonda, elas foram implementadas conforme o formato e a identificação utilizada pelo programa ROSI (KRAMER; CRAUBNER; ZIEGLTRUM, 1976).

Desenvolveu-se, no pacote *integracao*, o método abstrato especificado na interface *fenix.util.integracao.MetodoNumerico*, para a integração das equações diferenciais de movimento do foguete, sendo considerado o método Runge Kutta de quarta ordem utilizado pelo programa ROSI.

FIGURA G.10 – Classes que formam o pacote *simsonda.simulacao.util*

G.2.2 Pacote *simsonda.infraestrutura*

O pacote *infraestrutura* implementa o Módulo de Infra-estrutura de Execução, conforme ilustra na Figura G.11. A interface *InterfaceUsuario* especifica os métodos utilizados, pela infra-estrutura de execução, para apresentar ao usuário os resultados obtidos durante a execução de uma simulação. Qualquer interface com o usuário de simuladores

que utilizem esta infra-estrutura deve fornecer a implementação destes métodos, conforme ocorre na Classe *simsonda.aplicacao.Aplicacao*.

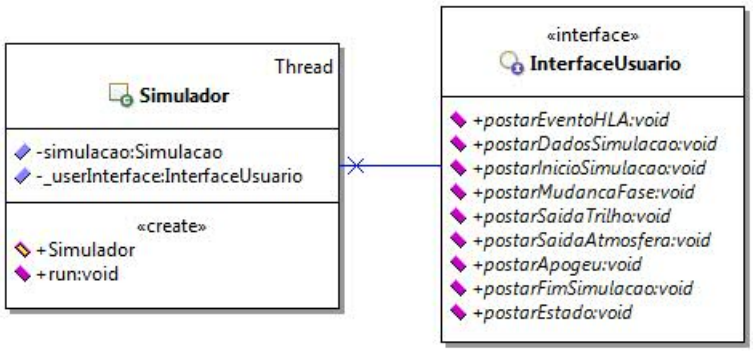

FIGURA G.11 – Estrutura do pacote *simsonda.infraestrutura*

A classe *Simulador* implementa o fluxo de execução que realiza o avanço do tempo de simulação e verifica a ocorrência de eventos como mudanças de fase, apogeu e impacto. Ela herda a classe *java.lang.Thread* para permitir que o fluxo de execução da simulação seja independente do fluxo de execução da interface gráfica com o usuário, possibilitando ao usuário interagir com a interface gráfica durante a execução de uma simulação. Esta classe utiliza todos os métodos especificados pela interface *InterfaceUsuario*, exceto o método *postarEventoHLA()*, que foi definido para uma infra-estrutura distribuída.

G.2.3 Pacote *simsonda.aplicacao*

A Figura G.12 apresenta as classes que formam o pacote *aplicacao*. Para simplificar foram omitidos os atributos e as classes internas da classe *Aplicacao*, onde foi implementada a interface gráfica do Simulador SIMSonda.

O pacote *tabbedpane* consiste na biblioteca *CloseAndMaxTabbedPane*, disponível em (JAVAWORLD.COM, 2004), que possibilitou inserir na interface gráfica um componente do tipo painel com múltiplas guias personalizadas. Além disso, para produzir os gráfi-

FIGURA G.12 – Classes que formam a interface com o usuário

cos utilizou-se a biblioteca *JFreeChart*, disponível em (JFREE.ORG, 2000), que permite o controle de *zoom* e a exportação dos gráficos como imagens, dentre outras coisas.

Na classe *Sobre* foi criada uma janela com informações sobre o autor do Simulador SIMSonda. A classe *Arquivo* permite ler os dados do arquivo de entrada para a simulação, conforme o formato especificado para o programa ROSI.

As demais classes proporcionam a criação das tabelas utilizadas no Protótipo. A classe *TabelaConfiguracao* apresenta uma tabela que armazena informações sobre os estágios que integram o foguete, definidos na classe *Estagio*. O modelo de dados desta tabela de estágios foi determinado na classe *Configuracao*. A classe *TabelaResultados* apresenta os dados de saída da simulação, sendo seu modelo de dados definido na classe *Resultados*. Esse modelo de dados armazena objetos do tipo *simsonda.simulacao.util.tipos.DadosSaida*.

A classe *TabelaFuncao* exibe os dados das tabelas de entrada, considerando apenas uma variável independente. Esta tabela pode utilizar como modelo de dados os modelos definidos nas classes *fenix.util.tabelas.TabelaFX* e *fenix.util.tabelas.TabelaXYZ*. A classe *TabelaFuncaoXY* apresenta os dados das tabelas que possuem duas variáveis independentes, sendo o seu modelo de dados definido na classe *fenix.util.tabelas.TabelaFXY*. Quanto aos parâmetros de entrada, eles foram apresentados através da classe *TabelaParametros*, considerando o modelo de dados definido na classe *fenix.util.parametros.Parametros*.

G.3 Pacote *hla.federacao*

No pacote *hla.federacao* foi desenvolvida a Federação de Visualização Remota de Trajetórias, envolvendo a construção dos Federados Simulador SIMSonda, Sistema de Visualização e Sistema Coordenador. Este pacote divide-se nos pacotes *coordenador*, *visualizacao*,

simulador e *util*, como apresentado na Figura G.13.

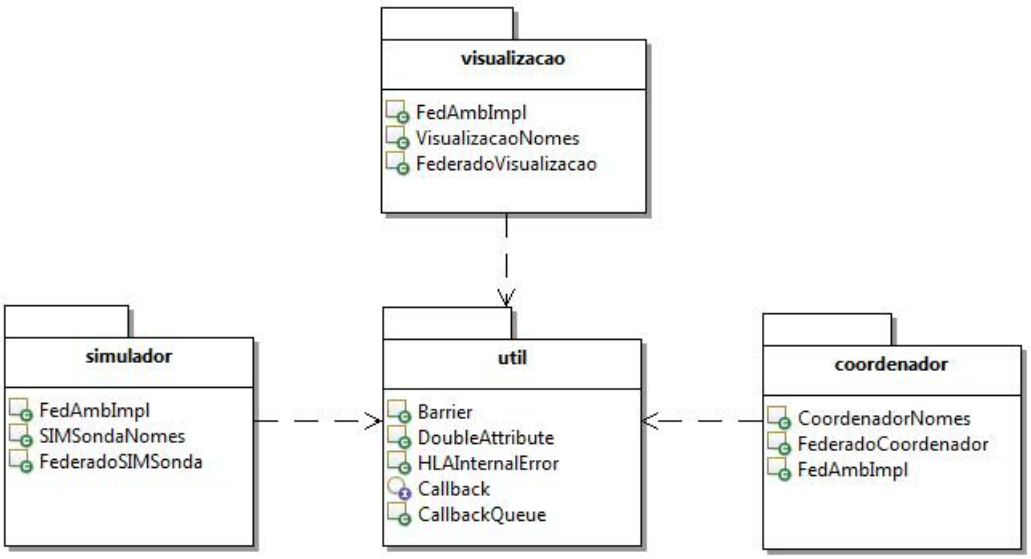

FIGURA G.13 – Estrutura do pacote *hla.federacao*

No pacote *util* implementou-se algumas classes utilitárias de suporte, compartilhadas pelos três Federados, como ilustrado na Figura G.14.

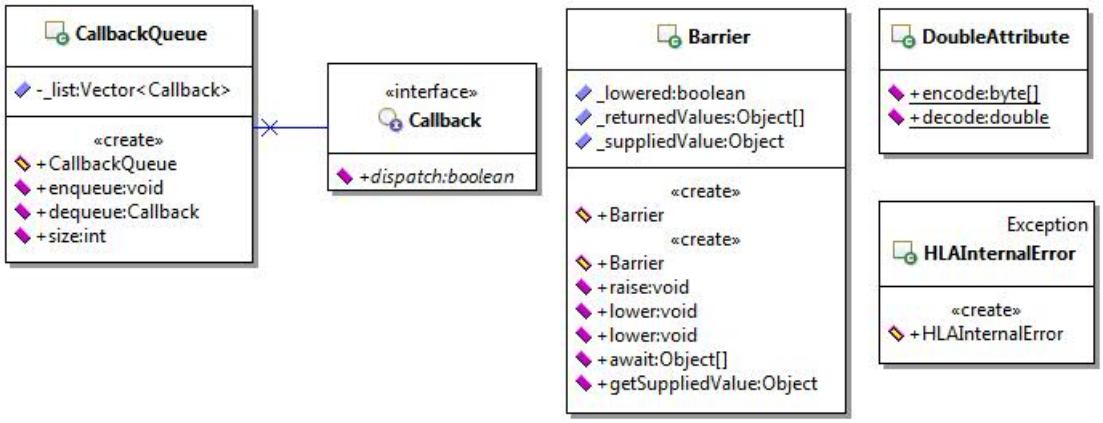

FIGURA G.14 – Classes do pacote *hla.federacao.util*

Para se comunicar com os Federados, a RTI utiliza os métodos da interface *FederateAmbassador* implementados para cada Federado em suas classes *FedAmbImpl*. Quando ocorre a invocação de um método desta interface, o Federado insere a invocação no final de uma estrutura de dados do tipo fila, definida na classe *CallbackQueue*. Esta estrutura armazena as invocações como objetos na forma definida pela interface *Callback*.

A classe *Barrier* permite suspender o fluxo de execução do Federado até chegar a resposta da RTI avisando que a Federação encontra-se sincronizada no ponto de sincronismo envolvido. Esta classe foi utilizada quando o Federado informa a RTI que atingiu um ponto de sincronismo e precisa aguardar a resposta avisando que os demais Federados também encontram-se sincronizados.

Na comunicação entre os Federados e a RTI, o fluxo de dados da forma como visualizada pela RTI encontra-se no formato de *bytes*. A classe *DoubleAttribute* permite a conversão de valores no formato *double* para *bytes* ou o contrário. Já a classe *HLAInternalError* permite o tratamento das exceções que ocorrem durante a execução dos Federados.

As Figuras G.15, G.16 e G.17 apresentam as classes que integram, respectivamente, os Federados Coordenador, Sistema de Visualização e SIMSonda. Os três federados possuem uma classe chamada *FedAmbImpl*, que consiste na implementação da interface *FederateAmbassador*, disponibilizada com o *software* pRTI. Esta interface especifica os métodos dos Federados invocados pela RTI.

A classe *coordenador.InterfaceUsuario* consiste na interface gráfica com o usuário do Coordenador. A classe *FederadoCoordenador* determina o fluxo de execução do Federado Coordenador durante a execução da Federação. Esta classe possui uma estrutura de dados, do tipo definido pela classe *FederateTable*, para armazenar informações sobre os outros Federados membros. O Coordenador controla a quantidade de participantes da Federação, sendo também responsável por registrar os pontos de sincronismo da Federação. Na classe *CoordenadorNomes* foram declarados os nomes das classes de objetos e dos atributos relacionados ao Federado Sistema Coordenador.

Da mesma forma, declarou-se na classe *VisualizacaoNomes* os nomes das classes de objetos e dos atributos relacionados ao Federado Sistema de Visualizacao. A classe *visu-*

FIGURA G.15 – Classes do Federado Sistema Coordenador

alizacao.InterfaceUsuario corresponde a interface gráfica com o usuário. A classe *Dados* encapsula os dados recebidos pelo Federado Visualização durante a execução de uma Federação. A classe *FederadoVisualizacao* estabelece o fluxo de execução para este Federado dentro da Federação.

O fluxo de execução do Federado Simulador SIMSonda, definido na classe FederadoSIMSonda, determina a execução de uma simulação de trajetória de um foguete. Os

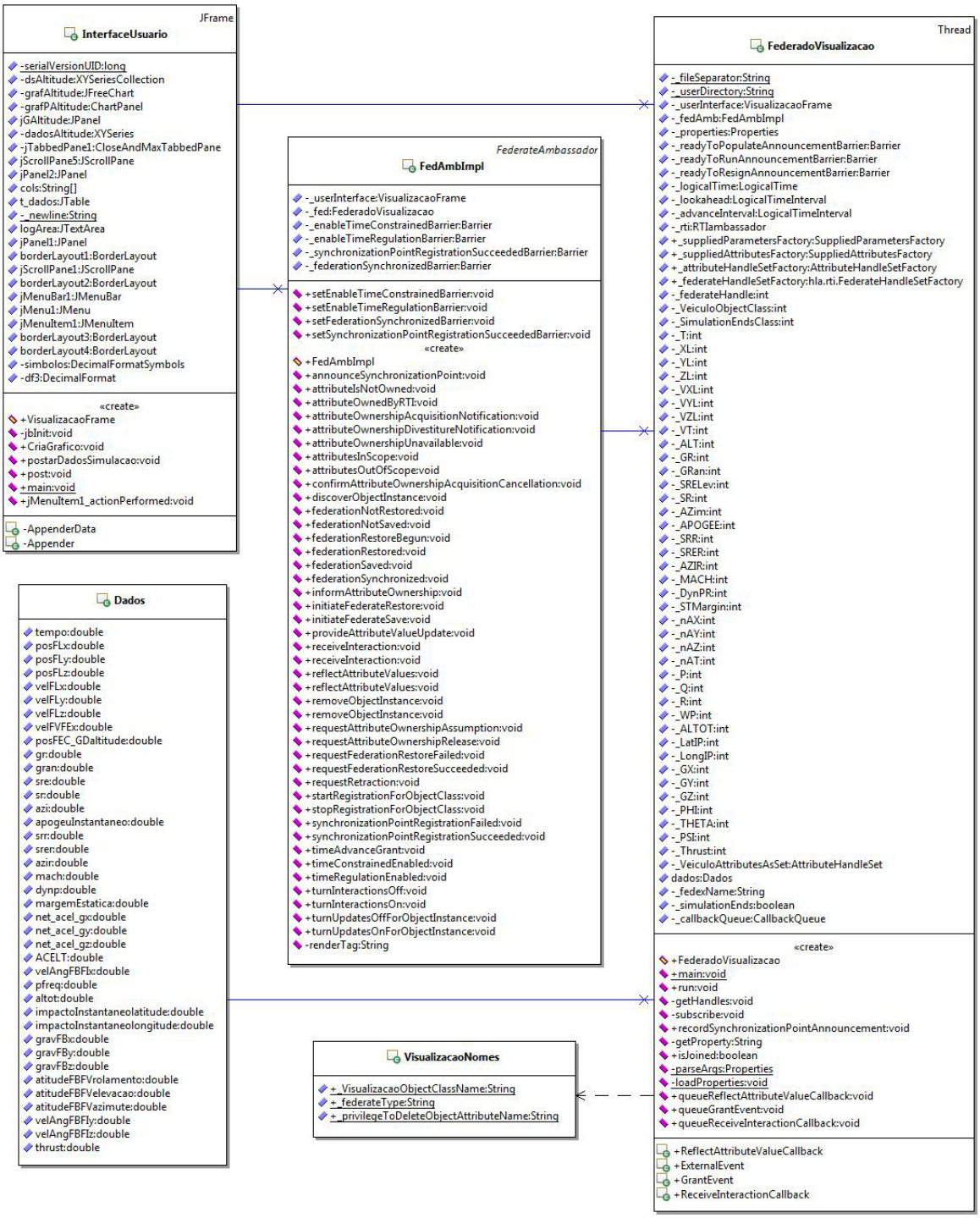

FIGURA G.16 – Classes do Federado Sistema de Visualização

objetos e interações relacionados a trajetória do foguete foram declarados na classe *SIMSondaNomes*. A interface *fenix.infraestrutura.InterfaceUsuario* especifica os métodos abstratos utilizados pelo Federado SIMSonda para apresentar informações na interface com o usuário, sendo implementada no pacote *simsonda.aplicacao*.

FIGURA G.17 – Classes do Federado Simulador SIMSonda

G.4 Modelo de Classes por Caso de Uso

A Figura C.10 mostra o relacionamento das principais classes para a realização do SIM-CDU01 - Criar Simulação, considerando a realização do FEN-CDU01 - Criar Simulação, descrito no Apêndice C. Cria-se uma simulação definindo os modelos matemáticos das entidades terra, radar, lançador, atmosfera e veículo.

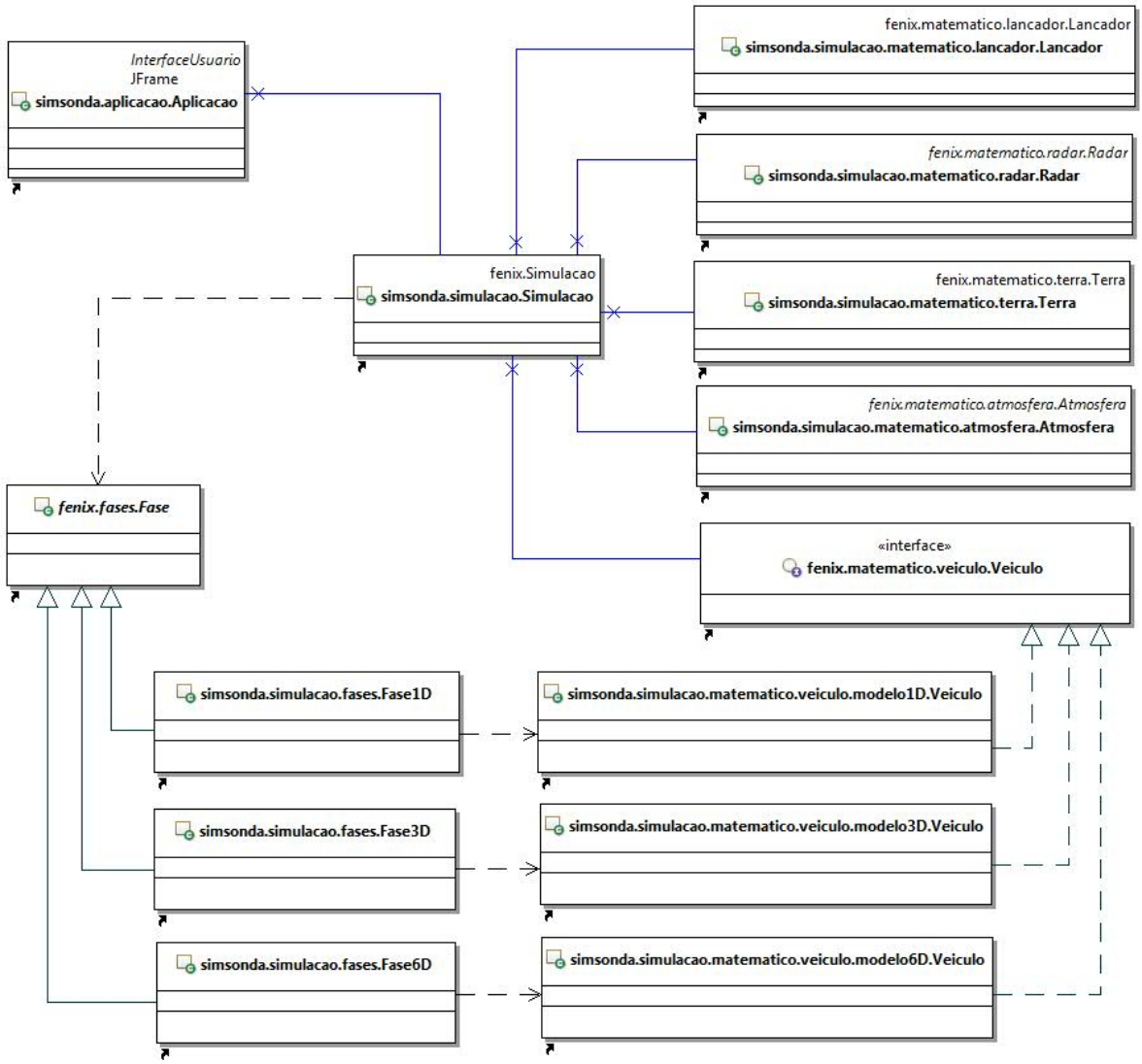

FIGURA G.18 – Diagrama de Classes do SIM-CDU01

Para a realização do SIM-CDU02 - Simular Lançamento, as classes foram relacionadas como ilustra a Figura G.19. Este caso de uso também inclui o FEN-CDU02 - Simular Lançamento, descrito no Apêndice C. As equações diferenciais usadas para o cálculo do

movimento do foguete dependem do modelo adotado para o veículo em cada fase da simulação.

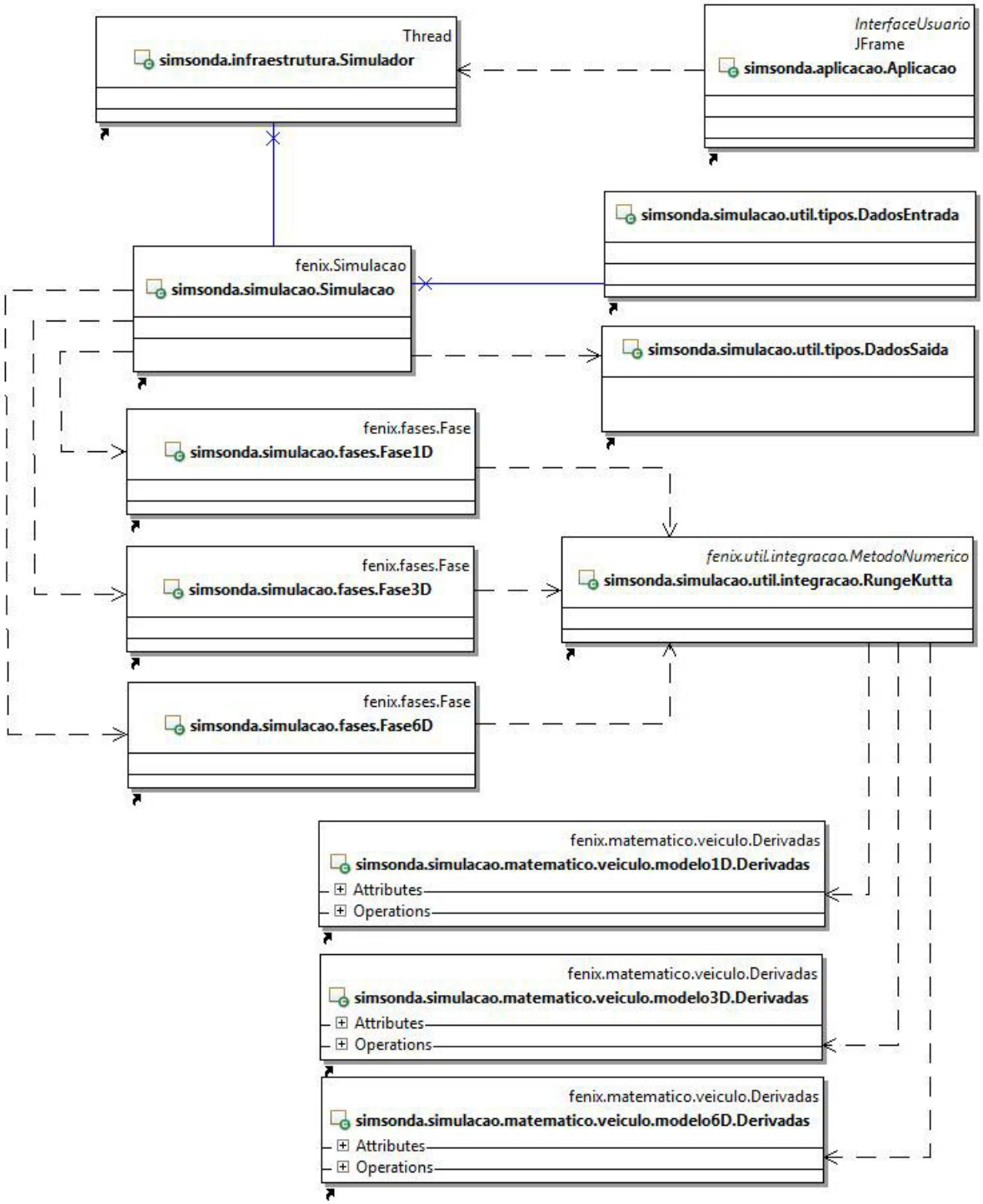

FIGURA G.19 – Diagrama de Classes do SIM-CDU02

A Figura G.20 apresenta as classes utilizadas para a realização do SIM-CDU03 - Visualizar Trajetória do Foguete. Outras classes utilizadas para gerar tabelas, gráficos e

caixas de textos, e que não fazem parte dos pacotes *fenix* e *simsonda*, foram omitidas neste diagrama por razão de simplicidade.

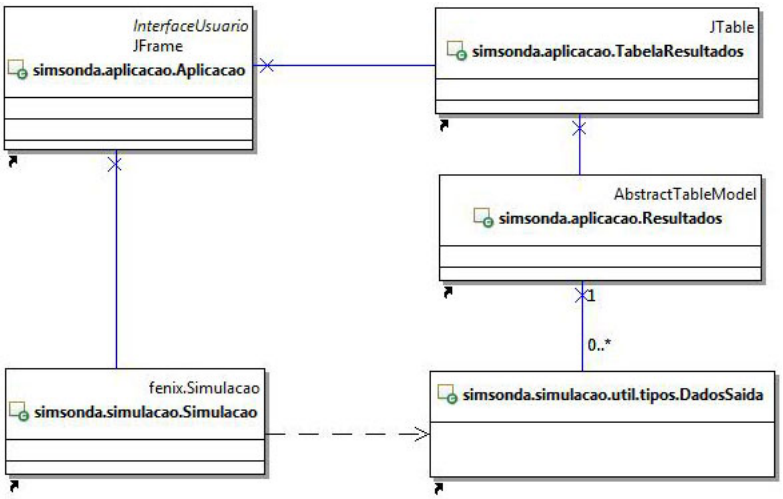

FIGURA G.20 – Diagrama de Classes do SIM-CDU03

Para permitir a realização do caso de uso SIM-CDU04 - Compartilhar Dados de Visualização, visando a criação de uma Federação de Visualização Remota de Trajetórias, as principais classes do pacote *hla.federacao* foram relacionadas conforme apresentado a Figura G.21. Os Federados utilizam a interface *RTIAmbassador* quando precisam invocar os serviços da RTI. Esta por sua vez utiliza a classe *FedAmbImpl* para invocar métodos dos Federados, sendo esta classe uma implementação da interface *FederateAmbassador*.

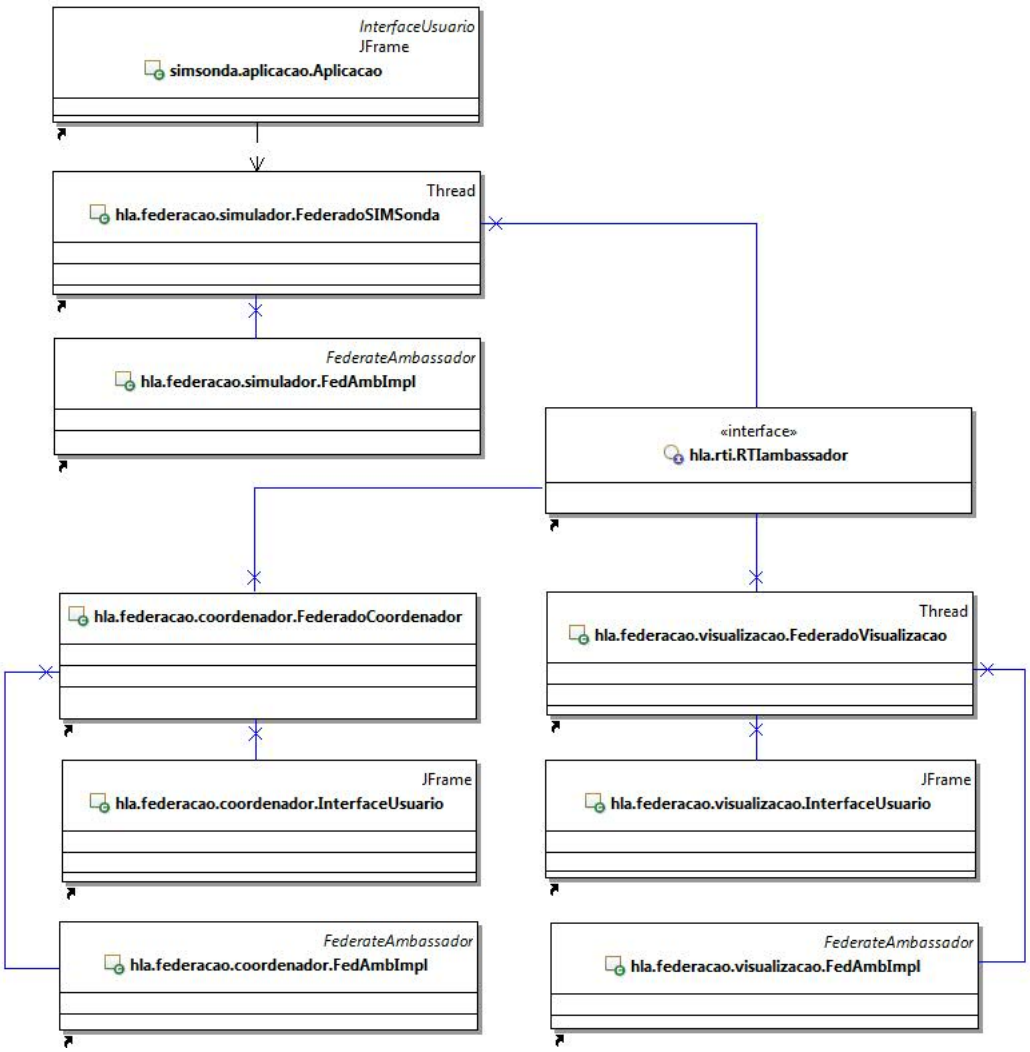

FIGURA G.21 – Diagrama de classe para o SIM-CDU04

Apêndice H - Diagrama de Sequência do Protótipo SIMSonda

H.1 Introdução

Este documento apresenta um diagrama de seqüência do Simulador SIMSonda, referente ao fluxo de execução de sua infra-estrutura de execução, como especificado pelo caso de uso SIM-CDU02 - Simular Lançamento. Além de três diagramas relacionados aos Federados Sistema Coordenador, Sistema de Visualização e Simulador SIMSonda, referentes ao ciclo de vida destes Federados durante a execução de uma Federação, como especificado no caso de uso SIM-CDU04.

H.2 Diagramas de Sequência

Esta Seção apresenta os diagramas de sequência elaborados para o SIMSonda e os Federados Coordenador, Visualização e SIMSonda, além de descrever os eventos enumerados em cada diagrama.

H.2.1 Infra-estrutura de Execução

O diagrama de seqüência observado na Figura H.1 mostra os principais eventos ocorridos no método *run()*, da classe *simsonda.infraestrutura.Simulador*. Este método solicita o avanço de tempo da simulação em intervalos de tempo fixos para cada fase executada.

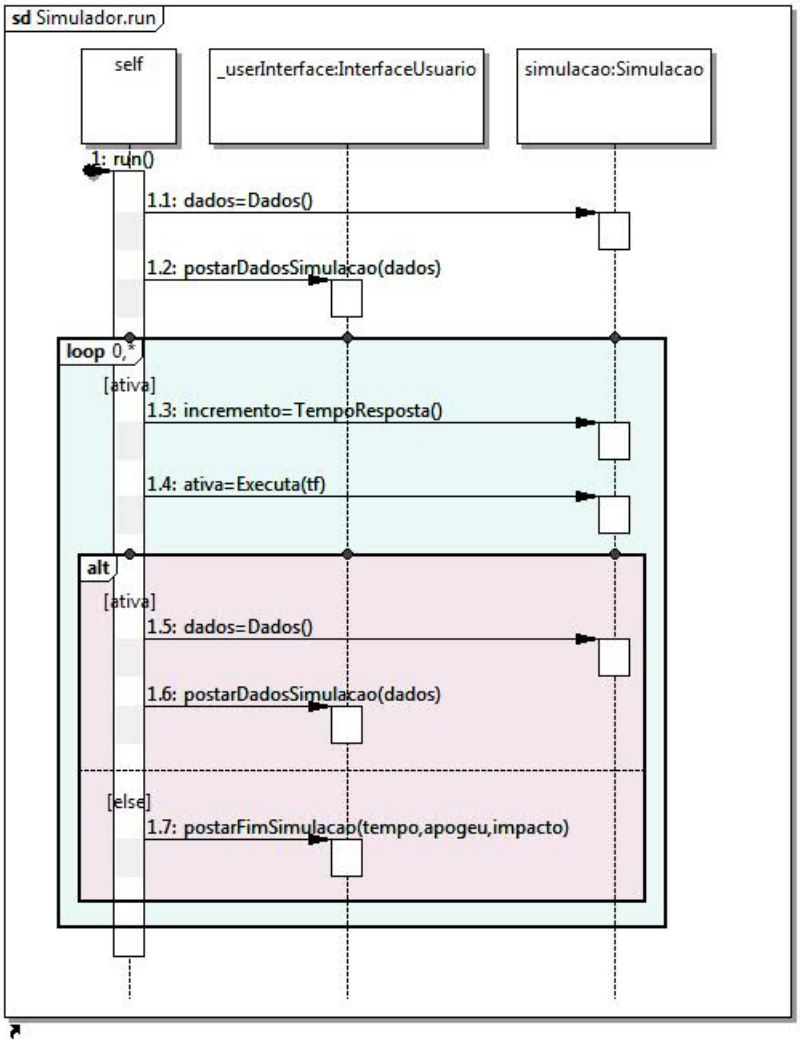

FIGURA H.1 – Diagrama de sequência para infra-estrutura padrão

H.2.1.1 Descrição dos Eventos

Os eventos que ocorrem no método *run()*, conforme mostrados na Figura H.1, são descritos a seguir:

1.1 - O método *run()* consulta os dados iniciais relacionados a trajetória do foguete;

1.2 - Estes dados iniciais são apresentados na interface gráfica com o usuário;

Repetição - Enquanto não ocorrer o término da simulação, os Eventos 1.3 a 1.7 são continuamente repetidos. O fim da repetição acontece quando a variável *ativa* tiver valor igual a falso;

1.3 - A variável *incremento* recebe o valor do tempo de resposta definido para a fase atual. Determina-se o novo tempo desejado para a simulação incrementando o tempo atual de simulação com o valor desta variável;

1.4 - O método *run()* solicita o avanço do tempo da simulação para o novo tempo desejado. O fluxo de execução do método *Executa()* foi descrito no diagrama de sequência apresentado no Apêndice D. O retorno do método *Executa()* é armazenado na variável *ativa* do método *run()*, sendo o valor verdadeiro se a simulação ainda não terminou;

Condição - O método *run()* testa o valor da variável *ativa*. Caso a simulação ainda não tenha terminado, executa-se os Eventos 1.5 e 1.6; caso contrário, executa-se o passo 1.7 antes do término da repetição;

1.5 - O método *run()* consulta os dados atuais relacionados a trajetória do foguete;

1.6 - Estes dados são apresentados na interface gráfica com o usuário; e

1.7 - O método *run()* exibe uma mensagem na interface gráfica para informar ao usuário o término da simulação.

H.2.2 Federado Sistema Coordenador

O diagrama de seqüência mostrado na Figura H.2 apresenta o fluxo principal de execução do Federado Coordenador, implementado no método *mainThread()*, da classe *hla.federacao.coordenador.FederadoCoordenador*. Este Federado registra os três pontos de sincronismo e controla a quantidade de Federados membros para a execução da Federação. Após o término da simulação, ele aguarda a saída dos demais federados, pois tem como responsabilidade destruir a execução da Federação na RTI.

O primeiro Federado que se connecta a RTI deve criar a Federação, fornecendo o arquivo com o *Federation Object Model (FOM)* como parâmetro ao método correspondente na interface *RTIAmbassador*. Os demais Federados ao se conectarem também tentarão criar a Federação, mas receberão como resposta da RTI que a Federação já existe. A interface *RTIAmbassador* permite aos Federados solicitarem a execução dos serviços oferecidos pela RTI, como avanço de tempo, atualização de valores de atributos, envio de interações, dentre outros.

Para se comunicar com os Federados, a RTI utiliza os métodos da interface *FederateAmbassador* implementados para cada Federado em suas classes *FedAmbImpl*. Quando ocorre a invocação de um método desta interface, o Federado insere a invocação no final de uma estrutura de dados do tipo fila, definida na classe *CallbackQueue*. Esta estrutura armazena as invocações como objetos na forma definida pela interface *Callback*.

Para construir tipos diferentes de respostas basta herdar a classe *Callback* adicionando o comportamento específico. A interface *Callback* define um método, chamado *dispatch()*, onde deve ser implementado como um Federado responde a cada invocação em particular. Por exemplo, o Federado Sistema de Visualização implementa esta interface numa classe

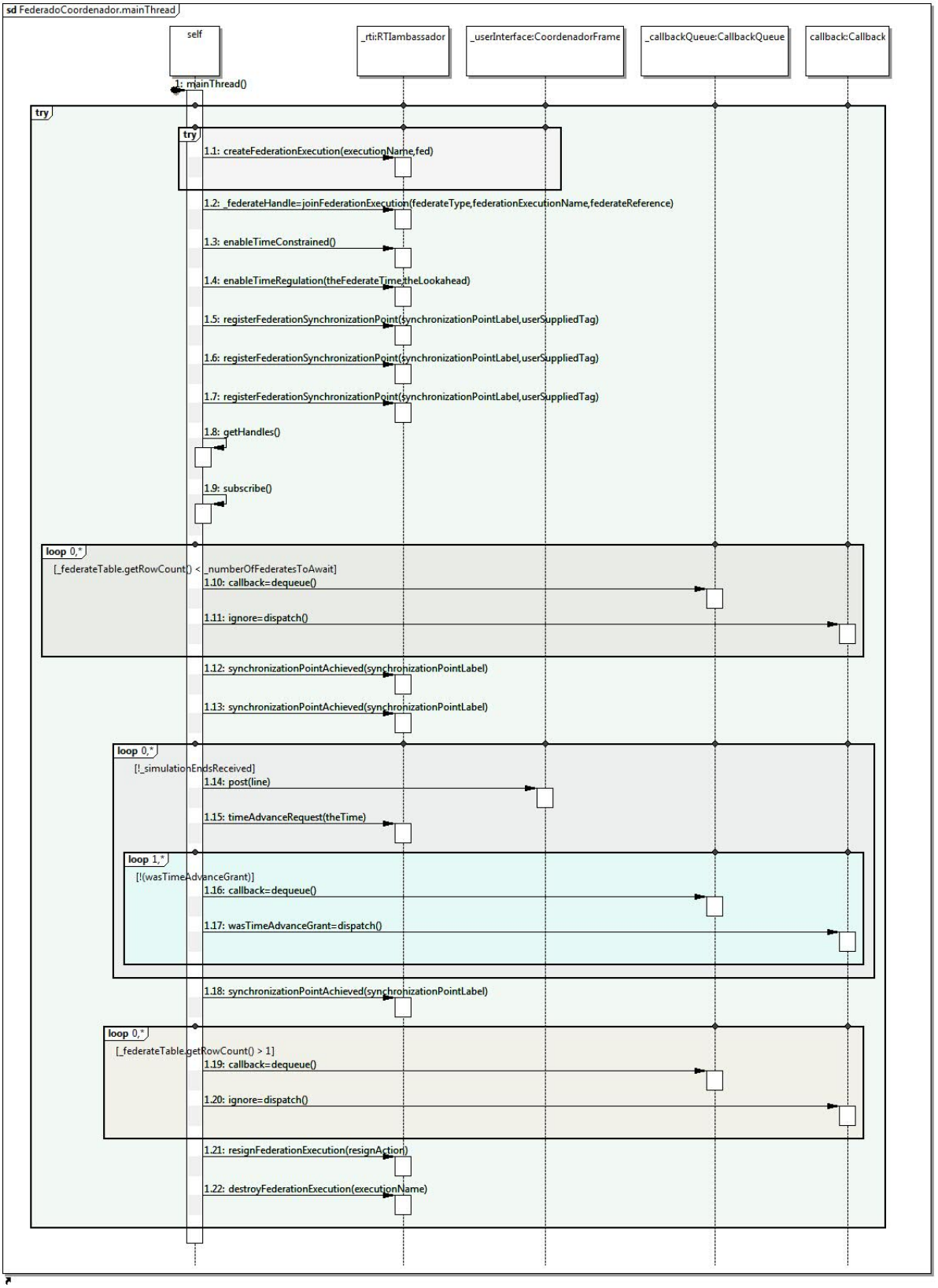

FIGURA H.2 – Diagrama de sequência para o Federado Coordenador

interna, chamada *ReflectAttributeValueCallback*. Este Federado cria uma instância desta classe, e a insere no final da fila de invocações, cada vez que a RTI invoca o método da interface *FederateAmbassador* referente a divulgação de novos valores dos atributos. O método *dispatch()* desta classe salva os novos valores recebidos num objeto mantido pelo Federado Visualização. Este método será executado quando o Federado estiver processando as invocações armazenadas na fila.

A Federação de Visualização Remota de Trajetórias é totalmente sincronizada, pois todos os Federados são restringidos e reguladores de tempo, como descrito no Anexo A. Os Federados solicitam o avanço do seu tempo lógico e enquanto aguardam a aprovação da RTI, eles recebem e processam todas as *Callbacks* com estampas de tempo menores que o instante de tempo solicitado. Neste esquema de gerenciamento de tempo, a RTI permite o avanço de tempo do federando somente quando todas as mensagens com estampas de tempo menores que o instante de tempo solicitado já tenham sido entregues. Uma vez permitido o avanço de tempo, os Federados não connseguem mais enviar mensagens com estampas menores que o tempo atual.

H.2.2.1 Descrição dos Eventos

Os eventos que ocorrem no método *mainThread()*, conforme mostrados na Figura H.2, são descritos a seguir:

1.1 - O Federado Coordenador cria a Federação na RTI, fornecendo o nome da Federação e o arquivo com os dados de entrada. Caso a simulação já tenha sido criada por outro Federado ocorre uma exceção;

1.2 - O Federado Coordenador solicita a RTI para ingressar na Federação, informando a

identificação do Federado, o nome da Federação e a instância da classe *FedAmbImpl* deste Federado. O seu estado atual é *Pré-Processamento*;

1.3 - O Federado Coordenador informa a RTI que é restringido por tempo;

1.4 - O Federado Coordenador informa a RTI que é regulador de tempo na Federação, fornecendo o valor do seu tempo lógico e do *lookahead*;

1.5 - O Federado Coordenador registra o primeito ponto de sincronismo da Federação, informando seu nome como *ProntoParaDivulgar*;

1.6 - O Federado Coordenador registra o segundo ponto de sincronismo, chamado *ProntoParaExecutar*;

1.7 - O Federado Coordenador registra o terceiro ponto de sincronismo, denominado *ProntoParaSair*;

1.8 - O Federado Coordenador solicita os identificadores definidos pela RTI para os objetos e atributos relativos aos Federados membros da Federação, como nome e endereço de rede, além do identificador da interação de fim da simulação;

1.9 - O Federado Coordenador se inscreve nos atributos e na interação de interesse, fornecendo seus identificadores;

Repetição - Os Eventos 1.10 e 1.11 são repetidos enquanto o número de Federados membros da Federação for menor que o valor definido para esta Federação;

1.10 - O Federado Coordenador remove uma invocação da fila de invocações da RTI;

1.11 - O Federado Coordenador processa a invocação, referente à descoberta de um novo Federado membro da Federação ou a atualização de atributos relacionados a algum Federado;

1.12 - O Federado Coordenador informa que alcançou o ponto de sincronização *Pronto-ParaDivulgar*, uma vez que já terminou o *Pré-Processamento*. O estado atual é *Esperando ProntoParaDivulgar*. O Federado utiliza a classe *hla.federacao.util.Barrier* para suspender a sua execução até receber a invocação da RTI informando que a Federação encontra-se sincronizada neste ponto;

1.13 - O Federado Coordenador informa a RTI que alcançou o ponto de sincronismo *ProntoParaExecutar*. Como o Federado não publica nenhum valor de atributo, no estado *Divulgando* o Federado não executa nada, passando diretamente para o estado *Esperando ProntoParaExecutar*. Ao ser informado pela RTI que a Federação se encontra sincronizada neste ponto, o Federado assume o estado *Executando*;

Repetição - Os Eventos 1.14 a 1.17 são repetidos enquanto a simulação não terminar;

1.14 - O Federado Coordenador apresenta na interface gráfica com o usuário uma linha informando o novo tempo desejado para a simulação. Calcula-se este tempo incrementando o tempo lógico atual com um valor fixo pré-determinado para este Federado;

1.15 - O Federado Coordenador solicita a RTI o avanço do seu tempo lógico para o tempo desejado.

Repetição - Os Eventos 1.16 e 1.17 são repetidos enquanto o Federado não receber a permissão da RTI para avançar seu tempo lógico;

1.16 - O Federado Coordenador remove uma invocação da fila de invocações da RTI;

1.17 - O Federado Coordenador processa a invocação, referente a permissão para avanço de tempo, ou informando o fim da simulação;

1.18 - O Federado Coordenador informa que alcançou o ponto de sincronismo *ProntoParaSair*, uma vez que recebeu a interação informando o fim da simulação. Como o estado *Executando* terminou, o Federado assume o estado *Esperando ProntoParaSair*, sendo suspenso até que a RTI informe que a Federação encontra-se sincronizada neste ponto. Em seguida, o Federado passa para o estado *Saindo*;

Repetição - Os Eventos 1.19 e 1.20 são repetidos até que o Federado Coordenador seja o único membro da Federação;

1.19 - O Federado Coordenador remove novamente uma invocação da fila de invocações da RTI;

1.20 - O Federado Coordenador executa o tratamento da invocação, referente a informação da remoção de um Federado da Federação;

1.21 - O Federado Coordenador informa a RTI que está deixando a Federação; e

1.22 - O Federado Coordenador destroi a execução da Federação na RTI.

H.2.3 Federado Sistema de Visualização

O diagrama de seqüência mostrado na Figura H.3 apresenta o fluxo principal de execução do Federado Visualização, implementado no método *run()*, da classe *hla.federacao.visualizacao.FederadoVisualizacao*. Este Federado não publica nenhum atributo na Federação, seu propósito é apenas exibir as atualizações dos atributos da classe de objetos *ObjectRoot.Veiculo*.

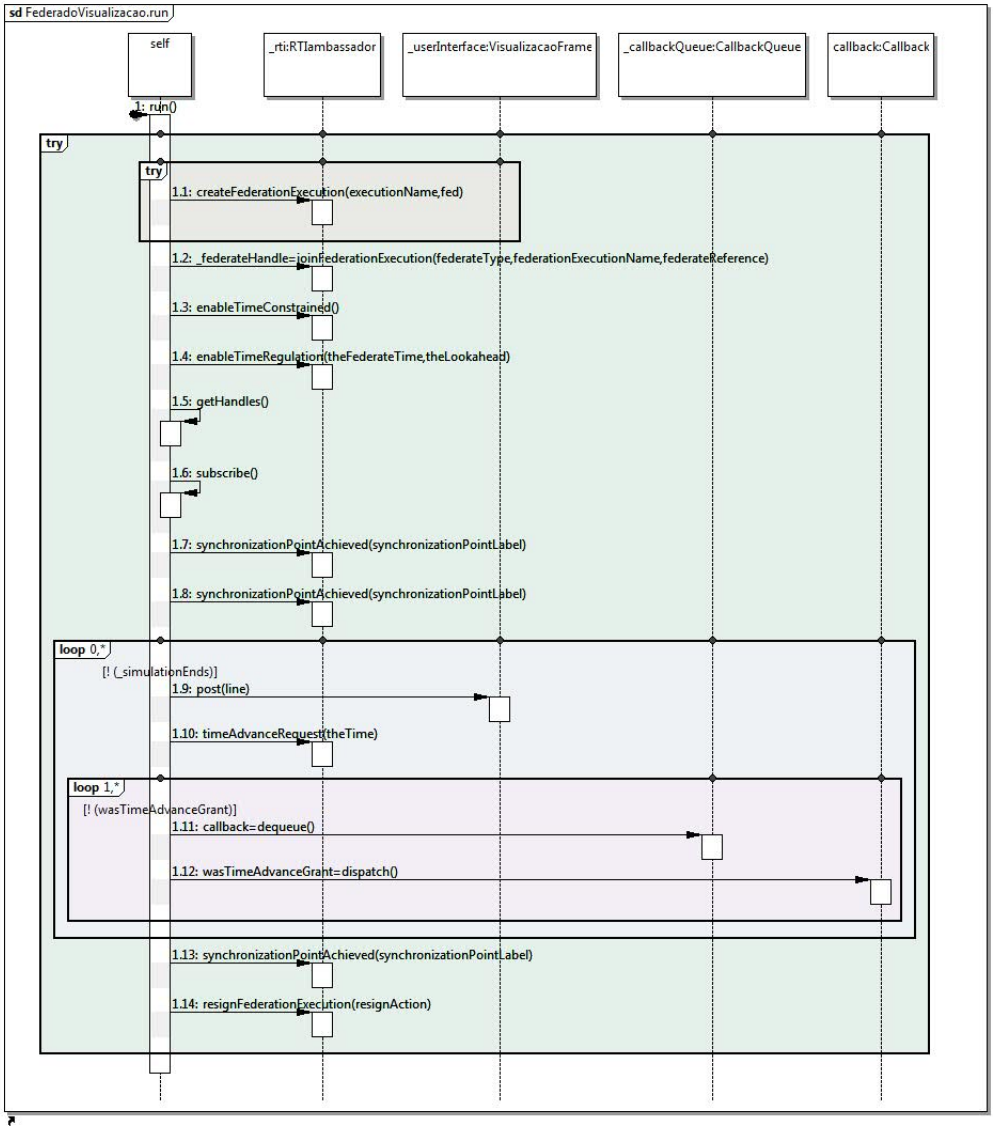

FIGURA H.3 – Diagrama de sequência para Federado Visualização

H.2.3.1 Descrição dos Eventos

Os eventos que ocorrem no método *run()*, conforme mostrados na Figura H.3, são descritos a seguir:

1.1 - O Federado Visualização cria a Federação na RTI, fornecendo o nome da Federação e o arquivo com os dados de entrada. Caso ela já tenha sido criada por outro Federado ocorre uma exceção;

1.2 - O Federado Visualização solicita a RTI para ingressar na Federação, informando

sua identificação, o nome da Federação e a sua instância da classe FedAmbImpl. O seu estado atual é *Pré-Processamento*;

1.3 - O Federado Visualização informa a RTI que é restringido por tempo;

1.4 - O Federado Visualização informa a RTI que é regulador de tempo na Federação;

1.5 - O Federado Visualização solicita os identificadores dos atributos do objeto *Veiculo* e da interação de fim da simulação;

1.6 - O Federado Visualização se inscreve nos atributos e na interação de interesse, fornecendo seus identificadores;

1.7 - O Federado Visualização informa que alcançou o ponto de sincronização *ProntoParaDivulgar*, uma vez que já terminou o *Pré-Processamento*. O seu estado atual é *Esperando ProntoParaDivulgar*. A execução do Federado é suspensa até receber a invocação da RTI informando que a Federação está sincronizada neste ponto;

1.8 - O Federado Visualização informa a RTI que alcançou o ponto de sincronismo *ProntoParaExecutar*. Como o Federado não publica nenhum valor de atributo, no estado *Divulgando* o Federado não executa nada, passando diretamente para o estado *Esperando ProntoParaExecutar*. Ao ser informado pela RTI que a Federação se encontra sincronizada neste ponto, o Federado assume o estado *Executando*;

Repetição - Os Eventos 1.9 a 1.12 são repetidos enquanto a simulação não terminar;

1.9 - O Federado Visualização apresenta na interface gráfica com o usuário uma linha informando o tempo desejado para a simulação. Calcula-se este tempo incrementando o tempo lógico atual de um valor fixo pré-determinado para este Federado;

1.10 - O Federado Visualização solicita a RTI o avanço do seu tempo lógico para o tempo desejado;

Repetição - Os Eventos 1.11 e 1.12 são repetidos enquanto o Federado não receber a permissão da RTI para avançar seu tempo lógico;

1.11 - O Federado Visualização remove uma invocação da fila de invocações da RTI;

1.12 - O Federado Visualização processa a invocação, referente a atualização dos atributos do objeto *Veiculo*, ou a permissão para avanço de tempo, ou ainda a informação de fim da simulação. No tratamento da atualização de atributos, os novos valores são apresentados na interface com o usuário numa tabela, sendo o gráfico da altitude do foguete atualizado com a atualização da altitude;

1.13 - O Federado Visualização informa que alcançou o ponto de sincronismo *ProntoParaSair*, uma vez que recebeu a interação informando o fim da simulação. Como o estado *Executando* terminou, o Federado assume o estado *Esperando ProntoParaSair*, sendo suspenso até a RTI confirmar que a Federação encontra-se sincronizada neste ponto; e

1.14 - O Federado Visualização informa a RTI que está deixando a Federação ao assumir o estado *Saindo*.

H.2.4 Federado Simulador SIMSonda

O diagrama de seqüência apresentado na Figura H.4 apresenta o fluxo principal de execução do Federado SIMSondaa, implementado no método *run()*, da classe *hla.federacao.simulador.FederadoSIMSonda*. Este Federado não está inscrito em nenhum atributo

ou interação, sendo responsável por publicar os atributos da classe *ObjectRoot.Veiculo* e enviar a interação *InteractionRoot.SimulationEnds*.

H.2.4.1 Descrição dos Eventos

Os eventos que ocorrem no método *run()*, conforme mostrados na Figura H.4, são descritos a seguir:

1.1 - O Federado SIMSonda cria a Federação na RTI. Caso ela já tenha sido criada por outro Federado ocorre uma exceção;

1.2 - O Federado SIMSonda solicita a RTI para ingressar na Federação, informando sua identificação, o nome da Federação e a sua instância da classe FedAmbImpl. O seu estado atual é *Pré-Processamento*;

1.3 - O Federado SIMSonda informa a RTI que é restringido por tempo;

1.4 - O Federado SIMSonda informa a RTI que é regulador de tempo na Federação, fornecendo o valor do seu tempo lógico e do *lookahead*;

1.5 - O Federado SIMSonda solicita os identificadores dos atributos do objeto *Veiculo* e da interação de fim da simulação;

1.6 - O Federado SIMSonda informa que irá publicar os atributos do objeto *Veiculo* e a interação *SimulationEnds*, fornecendo seus identificadores;

1.7 - O Federado SIMSonda informa que alcançou o ponto de sincronização *ProntoParaDivulgar*, uma vez que já terminou o *Pré-Processamento*. O seu estado atual é *Esperando ProntoParaDivulgar*. A execução do Federado é suspensa até a RTI confirmar que a Federação encontra-se sincronizada neste ponto;

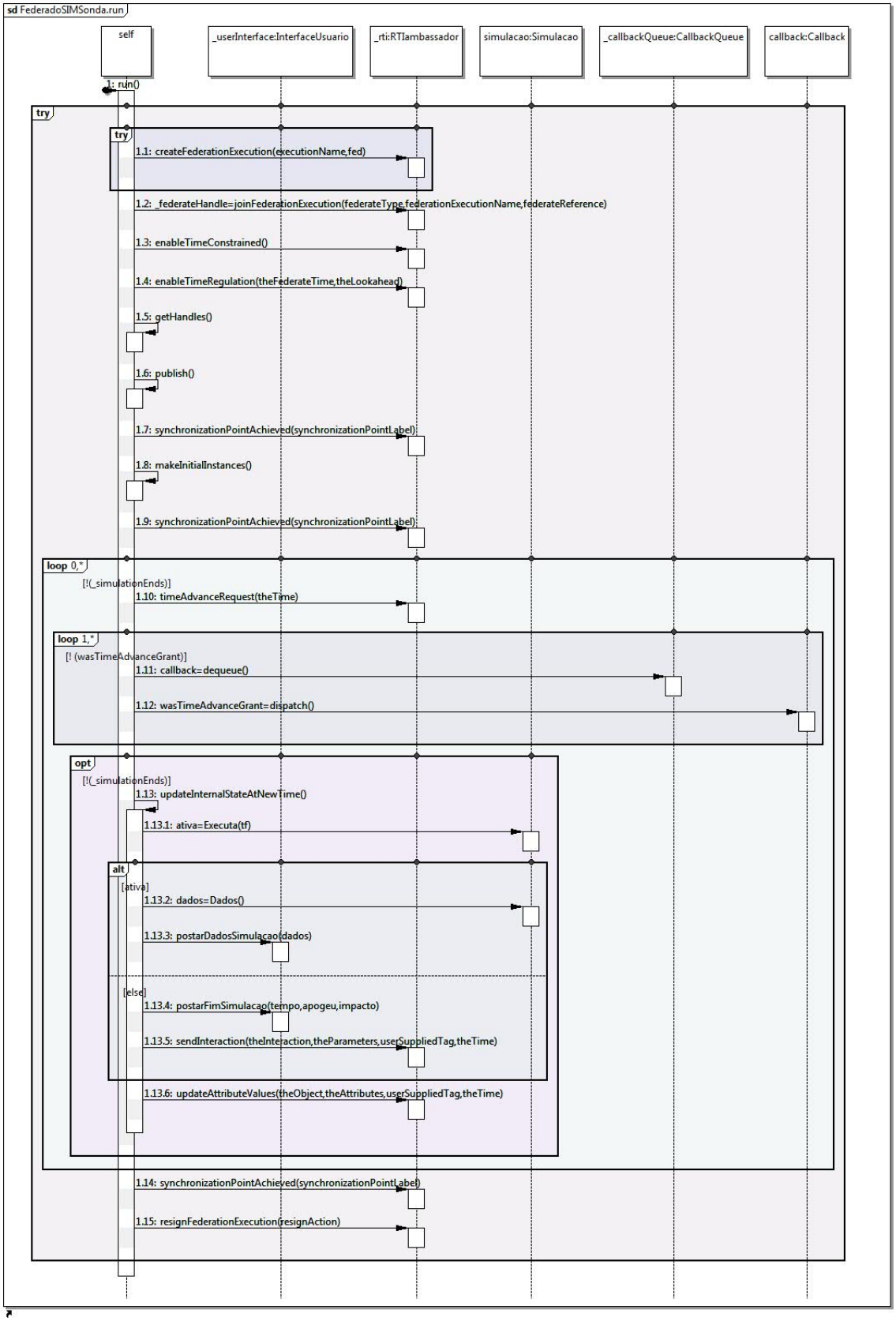

FIGURA H.4 – Diagrama de sequência para Federado SIMSonda

1.8 - O Federado SIMSonda registra a instância do objeto *Veiculo*, definindo os valores iniciais de seus atributos. Este valores são postados na interface gráfica com o usuário. Este Evento corresponde ao estado *Divulgando*;

1.9 - O Federado SIMSonda informa a RTI que alcançou o ponto de sincronismo *ProntoParaExecutar*. O Federado assume o estado *Esperando ProntoParaExecutar*. Ao ser informado pela RTI que a Federação se encontra sincronizada neste ponto, o Federado assume o estado *Executando*;

Repetição - Os Eventos 1.10 a 1.13 são repetidos enquanto a simulação não terminar;

1.10 - O Federado SIMSonda solicita a RTI o avanço do seu tempo lógico para um valor determinado incrementando-se o tempo de simulação com o tempo de resposta.

Repetição - Os Eventos 1.11 e 1.12 são repetidos enquanto o Federado não receber a permissão da RTI para avançar seu tempo lógico;

1.11 - O Federado SIMSonda remove uma invocação da fila de invocações da RTI;

1.12 - O Federado SIMSonda processa a invocação, referente a permissão para avanço do seu tempo lógico;

Condição - O Federado SIMSonda verifica se ocorreu o fim da simulação. Caso faso, executa-se o Evento 1.13; caso contrário, executa-se o Evento 1.14;

1.13 - O Federado SIMSonda atualiza seu o estado interno para o novo tempo lógico. Isto é realizado através dos Eventos 1.13.1 a 1.13.6;

1.13.1 - O Federado SIMSonda avança o tempo de simulação da trajetória do foguete para o novo tempo lógico. Caso o Simulador SIMSonda constate o fim da simulação, retorna falso na variável *ativa*; caso contrário, retorna verdadeiro;

Condição - O Federado SIMSonda testa o valor da variável *ativa*. Caso verdadeiro, executa-se os Eventos 1.13.2 e 1.13.3; caso contrário, executa-se os Eventos 1.13.4 e 1.13.5;

1.13.2 - O Federado SIMSonda consulta os dados da trajetória do foguete;

1.13.3 - O Federado SIMSonda apresenta os dados na interface gráfica com o usuário e, em seguida, avança para o Evento 1.13.6;

1.13.4 - O Federado SIMSonda exibe uma mensagem na interface gráfica para informar ao usuário o término da simulação, apresentando o tempo final daa simulação e dados sobre o apogeu e o ponto de impacto do foguete;

1.13.5 - O Federado SIMSonda publica uma interação para informar aos outros Federados membros o término da execução da Federação, fornecendo o identificador da interação e o tempo em que ela ocorreu;

1.13.6 - O Federado SIMSonda solicita a RTI a publicação dos novos valores calculados para a trajetória do foguete, fornecendo o tempo atual, os identificadores do objeto *Veiculo* e dos seus atributos e os novos valores destes atributos;

1.14 - O Federado SIMSonda informa que alcançou o ponto de sincronismo *ProntoParaSair*. Como o estado *Executando* terminou, o Federado assume o estado *Esperando ProntoParaSair*, sendo suspenso até que a RTI confirme que a Federação está sincronizada neste ponto; e

1.15 - O Federado SIMSonda informa a RTI que está deixando a Federação ao assumir o estado *Saindo*.

Anexo A - Detalhamento do Padrão IEEE STD 1516

Desde a década de 80, durante a Guerra Fria, o Ministério de Defesa Americano (*U. S. Department of Defense - DoD*) já desenvolvia inúmeras simulações militares num cenário de guerra para treinar e preparar seus oficiais na tomanda de decisão na resolução de problemas e execução de ações complexas (KUHL; WEATHERLY; DAHMANN, 1999).

As primeiras simulações militares distribuídas para ambientes virtuais começaram na década de 80 com o projeto denominado *"SIMulator NETworking (SIMNET)"*, mais tarde denominado *Distributed Interactive Simulation (DIS)*. Neste projeto definiram-se os padrões para suportar a interoperabilidade entre os simuladores de treinamento autônomos em ambientes distribuídos geograficamente (TRIVELATO, 2003). O segundo maior impulso a partir do SIMNET foi o protocolo *Aggregate Level Simulation Protocol (ALSP)*, que expandiu o conceito de interoperabilidade para simulações de jogos de guerra.

Entretanto, por causa dos custos surgiu a necessidade de assegurar a interoperabilidade entre simulações construídas por diferentes organizações, e a reusabilidade dos componentes de simulação para vários propósitos.

Para atingir este objetivo foi necessária a criação de um padrão de arquitetura de simu-

lação. O maior avanço teve a sua origem no desenvolvimento do *Modeling & Simulation High Level Architecture Master Plan*, inicialmente denominado *M&S HLA*, criado pela *Defense Advanced Research Projects Agency (DARPA)*, que definiu os objetivos básicos desejáveis para a área de simulação no DoD.

O HLA foi definido como um padrão internacional em 2000, denominado *IEEE STD 1516* (IEEE, 2000a; IEEE, 2000b; IEEE, 2000c; IEEE, 2003). O padrão HLA tornou-se importante por prover uma arquitetura única que engloba tanto as simulações analíticas como as de ambiente virtuais.

Na especificação do padrão definiu-se os serviços (em ambas as direções) como chamadas de procedimentos com parâmetros ou argumentos, valores de retorno, pré e pós-condições e exceções. A definição dos serviços na especificação da interface não faz nenhuma referência a qualquer linguagem de programação, apenas apresenta interfaces de programação de aplicação (*Application Program Interface - API*) em várias linguagens de programação.

É importante salientar que o HLA especifica uma abordagem para problemas persistentes na simulação de Federações, evitando a definição desta abordagem em termos de tecnologias transitórias. O HLA irá se manter no ritmo do surgimento de novas tecnologias, definindo APIs para novas linguagens e incorporando, na implementação da RTI, novas tecnologias para comunicação e coordenação de sistemas distribuídos.

As próximas Seções descrevem os três elementos principais especificados no padrão HLA: a interface de comunicação entre Federados e a RTI, o gabarito para declaração de modelos de objetos HLA, e as regras definindo Federação e Federado.

A.1 Especificação da Interface

O padrão HLA requer que as comunicações entre Federados usem uma API padrão, provendo uma especificação para as interfaces funcionais entre os Federados e a RTI. Esta espeficicação é proposta como uma API em diferentes formas incluíndo CORBA IDL, C++, Ada95 e Java (BUSS; JACKSON, 1998).

Qualquer *software* agindo como uma RTI deve implementar todos os serviços fornecidos aos Federados de acordo com a especificação da interface. Da mesma forma, qualquer *software* agindo como um Federado deve implementar todos os serviços invocados pela RTI de acordo com a especificação da interface.

A seguir serão descritos detalhadamente os setes grupos de serviços definidos na especificação da interface de acordo com o padrão IEEE (2000c):

A.1.1 Gerenciamento de Federação

Este grupo de serviços permite a criação, o controle dinâmico, a modificação e a finalização de uma execução de Federação. Os serviços possibilitam a inclusão ou exclusão de simulações (Federados) numa Federação existente (STEELE *et al.*, 2002). Esse gerenciamento pode ser observado através da Figura A.1, que mostra os estados possíveis de uma execução de Federação e como certos serviços de gerenciamento de Federação são empregados.

Inicialmente tem-se o estado “Execução da Federação Não Existe”, representando que a RTI teve a execução iniciada, mas que nenhum serviço relativo à execução da Federação foi ainda invocado.

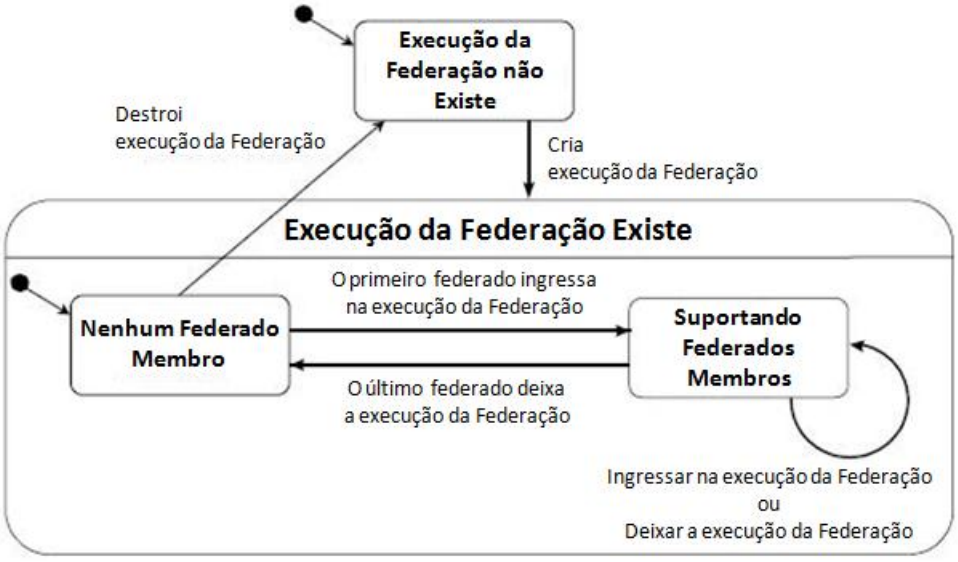

FIGURA A.1 – Estados básicos da execução de Federação. (Adaptada de IEEE, 2000c)

Após a criação da execução de uma Federação, ela passa para o estado "Nenhum Federado Membro". Nesse estado nenhum Federado ainda ingressou na Federação, nenhuma instância de objetos existe e nenhuma mensagem está enfileirada.

Assim que o primeiro Federado se torna membro, a Federação executa até o momento em que o último Federado abandona a execução. A Federação se encontra no estado "Suportando Federados Membros", que representa uma execução operacional da Federação, na qual os serviços documentados na especificação da interface estão disponíveis aos Federados.

Uma vez que a execução da Federação existe, Federados podem se tornar membros e abandoná-la em qualquer seqüência que é significante para o criador da Federação. Após o último Federado abandonar a execução da Federação, ela retorna para o estado "Nenhum Federado Membro".

A.1.2 Gerenciamento de Declaração

Os Federados podem usar os serviços de gerenciamento de declaração para declarar a sua intenção de produzir e de receber informação. Um Federado deve invocar o serviço apropriado de declaração antes de registrar ou descobrir instâncias de objeto, atualizar ou refletir valores de atributos de instância, e enviar ou receber interações (IEEE, 2000c).

Uma demonstração do uso desses serviços é apresentada na Figura A.2, onde os Federados A e B estão declarando as suas intenções de gerar e receber informações. O Federado A declara a intenção de publicar os dados de uma instância de objeto, e também se inscreve numa classe de interação publicada por B. O Federado B, por sua vez, declara a inteção de publicar essa classe de interação e também se inscreve em atributos da instância da classe do objeto publicado por A.

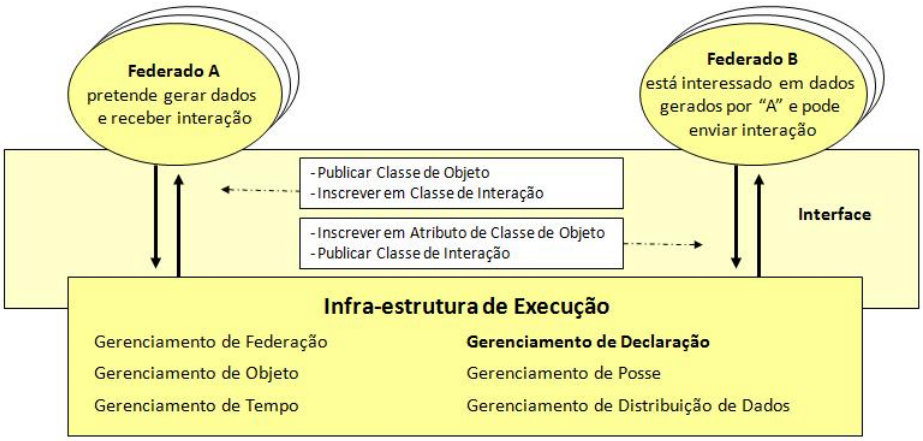

FIGURA A.2 – Exemplo do gerenciamento de declaração. (Adaptada de KOMPETENZ-ZENTRUM HLA, 2003)

A.1.3 Gerenciamento de Objeto

Os serviços deste grupo possibilitam a troca de dados entre Federados, o controle da forma de transporte dos dados, a solicitação de novos valores de atributos, e a confirmação se um Federado deve aguardar por dados (KUHL; WEATHERLY; DAHMANN, 1999).

Um Federado solicita os serviços desse grupo a RTI para registrar novas instâncias de objeto, atualizar valores de atributos de uma instância e enviar interações. Por outro lado, um Federados sofrerá a invocação em sua interface, por parte da RTI, de serviços desse grupo para a descoberta de novas instâncias dos objetos em que se inscreveu, para refletir as atualizações dos valores desses atributos e para receber as interações em que se inscreveu anteriormete.

A Figura A.3 ilustra como alguns desses serviços são utilizados para a troca de dados entre dois Federados A e B. O Federado A solicita a RTI o identificador do objeto que ele declarou a intenção de publicá-lo. Em seguida, o Federado registra a instância do objeto, para então atualizar os valores dos seus atributos, divulgando seus valores iniciais.

FIGURA A.3 – Exemplo do gerenciamento de objeto. (Adaptada de KOMPETENZZENTRUM HLA, 2003)

Observa-se na Figura A.3 que os serviços utilizados pelo Federado A provocam a invocação, por parte da RTI, de métodos na interface do Federado B, pois ele anteriormente se inscreveu em atributos da instância do objeto publicado por A. O registro da instãncia causou a invoção de um serviço de descoberta em B, informando que a instância do objeto já se encontra disponível. Além disso, a atualização de valores dos atributos por A causou a invocação de um serviço na interface de B, permitindo a ele refletir internamente a atualização desses valores.

A.1.4 Gerenciamento de Posse

Os Federados e a RTI usam os serviços deste grupo para transferir a posse dos atributos de uma instância entre Federados. Essa habilidade é requerida para suportar a modelagem cooperativa de uma determinada instância de objeto numa Federação. Apenas o Federado proprietário de um atributo de uma instância pode invocar o serviço para a atualização do seu valor, ou receber uma solicitação de atualização desse valor através do serviço apropriado invocado pela RTI (IEEE, 2000c).

No exemplo apresentado na Figura A.4, como o Federado A não deseja mais atualizar determinados atributos, ele então requisita a RTI a renúncia da posse deles. O Federado B, por outro lado, tem interesse em prover a atualização desses atributos e ao receber, da RTI, a informação do pedido de renúncia de A, o Federado B então requisita a RTI a posse deles. Em seguida, a RTI notifica o Federado A que ele não é mais proprietário dos atributos e, após isso, confirma ao Federado B que ele agora tornou-se responsável pela manutenção dos valores desses atributos.

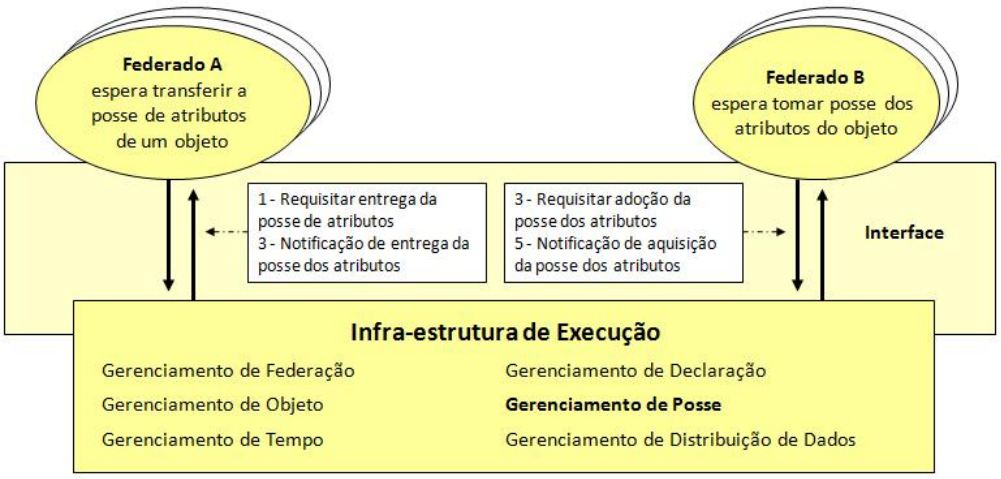

FIGURA A.4 – Exemplo do gerenciamento de posse. (Adaptada de KOMPETENZZENTRUM HLA, 2003)

A.1.5 Gerenciamento de Tempo

Como durante a execução de uma Federação os Federados executam de acordo com os seus próprios fluxos de controle, a ordenação adequada de eventos é um problema significante a ser resolvido. No padrão HLA, a ordem de eventos se expressa num "tempo lógico", consistindo numa noção abstrata e necessariamente independente de qualquer representação ou unidade de tempo. Os serviços de gerenciamento de tempo da RTI têm duas funções (KUHL; WEATHERLY; DAHMANN, 1999):

- Permitir cada Federado avançar seu tempo lógico em coordenação com outros Federados; e
- Controlar a entrega de eventos com estampas de tempo *(time stamp)*, de modo que um Federado nunca receba eventos de outros Federados no "passado", i.e., eventos com tempo lógico menor do que o seu.

A RTI permite a um Federado escolher o grau no qual participa do gerenciamento de tempo. Um Federado pode ser restringido por tempo, no caso em que o avanço do seu tempo lógico é restringido pelo resto da Federação, ou pode ser regulador de tempo, onde o avanço do seu tempo lógico regula o resto da Federação.

Essas duas escolhas ocorrem de forma independente para os Federados, resultando em quatro possibilidades, conforme ilustrado na Figura A.5.

Os Federados escolhem seu nível de participação dependendo do propósito e dos requisitos da Federação. A seguir são descritas as conseqüências possíveis para cada uma dessas escolhas (KUHL; WEATHERLY; DAHMANN, 1999):

- Um Federado não regulador de tempo e nem restringido simplemente não participa

FIGURA A.5 – Escolhas para o gerenciamento de tempo. (Adaptada de KUHL; WEATHERLY; DAHMANN, 1999)

do gerenciamento de tempo da Federação. Esta consiste na condição padrão para um Federado quando ingressa na Federação.

- Um Federado ao mesmo tempo regulador e restringido envolve-se totalmente no gerenciamento de tempo, sendo completamente sincronizado. Seu tempo lógico não pode avançar mais rápido do que o resto da Federação, nem a Federação pode avançar sem ele.

- Um Federado restringido mas não regulador de tempo permite o resto da Federação regular seu tempo lógico, mas ele não afeta os outros Federados. Corresponde uma escolha útil para Federados que exibem ou documentam historicamente os dados, ou outros Federados passivos.

- Um Federado regulador de tempo mas não restringido estabelece a velocidade do resto da Federação mas não é afetado por ela. Como exemplo, suponha uma Federação sincronizada de forma conservativa (todos os Federados são restringidos e reguladores), executando mais rápido do que um relógio normal, baseado em se-

gundos. Com o propósito de permitir a visualização da simulação em tempo real, pode-se adicionar um Federado que avance seu tempo lógico segundo uma taxa desejada relativa a esse relógio, retardando assim a execução da Federação.

Para se tornar regulador o Federado deve fornecer um valor, chamado *lookahead*, usado pela RTI para garantir que o Federado não possa enviar mensagens com estampas de tempo menores que o seu tempo lógico acréscido do *lookahead*. Conforme KUHL, WEATHERLY e DAHMANN (1999), isto se faz necessário para evitar a ocorrência de *deadlocks*. A RTI suporta valor nulo para *lookahead* em situações onde o Federado tem a capacidade de reverter para um estado anterior quando detectar um erro causado por uma entrega de mensagem fora de ordem.

A.1.6 Gerenciamento de Distribuição de Dados

Os serviços de gerenciamento de distribuição de dados *(Data Distribution Management (DDM))* podem ser usados para reduzir a transmissão e a recepção de dados irrelevantes. Durante operações de troca de informações, os Federados responsáveis pela atualização dos valores, denominados produtores, e os Federados inscritos nessas informações, denominados consumidores, podem usar os serviços deste grupo para habilitar na RTI a capacidade de reconhecer dados irrelevantes e prevenir as suas entregas aos consumidores (IEEE, 2000c).

As comunições relevantes envolvendo interações e atualizações de atributos de instância delimitam-se num espaço de dimensões definido pelo usuário. Os consumidores e produtores especificam limites superiores e inferiores para a porção relevante deste espaço, e a sobreposição do conjunto das regiões dos produtores e consumidores restringe a

comunicação relevante.

Apesar de opcionais, os serviços de DDM fornecem meios sofisticados para minimizar a quantia de dados transferidos, reduzindo assim a carga na rede. A Figura A.6 ilustra regiões de dados inscritos por dois consumidores, e uma região de dados de posse de um produtor. Quando o Federado produtor atualizar os seus dados, a reflexão dos novos valores ocorrerá apenas na área sobreposta entre a região do produtor e do consumidor.

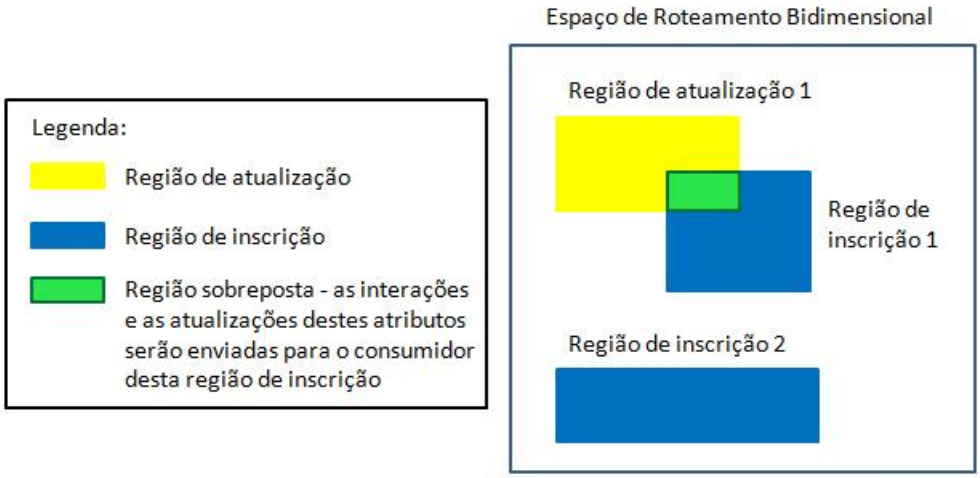

FIGURA A.6 – Exemplo de gerenciamento de distribuição de dados. (Adaptada de KOMPETENZZENTRUM HLA, 2003)

A.1.7 Serviços de Suporte

Este grupo disponibiliza aos Federados serviços adicionais para: conversão de identificadores para nomes, ou vice-versa; manipulação de regiões no DDM; inicialização e finalização da execução da RTI; dentre outros.

A.2 Gabarito de Modelo de Objetos

O padrão IEEE (2000c) define o *Object Model Template (OMT)*, correpondendo a um *framework* estrutural padronizado para especificar os modelos de objetos HLA. Este

gabarito consiste num componente essencial do padrão HLA, pois provê um mecanismo comum e padronizado para:

- Especificar a troca de dados e a coordenação geral entre os membros de uma Federação;
- Descrever a capacidade dos membros de uma Federação; e

Por ser uma estrutura padronizada para documentação os modelos de objetos HLA, este gabarito facilita o desenvolvimento e a aplicação de ferramentas utilitárias para a criação desses modelos.

Deve-se ressaltar, que tanto o *Federation Object Model - FOM* quanto o *Simulation Object Model - SOM* não correspondem completamente as definições comuns de modelos de objetos nas técnicas de Projeto e Análise Orientados a Objetos (*Object-Oriented Analysis and Design - OOAD*).

Na literatura de OOAD, descreve-se sistemas desenvolvidos usando objetos como abstrações, com o propósito do entendimento completo do sistema. Já nos modelos de objetos HLA, descreve-se o sistema de forma muito limitada, focando especificamente nos requisitos e nas capacidades de um Federado para trocar informações (IEEE, 2000c).

Um SOM tem a intenção de descrever a interface pública do Federado em termos de um conjunto identificado de classes de objetos e interações suportadas. Para uma descrição mais completa do funcionamento interno de um Federado pode-se usar outros recursos de documentação ao invés do SOM, por exemplo, um modelo OO tradicional. O SOM descreve somente quais informações um Federado compartilha na Federação. De forma semelhante, o FOM descreve a troca de informações que ocorrem durante a execução

da Federação, considerando todos os objetos e interações manipulados pelos Federados membros.

As diferenças de princípios e conceitos entre HLA e OOAD também aparecem no nível de objetos individuais. Na OOAD, define-se que os objetos encapsulam dados e operações, ou seja, atributos e métodos. Já no padrão HLA, define-se objetos apenas por características identificadas (atributos), sendo que nenhuma das operações, que afetam os valores dos atributos na OO, fazem parte do modelo de classes de objeto de um Federado.

No HLA, as classes de objetos formam uma hierárquia, onde herda-se atributos definidos em superclasses. Os atributos correspondem a propriedades abstratas das classes de objetos HLA, sendo referidos como atributos de classe. Usa-se o serviço apropriado na RTI para gerar uma nova instância de objeto, fornecendo sua classe de objeto como um gabarito. Cada atributo mantido por uma instância de objeto é denominado atributo de instância.

Na OO, os objetos interagem através de trocas de mensagem, quando um objeto invoca uma operação fornecida por outro objeto. Um objeto encapsula o seu estado localmente e associa a responsabilidade de atualização desse estado com operações implementadas no objeto numa linguagem de programação OO.

Já no padrão HLA, os objetos não interagem diretamente. OS Federados interagem, através dos serviços da RTI, para atualizar os valores de um ou mais atributos de instância ou enviar interações. Além disso, a responsabilidade para atualizar os atributos de uma instância pode ser distribuída entre diferentes Federados numa Federação.

De acordo com KUHL, WEATHERLY e DAHMANN (1999), os objetos HLA não possuem comportamentos associados a eles no FOM. A noção de “atributo de classe”

não corresponde à "variável de classe", como em algumas linguagens OO. No HLA, um atributo de classe considera a noção abstrata da classe e não uma instância dessa classe.

A estrutura para modelar a informação compartilhada, definida no OMT, corresponde a um conjunto de tabelas descrevendo classes com sub-classes e atributos/parâmetros descrevendo as respectivas classes e sub-classes. Assim, torna-se possível definir conceitos e também propriedades definindo os respectivos conceitos, numa estrutura de herança simples entre classes e sub-classes (TOLK, 2001).

Os modelos de objetos HLA não permitem definir relações como agregação, associação, herança múltipla, etc. Quando for necessário o compartilhamento de tais informações, será preciso a adição e a respectiva interpretação comum de dados adicionais ou de camadas de *software* nas interfaces dos Federados e da RTI (TOLK, 2001).

A.3 Regras

A especificação da arquitetura do padrão HLA estabelece um conjunto de regras HLA, descritas nas próximas Seções. Estas regras consistem em princípios e convenções considerados na criação de Federados e federações.

A.3.1 Regras de Federação

1. **Federações devem ter um FOM, documentado de acordo com o OMT** - todos os dados compartilhados devem ser declarados num FOM, que documenta o acordo realizado entre os Federados a respeito dos dados a serem trocados durante uma execução de Federação, usando serviços HLA, e também o conjunto mínimo de condições para troca de dados. Dessa forma, um FOM é um elemento essencial

definindo uma Federação. O HLA não define quais dados são incluídos num FOM, pois isto é responsabilidade do usuário e desenvolvedor da Federação. Os FOMs devem ser documentados no formato prescrito pelo OMT para possibilitar o seu reuso para os propósitos de novos usuários.

2. **Numa Federação, toda representação de instância de objeto associado à simulação deve estar nos Federados e não na RTI** - a manunteção dos valores dos atributos da instância de um objeto deve acontecer no Federado membro. Numa Federação, todos os atributos de instância associada a um Federado membro deve pertencer ao Federado e não a RTI. Porém, a RTI pode possuir alguns atributos de instância associada com o modelo de gerenciamento de objetos da Federação. A RTI pode usar dados sobre os atributos de instância e interações para suportar os serviços RTI (e.g., para o gerenciamento de declaração), mas esses dados são simplesmente usados pela RTI, não sendo alterados.

3. **Durante a execução de uma Federação, toda troca de dados entre Federados membros deve ocorrer via a RTI** - a especificação da interface HLA especifica um conjunto de serviços da RTI para suportar a troca coordenada de valores dos atributos de instância e interações conforme o FOM da Federação. A intercomunicação de dados entre Federados membros de uma Federação deve ser realizada através dos serviços da RTI. Com base no FOM, os Federados membros devem identificar a RTI quais informações eles irão prover e requerer. A RTI deverá então prover coordenação, sincronização, e troca de dados entre os Federados membros para permitir uma execução coerente da Federação. Os Federados membros têm por responsabilidade assegurar a produção e a utilização de dados no momento correto e de forma correta. A RTI, por sua vez, deve assegurar a entrega dos da-

dos aos Federados membros conforme seus requisitos declarados (nome dos dados, tipo de confiança no transporte, forma de ordenação dos evento, etc.) para prover uma visão comum de dados compartilhados durante uma determinada execução de Federação, como especificado no FOM.

4. **Durante uma execução de Federação, Federados membros devem interagir com a RTI de acordo com a especificação da interface** - o padrão HLA provê uma especificação para a interface padrão entre o Federado e a RTI. Os Federados membros devem usar essa interface padrão para acessar os serviços da RTI. Como a interface e a RTI serão usados por uma grande variedade de aplicações que requerem troca de dados de diversas características, a especificação não estabelece nada a respeito de dados específicos trocados através da interface.

5. **Durante uma execução de Federação, um atributo de instância pode pertencer a, no máximo, um Federado em qualquer dado momento** - o permite que diferentes Federados possuam diferentes atributos da mesma instância de objeto. Para assegurar a coerência de dados pela Federação, no máximo, um Federado membro pode possuir qualquer atributo de uma instância de objeto num determinado momento qualquer. Os Federados membros podem adquirir ou descartar a posse de atributos de uma instância, dinamicamente, durante a execução de uma Federação, permitindo que a propriedade de um atributo de instância seja transferida de um Federado membro para outro.

A.3.2 Regras de Federados

1. **Federados devem ter um SOM, documentado de acordo com o OMT** - o SOM deve incluir classes de objetos e de interação públicas do Federado numa Federação. O padrão HLA não especifica os dados incluídos no SOM, pois isso torna-se responsabilidade do desenvolvedor do Federado.

2. **Federados devem ser capazes de atualizar e/ou refletir qualquer atributos de instância e enviar e/ou receber interações, como especificado em seus SOMs** - o HLA permite Federados membros criarem representações internas dos objetos e interações disponíveis para o uso externo como parte da execução de uma Federação. Essa capacidade de interação externa deve ser documentada no SOM do Federado e deve incluir a obrigação de exportar valores atualizados de atributos de instância calculados internamente no Federado, e de receber interações representadas externamente (i.e., por outro Federado numa Federação).

3. **Federados devem ser capazes de transferir e/ou aceitar posse de atributos de instância dinamicamente durante uma execução de Federação, como especificado em seus SOMs** - o padrão HLA permite transferir a posse de atributos de instância de um objeto dinamicamente durante uma execução de Federação. Os atributos de instância que um Federado possui ou reflete, e cuja posse pode ser dinamicamente adquirida ou descartada durante a execução, devem ser documentados no SOM deste Federado.

4. **Federados devem ser capazes de variar as condições (e.g., limiares) sob as quais eles fornecem atualizações de atributos de instância, como especificado em seus SOMs** - o HLA permite aos Federados possuírem (i.e., proverem o

privilégio para produzir valores atualizados para) atributos de instâncias de objeto representados no Federado e então tornar esses valores disponíveis para outros Federados através da RTI. Diferentes federações podem especificar diferentes condições sob as quais atualiza-se atributos de instância (e.g., em alguma taxa especificada, ou quando a quantidade de mundanças no valor exceder um limiar específico, etc.). As condições aplicáveis à atualização de atributos específicos de instância por um Federado devem ser documentadas no SOM deste Federado.

5. **Federados devem ser capazes de gerenciar o tempo local de modo que permitam coordenar troca de dados com outros membros da Federação** - os projetistas de uma Federação identificarão suas abordagens de gerenciamento de tempo como parte do seu projeto de implementação. Os Federados membros devem aderir à abordagem de gerenciamento de tempo da Federação.

Anexo B - Glossário

A

Alcance - Medida da reta que liga a posição do lançador à posição do foguete em relação ao plano tangente à superfície da terra.

Apogeu - Altitude máxima alcançada por um veículo espacial durante uma operação de lançamento.

Arfagem - Movimento de um veículo espacial em torno do seu eixo transversal, relacionado ao efeito de subir e descer.

Arquitetura de Software - Estrutura global do *software* e os modos pelos quais essa estrutura fornece integridade conceitual para um sistema. É a estrutura hierárquica dos componentes de programa (módulos), o modo pelo qual esses componentes interagem e a estrutura dos dados que são usados pelos componentes (PRESSMAN, 2002).

Artefato - Termo geral para qualquer tipo de informação criada, produzida, alterada ou usada pelos responsáveis no desenvolvimento do sistema. Exemplos: os diagramas UML e seus textos associados, planos de interface, protótipos, códigos, planos de testes, destre outros (RESENDE, 2006).

Atitude - Corresponde às inclinações sofridas pelo foguete em função dos ângulos de elevação, azimute e rolamento (*theta, psi, phi*).

Azimute - Distância angular, medida sobre o horizonte, a partir de um ponto origem, geralmente o sul, no sentido dos ponteiros do relógio ou no sentido inverso, até o círculo vertical que passa por um dado astro.

B

***Benchmark* -** Avaliação de desempenho; ponto de referência para comparação.

C

Campanha de Lançamento - Caracteriza-se pelo conjunto de atividades planejadas para: o treinamento de meios operacionais, o lançamento de veículos espaciais e o rastreio de suas cargas úteis num centro de lançamento (CLA, 2007).

Classe Concreta - Classe que possui todos os seus métodos implementados e possibilita a criação de instâncias.

Classe Abstrata - Representa entidades e conceitos abstratos, não possuindo instâncias. Ela define um gabarito *(template)* para uma funcionalidade e fornece uma implementação incompleta (a parte genérica dessa funcionalidade), que é compartilhada por um grupo de classes derivadas (filhas). Cada uma das classes derivadas completa a funcionalidade da classe abstrata implementando o comportamento abstrato de modo específico.

Componente - Unidade funcional significativa de um sistema, podendo representar desde uma função de alto nível até uma tarefa de baixo nível. A arquitetura de um sistema

completo apresenta-se como uma organização hierárquica de componentes compostos ou atômicos. Além disso, define-se também componentes como unidades reutilizáveis e compartilhadas, podendo ser utilizadas em diversos projetos (RESENDE, 2006).

Coordenadas Polares Geocêntricas - Coordenadas esféricas (latitude, longitude e raio) de um ponto da esfera celeste, referidas ao centro da Terra.

Coordenadas Polares Geodésicas - Coordenadas esféricas (latitude, longitude e altitude acima do elipsóide de referência) de um ponto da esfera celeste, referidas ao centro da Terra.

D

Distância Radial - Medida da reta que liga a posição do lançador à posição do foguete.

E

Elevação - Altura do objeto (satélite, planeta, avião, foguete, etc.) acima da linha do horizonte medida em graus.

Elipsóide de Referência - Forma do planeta terra descrita por dois parâmetros geométricos, um semi-eixo maior do elipsóide (raio) e um semi-eixo menor (usado para definir o achatamento).

Equinócio Vernal - Ponto onde o Sol cruza o equador celeste.

F

Fases de Vôo - Etapas pré-definidas a serem cumpridas em ordem cronológica, sendo associadas a ocorrência de eventos na simulação como: liberação de estágios, apogeu, impacto, dentre outros. A divisão da simulação em fases possilita modelar descontinuidades de parâmetros de vôo como variações abruptas de massa causadas pelo fim da queima e liberação de estágios.

Federação - Sistema de simulação criado a partir da união de simulações. Numa Federação, qualquer número de sistemas de simulação (Federado), geralmente distribuído fisicamente, pode ser reunido num ambiente de simulação unificado para endereçar as necessidades de novas aplicações.

Federado - Cada simulação combinada para formar uma Federação. Os Federados podem consistir de diversos processos, executando em diversos computadores. Eles reunem-se numa Federação e são vistos como componentes na arquitetura do padrão HLA.

Ferramenta de Engenharia de *Software* - Usadas para fornecer apoio automatizado ou semi-automatizados para o processo e para os métodos. Quando ferramentas são integradas, de modo que a informação criada por uma ferramenta pode ser usada por outra, estabelece-se um sistema de apoio ao desenvolvimento de *software*, chamado Engenharia de *Software* Ajudada por Computador (*Computer-Aided Sofware Engineering - CASE*) (PRESSMAN, 2002).

Fluxo de Controle - Seqüência de execução das instruções de um programa, envolvendo repetição de instruções, desvios causados por instruções condicionais, etc.

Cada execução de um programa pode resultar numa seqüência diferente de execução das instruções, resultando em fluxo de controle e de dados diferentes.

Fluxo de Dados - Fluxo de informações durante a execução de um programa, envolvendo arquivos, variáveis, vetores e outras estruturas de dados.

Fluxo de Execução - Execução de uma seqüência de instruções em linguagem de máquina, normalmente gerada pela compilação de um programa escrito numa linguagem qualquer, também denominada "tarefa", possuindo um fluxo de controle e um fluxo de dados.

Foguete de Sondagem - Veículo espacial utilizado para missões sub-orbitais de exploração do espaço capazes de lançar cargas úteis compostas por um ou mais experimentos científicos e tecnológicos (AEB, 2007).

***Framework* -** Esqueleto de solução para uma classe de problemas similares (BUDD, 2002). Um conjunto de classes que incorporam um projeto abstrato para soluções a uma família de problemas relacionados (JOHNSON; FOOTE, 1988).

G

Gabarito (*Template*) - Artefato com formatação padrão e orientações quanto a seu preenchimento (RESENDE, 2006).

Guinada - Movimento de um veículo espacial em torno de um eixo contido num plano vertical, e normal ao eixo longitudinal do veículo, relacionado ao efeito de mover para esquerda ou direita.

H

***High Level Archicteture:* -** *Framework* para o desenvolvimento de sistemas de simulação num alto nível de abstração, através da especificação de interfaces e regras de projeto (KUHL; WEATHERLY; DAHMANN, 1999), definido no *IEEE STD 1516 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA).*

***Hot Spot* -** Partes da estrutura de classes do *framework* a serem mantidas flexíveis, para possibilitar sua adaptação a diferentes aplicações do domínio (SILVA; PRICE, 1997).

I

Interface - Classe sem atributos e com todos os métodos abstratos, i.e., não implementados. Define o comportamento das classes que derivam-se dela.

M

Mancha Solar - Região onde ocorre uma redução de temperatura e pressão das massas gasosas no Sol, estando intimamente relacionadas ao seu campo magnético.

Método - Palavra de origem greaga *méthodos* que significa "caminho para chegar a um fim", "modo ordenado de proceder" ou "conjunto de procedimentos técnicos e científicos" (RESENDE, 2006).

Metodologia - Composição de 2 palavras de origem grega *méthodos* + *lógos*, representando método + tratado ou ciência, significando Ciência dos Métodos ou Tratado dos Métodos, cujo objetivo consiste em estudar os métodos (RESENDE, 2006).

Modelo - Representação simplificada e abstrata de algo que se pretende conceber (RESENDE, 2006).

O

Objeto - A Programação Orientada a Objetos (POO) define um objeto como uma instância de uma classe, que encapsula dados e operações, ou seja, atributos e métodos. Já no padrão HLA, define-se um objeto apenas por atributos. Nenhuma operação, que afeta os valores dos atributos na OO, faz parte do modelo de classes de objeto do padrão HLA.

P

Ponto de Impacto - Coordenadas do local onde um foguete se chocou com a superfície terrestre.

Processo de *Software* - Conjunto de atividades e resultados associados que produzem um produto de *software* (RESENDE, 2006).

Programa - Código-fonte. Conjunto de uma ou mais seqüências de instruções em linguagem de computador, escritas para resolver um problema específico.

R

Requisitos - Propriedade geral, funcionalidade ou restrição do sistema de *software* (RESENDE, 2006).

Requisitos Funcionais - Funcionalidades que um sistema deve ter para satisfazer às necessidades do usuário.

Requisitos Não-Funcionais - Propriedade de qualidade ou restrição que o *software* precisa satisfazer.

***Runtime Infrastruture* -** *Software* de suporte responsável pela execução simultânea da Federação, permitindo que os Federados executem juntos para alcançar os objetivos da Federação.

Rolamento - Movimento de rotação de um foguete em torno do seu eixo longitudinal.

S

Simulação - Imitação da operação de um sistema ou processo do mundo real em relação ao tempo, envolvendo a criação de uma história artificial do sistema e a sua observação para extrair inferências relacionadas a características operacionais representadas. (BANKS, 1998).

Simulação Computacional - Um analista constrói um modelo de um sistema de interesse, escreve os programas que o contêm e usa um computador para imitar o seu comportamento quando sujeito a uma variedade de políticas de operações, escolhendo a mais desejada (PIDD, 1997).

Sistema de Controle - Sistema responsável por forçar o foguete a seguir a sua trajetória nominal, ou uma aproximação dessa trajetória, que garanta o cumprimento de sua missão (IAE, 1993).

Sistema de Referência - Sistema de Coordenadas num espaço tridimensional.

***Software* -** Pressman (2002) afirma que *software* são: instruções (programas de computadores) que quando executadas fornecem a função e o desempenho desejados, estruturas de dados que permitem aos programas manipular adequadamente a informação e documentos que descrevem a operação e o uso dos programas. O *software*

é um produto de Engenharia *Software*, que deve ser realizada adotando-se processos, padrões, métodos, técnicas e ferramentas definidos para estes tipos de sistemas.

***Stand-alone* -** Este termo representa: um computador independente que não está conectado a uma rede; um dispositivo que não requer outros dispositivos para funcionar; ou ainda um programa que é executado sem depender de um sistema operacional ou outro *software* de suporte.

T

Técnica - Forma de execução das atividades preconizadas nos métodos (RESENDE, 2006).

Tecnologia de *Software* - Aplicação da ciência. Conjunto de métodos, técnicas, ferramentas, processos e procedimentos para o desenvolvimento de *software*.

Trajetória Nominal - Previsão do caminho que um foguete percorre no espaço aéreo num período de tempo (LOUIS, 2006).

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO N°	4. N° DE PÁGINAS
TM	17 de dezembro de 2007	CTA/ITA-IEC/TM-018/2007	220

5. TÍTULO E SUBTÍTULO:

Fênix - Um *Framework* para Simulação de Trajetórias de Foguetes de Sondagem.

6. AUTOR(ES):

Juliano de Almeida Monte-Mor

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica. Divisão de Ciência da Computação – ITA/IEC

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Simulação Computacional; Trajetória; *High Level Architecture - HLA*; Veículos Espaciais de Sondagem

9.PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Simulação computadorizada; Trajetórias de foguetes; Arquitetura de software; Linguagens de alto nível; Foguetes-sonda; Engenharia de software; Engenharia aeroespacial

10. APRESENTAÇÃO: **X Nacional** **Internacional**

ITA, São José dos Campos, 2007, 220 páginas

11. RESUMO:

Cada vez mais os métodos, técnicas e ferramentas de Engenharia de *Software* têm sido aplicados para criação, manutenção e adaptação de sistemas de *software* aeroespaciais. A técnica de orientação a objetos tem provido o reuso de componentes de *software*, reduzindo o esforço para desenvolver aplicações similares num domínio específico. Este trabalho de pesquisa propõe um *Framework* para a Simulação de Trajetórias de Foguetes de Sondagem. Este *Framework* define um modelo abstrato para a construção de simuladores de trajetórias. Um Estudo de Caso e um Protótipo de Simulador foram desenvolvidos para avaliar a reusabilidade do *Framework* proposto. O protótipo provê cálculos de movimento de um foguete com um, três e seis graus de liberdade. Neste trabalho, também é avaliado o esforço necessário de incorporação da arquitetura de simulação distribuída definida no padrão *IEEE STD 1516 - High Level Architecture - HLA*. Uma análise dos principais resultados desta pesquisa propicia uma verificação do *Framework* proposto, quanto a reutilização significativa de código para a construção de modelos de simuladores de trajetórias. Esta análise também propicia uma avaliação do padrão HLA para a criação de simulações distribuídas, visando uma melhor padronização, interação e reusabilidade dos sistemas.

12. GRAU DE SIGILO:

(X) OSTENSIVO **( ) RESERVADO** **( ) CONFIDENCIAL** **( ) SECRETO**

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo