sid.inpe.br/mtc-m19/2012/03.01.14.50-TDI

UMA ARQUITETURA DE SOFTWARE EMBARCADO DO SEGMENTO ESPACIAL PARA HABILITAR A OPERAÇÃO DE MISSÕES BASEADA EM OBJETIVOS

Fabrício de Novaes Kucinskis

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 28 de junho de 2012.

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

INPE
São José dos Campos
2012

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

CONSELHO DE EDITORAÇÃO E PRESERVAÇÃO DA PRODUÇÃO INTELECTUAL DO INPE (RE/DIR-204):

Presidente:

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Membros:

Dr. Antonio Fernando Bertachini de Almeida Prado - Coordenação Engenharia e Tecnologia Espacial (ETE)
Drª Inez Staciarini Batista - Coordenação Ciências Espaciais e Atmosféricas (CEA)
Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)
Dr. Germano de Souza Kienbaum - Centro de Tecnologias Especiais (CTE)
Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos (CPT)
Drª Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação
Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

BIBLIOTECA DIGITAL:

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)
Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

EDITORAÇÃO ELETRÔNICA:

Ivone Martins - Serviço de Informação e Documentação (SID)

sid.inpe.br/mtc-m19/2012/03.01.14.50-TDI

UMA ARQUITETURA DE SOFTWARE EMBARCADO DO SEGMENTO ESPACIAL PARA HABILITAR A OPERAÇÃO DE MISSÕES BASEADA EM OBJETIVOS

Fabrício de Novaes Kucinskis

Tese de Doutorado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais, orientada pelo Dr. Maurício Gonçalves Vieira Ferreira, aprovada em 28 de junho de 2012.

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

INPE
São José dos Campos
2012

Dados Internacionais de Catalogação na Publicação (CIP)

Kucinskis, Fabrício de Novaes.
K952u Uma arquitetura de software embarcado do segmento espacial para habilitar a operação de missões baseada em objetivos / Fabrício de Novaes Kucinskis. – São José dos Campos : INPE, 2012.
xxvi + 166 p. ; (sid.inpe.br/mtc-m19/2012/03.01.14.50-TDI)

Tese (Doutorado em Engenharia e Tecnologia Espaciais) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2012.
Orientador : Dr. Maurício Gonçalves Vieira Ferreira.

1. operação de missões baseada em objetivos. 2. planejamento embarcado. 3. autonomia de satélites. 4. problemas de satisfação de restrições. 5. inteligência computacional. I.Título.

CDU 629.7:004.832.25

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

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

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

Engenharia e Tecnologia Espaciais/Gerenciamento de Sistemas Espaciais

Dra. Ana Maria Ambrosio

Presidente / INPE / São José dos Campos - SP

Dr. Mauricio Gonçalves Vieira Ferreira

Orientador(a) / INPE / SJCampos - SP

Dr. Nilson Sant'Anna

Membro da Banca / INPE / SJCampos - SP

Dr. Lamartine Nogueira Frutuoso Guimarães

Membro da Banca / IEAv/DCTA / SJCampos - SP

Dr. Reinaldo Gen Ichiro Arakaki

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

Dra. Adriana C. Ferrari Santos

Convidado(a) / Vens/IBTA / São José dos Campos - SP

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): ***Fabrício de Novaes Kucinskis***

São José dos Campos, 28 de Junho de 2012

À Sofia. Agora o papai pode brincar.

AGRADECIMENTOS

Há muitas pessoas a quem agradecer, mas devo prestar os primeiros e maiores agradecimentos à minha esposa Olga. Por seu apoio, sua confiança em mim, sua paciência comigo. Por acreditar que seria possível cumprir este doutorado em meio a fraldas, ao trabalho, meses e mais meses em missão no exterior, até vulcões encaramos! Não sei se é usual registrar isso numa Tese, mas Nega, te amo. Demais.

Agradeço ao meu amigo e orientador Maurício, que me apresentou ao INPE e vem me acompanhando desde a graduação, sempre confiando em meus esforços e contribuindo muito para o amadurecimento do trabalho que se apresenta aqui.

Ao Fábio Armelin, o revisor mais criterioso do lado de cá do Equador. Ao Dr. Mário Selingardi, chefe da Divisão de Eletrônica Aeroespacial do INPE, por me permitir a inscrição nesta pós-graduação, acreditando que ainda assim minhas obrigações para com os programas do Instituto seriam cumpridas.

Aos companheiros do Grupo de Supervisão de Bordo pelo constante apoio e pela troca produtiva de ideias.

A todos, muitíssimo obrigado. Espero poder retribuir à altura.

RESUMO

A presente Tese descreve uma arquitetura de *software* embarcado no segmento espacial para habilitar a operação de missões baseada em objetivos, bem como a operação autônoma. Nesse novo paradigma, objetivos são enviados ao satélite, ao invés de sequências de comandos, e cabe a ele determinar a melhor maneira de cumpri-los. Isso implica na transferência de parte do processo decisório do segmento solo para o espacial. A transferência pode ser feita pela realização a bordo de inferência dos estados futuros do satélite, baseada em modelos, e de planejamento embarcado. A Tese apresenta os conceitos da operação de missões e traz uma revisão sobre o estado da arte em *software* de bordo para a operação baseada em objetivos. Os componentes da arquitetura proposta foram implementados e submetidos a estudos de caso de missões do INPE. A proposta difere de outras por ser aplicável a diferentes missões, adequada para execução a bordo de satélites, integrada ao *software* de voo, e por unificar atribuições de sistemas planejadores e escalonadores. O *software* foi executado em um computador de bordo, e dados de desempenho demonstram que ele pode ser considerado para um experimento de validação tecnológica em uma missão futura do Instituto.

A SOFTWARE ARCHITECTURE ON-BOARD THE SPACE SEGMENT TO ENABLE THE GOAL-BASED MISSIONS OPERATION

ABSTRACT

This Thesis describes an on-board software architecture for the space segment which aims to enable the goal-based missions operation, as well as the autonomous operation. In this new paradigm, goals are sent to the satellite, instead of sequences of commands, and it is responsible to determine the best way to achieve them. This implies the transfer of part of the decisional process from the ground to the space segment. The transfer can be performed by model-based future satellite states inference and on-board planning. The Thesis presents the mission operation concepts and brings a review of the state of the art on on-board software for goal-based operations. The components of the proposed architecture were implemented and submitted to case studies from INPE's missions. The proposal differs from others by being applicable to different missions, adequate for execution on-board satellites, integrated to the flight software, and by unifying attributions from planning and scheduling systems. The software was executed in an on-board computer, and the performance data shows that it can be considered for a technological validation experiment in a future mission of the Institute.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

3CS	*Three Corner Sat*
ACM	*Association for Computing Machinery*
AEGIS	*Autonomous Exploration for Gathering Increased Science*
AGATA	*Autonomy Generic Architecture: Tests and Applications*
AIAA	*American Institute of Aeronautics and Astronautics*
AOC	*Attitude and Orbit Control*
ARC	*Ames Research Center*
ASE	*Autonomous Sciencecraft Experiment*
ASPEN	*Automated Scheduling and Planning Environment*
AUV	*Autonomous Underwater Vehicle*
AWFI	*Advanced Wide-Field Imager*
CASPER	*Continuous Activity Scheduling, Planning, Execution and Replanning*
CBERS	*China-Brazil Earth Resources Satellite*
CCD	*Charged-Coupled Device*
CLARAty	*Coupled Layer Architecture for Robotic Autonomy*
CLEaR	*Closed Loop and Recovery*
CNES	*Centre National d'Études Spatiales*
COMAV	Computador Avançado
CPU	*Central Processing Unit*
CSP	*Constraint Satisfaction Problem*
DCTA	Departamento de Ciência e Tecnologia Aeroespacial
DDR	*Digital Data Recorder*
DEA	Divisão de Eletrônica Aeroespacial
DS-1	*Deep Space One*
DSS	*Digital Signal Switch*
DT	*Data Transmitter*

ECSS	*European Committee for Space Standardization*
EO-1	*Earth Observing One*
EQUARS	*Equatorial Atmosphere Research Satellite*
ESA	*European Space Agency*
EUROPA	*Extendable Uniform Remote Operations Planning Architecture*
GCC	*GNU Cross-Compiler*
GOAC	*Goal-Oriented Autonomous Controller*
GOESA	*Goal-based Enabling Software Architecture*
GPM-Br	*Global Precipitation Measurement Brazil*
GSFC	*Goddard Space Flight Center*
GTEO	*Global Terrestrial Ecosystem Observatory*
HSTS	*Heuristic Scheduling Testbed System*
HTN	*Hierarchical Task Network*
IA	Inteligência Artificial
ID	Identificador
IDE	*Integrated Development Environment*
IDEA	*Intelligent Distributed Execution Architecture*
IEAv	Instituto de Estudos Avançados
INPE	Instituto Nacional de Pesquisas Espaciais
ISIS	*Internal State Inference Service*
ISISml	*ISIS modeling language*
ISTC	*Instituto di Scienze e Tecnologie della Cognizione*
JPL	*Jet Propulsion Laboratory*
LAAS	*Laboratoire d'Analyse et d'Architecture des Systèmes*
MDS	*Mission Data System*
MER	*Mars Exploration Rovers*
MIPS	*Millions of Instructions Per Second*
MIRAX	Monitor e Imageador de Raios X
MMOPS	*Mars Mission On-board Planner and Scheduler*

MSL	*Mars Science Laboratory*
NASA	*National Aeronautics and Space Administration*
NewMAAP	*New Millennium Autonomy Architecture Prototype*
OASIS	*Onboard Autonomous Science Investigation System*
OBDH	*On-Board Data Handling*
OBT	*On-Board Time*
ONERA	*Office National d'Etudes et de Recherches Aerospatiales*
OPS-USERS	*Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance*
PC	*Personal Computer*
PDDL	*Planning Domain Description Language*
PITER	Processamento de Imagens em Tempo Real
PLEXIL	*Plan Execution Interchange Language*
PMM	Plataforma Multimissão
PSL	*Plan Service Layer*
PSS	*Power Supply Subsystem*
PUS	*Packet Utilization Standard*
RAM	*Random Access Memory*
RASSO	*Resources Allocation Service for Scientific Opportunities*
RAX	*Remote Agent Experiment*
RCP	*Rich Client Platform*
RIACS	*Research Institute for Advanced Computer Science*
RTEMS	*Real-Time Executive for Multiprocessor Systems*
SADA	*Solar Array Drive Assembly*
SCL	*Spacecraft Command Language*
SIS	*SPARC Instruction Simulator*
SpaceOps	*International Conference on Space Operations*
SPARC	*Scalable Processor Architecture*
STN	*Simple Temporal Network*

SUBORD	Grupo de Supervisão de Bordo
TDL	*Task Description Language*
T-REX	*Teleo-Reactive Executive*
TVCR	*Timeline Validation, Control and Repair*
UAV	*Unmanned Aerial Vehicle*
VANT	Veículo Aéreo Não Tripulado
VML	*Virtual Machine Language*
VSNT	Veículo Submarino Não Tripulado
XML	*eXtensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Em uma missão espacial, a responsabilidade sobre a realização das tarefas operacionais é dividida entre o segmento solo e o segmento espacial. As primeiras missões alocavam para solo quase todas as tarefas. Em tais missões, o segmento espacial era um mero executor dos comandos enviados de solo.

A adição de computadores ao segmento espacial permitiu que os operadores em solo passassem a delegar tarefas para execução a bordo. A partir de então, o contínuo aumento da capacidade computacional embarcada, a experiência obtida em missões anteriores, assim como as próprias necessidades das novas missões, vêm fazendo com que mais e mais tarefas sejam alocadas ao segmento espacial.

Procedimentos de rotina como o monitoramento de parâmetros de bordo para a geração de alertas, o gerenciamento do envio de dados para solo e a sincronização de relógios são hoje executados pelo segmento espacial de forma autônoma. Esse tipo de autonomia tem como principais objetivos otimizar o uso do *link* de comunicação de serviço e reduzir o esforço, e portanto o custo, da operação.

Já as tarefas que demandam algum tipo de planejamento, como a operação das cargas úteis, continuam sendo de inteira responsabilidade da equipe de operações em solo. O planejamento envolve a análise de uma série de fatores e um processo de raciocínio sobre quais ações tomar, e quando, para se atingir um determinado objetivo da missão. Em uma missão de sensoriamento remoto, por exemplo, são levadas em conta a posição do satélite em órbita, a quantidade de memória de armazenamento disponível, e a previsão de passagens sobre a estação terrena, entre outras variáveis, para que se determine quando as câmeras devem ser ligadas e desligadas.

Transferir também o processo de planejamento para o segmento espacial, ou ao menos parte dele, é algo que vem sendo buscado por agências espaciais há pouco mais

de uma década. Isso tem o potencial de habilitar novas capacidades em missões futuras e aproveitar melhor os recursos disponíveis em missões existentes.

Nesse caso, o segmento espacial deixa de receber sequências de comandos previamente definidas em solo, para receber objetivos a serem cumpridos. Cabe ao *software* de voo, dotado de conhecimento sobre o domínio no qual se encontra e capacidade de raciocínio sobre tal domínio, realizar o planejamento e determinar, em bordo, quais sequências de comandos levam o segmento espacial a cumprir os objetivos que lhe foram passados. A essa forma de operação de missões espaciais dá-se o nome de 'operação baseada em objetivos'.

Embora as sequências de comandos tenham se mostrado suficientes até os dias de hoje, as missões previstas para os próximos anos apresentam um crescente nível de complexidade. Operar uma missão no nível de objetivos pode levar à redução da complexidade, esforço e custo de operação.

Essa forma de operação também pode levar a um aumento substancial na autonomia, se o segmento espacial for capaz de definir por si só, em determinadas situações, seus próprios objetivos. Essa nova característica transforma a operação baseada em objetivos em uma operação autônoma do segmento espacial.

Um satélite com tal característica estaria livre das restrições temporais impostas pela comunicação com o segmento solo para a tomada de decisões, podendo responder em tempo real a situações não previstas ou transitórias.

Satélites de monitoramento ambiental, ao detectar queimadas, enchentes ou a formação de tempestades, reprogramariam suas atividades para obter, já na próxima órbita, mais imagens do evento observado. Satélites científicos teriam a capacidade de reagir a fenômenos físicos de curta duração, aproveitando oportunidades científicas que hoje ficam sem resposta.

Além disso, satélites já em órbita, potencialmente aqueles que já cumpriram a vida útil

prevista, poderiam fazer uso de planejamento embarcado para aproveitar melhor seus recursos disponíveis.

Possuindo acesso completo e em tempo real aos dados do satélite – algo que um planejador em solo não pode ter –, o planejador embarcado poderia reduzir as margens de segurança usualmente adotadas no consumo de recursos (memória, energia e combustível para propulsores, por exemplo), ajustando os planos ou adaptando-os a modos de operação degradados, na medida em que as tarefas forem executadas.

Projetos vêm sendo desenvolvidos nos últimos anos pela *National Aeronautics and Space Administration* (NASA), pela *European Space Agency* (ESA) e por instituições de pesquisa francesas vinculadas ao *Centre National d'Études Spatiales* (CNES), adotando técnicas de Inteligência Computacional de forma a possibilitar a operação baseada em objetivos em missões futuras. Alguns desses projetos geraram sistemas ou protótipos de sistemas com planejamento embarcado.

Dos sistemas desenvolvidos, os poucos que realmente executaram em *hardware* de voo foram criados para uma missão específica, ou profundamente adaptados a ela. Outros que se propuseram a ser de uso geral, aplicáveis a diferentes missões, foram executados em plataformas do tipo PC, com poder de processamento ordens de grandeza superior àquele existente a bordo de veículos espaciais.

Para que a operação baseada em objetivos seja viável, é preciso desenvolver sistemas de planejamento aptos à execução em *hardware* de voo, com capacidade computacional bastante limitada, e que sejam aplicáveis a diferentes missões sem grande necessidade de adaptações – assim como são hoje os sistemas executores de sequências de comandos.

Entretanto, não há registro de um sistema de planejamento desenvolvido de forma a ser aplicável a diferentes missões sem a necessidade de adaptações importantes, e

ainda assim capaz de executar em *hardware* de voo.

Além disso, um problema presente em quase todos os planejadores voltados ao segmento espacial provém da existência de mais de uma descrição do conhecimento sobre o domínio, na forma de modelos embarcados. Há modelos distintos para planejamento, escalonamento, execução de planos e outros.

A presença de dois ou mais modelos que descrevam aspectos diferentes, mas inter-relacionados, de um mesmo domínio, abre a possibilidade para a ocorrência de falhas geradas por diferenças sutis entre os comportamentos previstos por cada modelo. Manter a sincronização entre os modelos é uma tarefa difícil e propensa a erros.

O principal motivo para a existência de mais de um modelo embarcado é a separação que os sistemas existentes fazem entre o planejamento, que é a determinação da sequência das atividades a realizar para atingir objetivos, e o escalonamento, que trata de posicionar temporalmente as atividades e alocar recursos a elas.

Nesse caso, o planejador e o escalonador operam de forma sequencial: o planejador determina um conjunto completo de atividades para atingir os objetivos, e então as encaminha ao escalonador. Cabe ao escalonador posicionar temporalmente as atividades e alocar os recursos necessários à sua execução. A Figura 1.1 ilustra essa forma de planejamento.

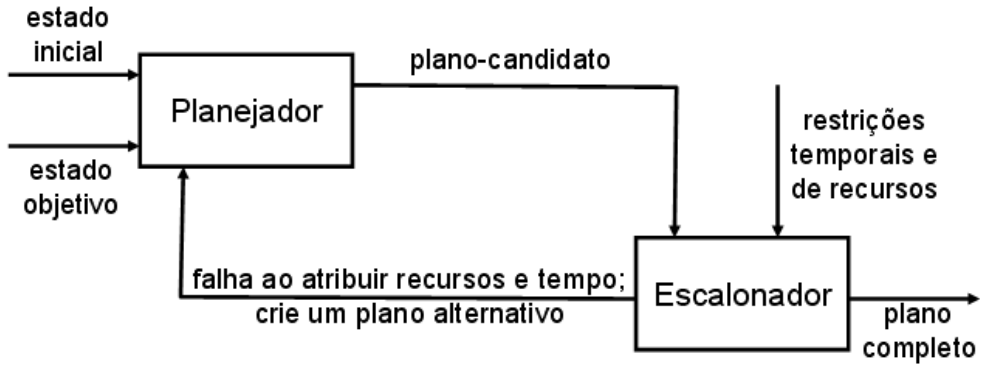


Figura 1.1 - Planejamento e escalonamento em cascata

Fonte: Kucinskis (2007, p. 59).

Essa separação também tem impactos no desempenho do sistema. Nem todos os planos são escalonáveis, e caso o escalonador não consiga posicionar as atividades no tempo ou alocar recursos a elas, o planejador é novamente chamado para gerar um plano alternativo. O processo se repete até que o escalonador consiga alocar tempo e recursos a todas as atividades, quando o plano finalmente pode ser encaminhado para execução.

Tal abordagem foi herdada de sistemas planejadores de missões espaciais utilizados nos centros de controle de operações em solo, concebidos para execução em computadores muito diferentes dos disponíveis para uso espacial.

Uma eventual unificação entre o planejador e o escalonador permitiria a existência de um modelo único, além de ter o potencial de melhorar o desempenho do sistema e de simplificar a arquitetura de *software* necessária à sua implementação. Isso impõe, entretanto, o estabelecimento de uma forma de descrever o conhecimento sobre o domínio diferente da que vem sendo usada nos planejadores de uso espacial.

A operação de missões baseada em objetivos e a operação autônoma do segmento espacial constituem uma evolução do paradigma atual de operação por sequências de comandos e, como tal, trazem uma série de novos problemas a serem tratados, sendo os listados aqui apenas uma parte deles.

Desenvolver a tecnologia necessária para isso, entretanto, tem o potencial de levar a operação de missões espaciais a um novo patamar, no qual o esforço e custos sejam reduzidos, ao mesmo tempo em que o retorno da missão cresça substancialmente.

1.1. Motivação do Trabalho

Um primeiro estudo sobre planejamento embarcado em satélites do Instituto Nacional de Pesquisas Espaciais (INPE) foi realizado por Kucinskis (2007). O estudo trouxe ainda uma extensa pesquisa bibliográfica sobre a adoção de técnicas de planejamento automatizado na área espacial, tanto em solo como em bordo.

A partir dessa pesquisa foram identificadas tecnologias adequadas para execução no segmento espacial. Isso resultou na implementação de um protótipo de replanejador embarcado, que tinha por objetivo modificar o plano original de operações, enviado de solo, para gerar respostas a fenômenos físicos de curta duração em um satélite científico. O protótipo foi executado em *hardware* de bordo de satélites, demonstrando a viabilidade da solução adotada.

O trabalho encerra sugerindo outras aplicações para a tecnologia demonstrada a futuras missões do INPE. Isso motivou o início de uma abordagem mais ampla sobre o tema de operações baseadas em objetivos e autonomia do segmento espacial, com a identificação de possíveis benefícios ao Instituto, bem como o desenvolvimento de melhorias diversas sobre o trabalho de 2007, dessa vez visando não apenas uma aplicação específica (a resposta a fenômenos de curta duração), mas uma gama delas.

Prover ao Instituto uma arquitetura de *software* capaz de gerenciar, a bordo do segmento espacial, conhecimento sobre o domínio em que se encontra, e permitir que aplicações embarcadas raciocinem sobre tal conhecimento, é um grande passo na busca por mais autonomia nas missões brasileiras. Este trabalho busca fazer isso de forma padronizada, capaz de atender a várias missões, e integrada ao *software* de voo atual.

1.2. Objetivos do Trabalho

Este trabalho tem como objetivo principal propiciar a habilitação das operações baseadas em objetivos e o consequente aumento da autonomia do segmento espacial nas missões do INPE, através do desenvolvimento de tecnologia de planejamento embarcado e de uma arquitetura de *software* que o suporte.

A arquitetura proposta visa as seguintes características, não encontradas em conjunto em sistemas similares:

- Unificar planejamento e escalonamento, permitindo a existência de um único modelo de domínio embarcado;
- Ser de uso geral, aplicável a diferentes missões;
- Ser executável em *hardware* de voo, e integrada ao *software* de voo.

Foi criado um sistema de *software* capaz de gerenciar o conhecimento sobre o domínio e permitir que aplicações de bordo usem tal conhecimento para, dado um conjunto de objetivos, realizar de forma autônoma o planejamento de operações.

O protótipo de replanejador embarcado desenvolvido por Kucinskis (2007) foi usado como ponto de partida. A pesquisa então realizada foi expandida e aprofundada. Enquanto tal protótipo visava uma aplicação específica, o presente trabalho é aplicável a diferentes tipos de missões e objetivos.

Pretende-se que o *software* desenvolvido como parte desta Tese seja considerado futuramente para a realização de um experimento de validação tecnológica em órbita. Portanto, todas as etapas da pesquisa e da implementação foram permeadas por considerações especiais com respeito à integração com o *software* de voo, padronização, e com as práticas da engenharia de satélites do INPE.

1.3. Metodologia de Trabalho e Organização desta Tese

Este trabalho foi iniciado com o estudo do paradigma da operação de missões espaciais baseada em objetivos, uma comparação com o paradigma vigente, e seu relacionamento com a tecnologia de planejamento autônomo, o que se encontra descrito no Capítulo 2 da presente Tese. O Capítulo apresenta ainda considerações sobre o risco e benefícios do aumento da autonomia em missões espaciais.

Foi então realizada uma ampla pesquisa sobre projetos de *software* embarcado voltados à operação de missões espaciais baseada em objetivos. Tal pesquisa, sumarizada no Capítulo 3, aprofunda o estudo inicial realizado por Kucinskis (2007),

mostrando não apenas os sistemas desenvolvidos, mas a evolução dos trabalhos realizados pelos grupos de pesquisa que os criaram. É ilustrada a evolução e o estágio atual de cada grupo, contextualizando o presente trabalho com relação aos demais.

Os estudos realizados foram utilizados na concepção e projeto de uma arquitetura de *software* embarcado no segmento espacial para habilitar a operação de missões baseada em objetivos. Tal arquitetura, apresentada no Capítulo 4, separa o gerenciamento do conhecimento do processo de raciocínio realizado sobre ele, o que permite aplicá-la a diferentes missões.

O Capítulo 5 descreve a etapa seguinte, que consistiu do estudo das principais formas de representação e manipulação de objetivos a bordo, e da definição dos conceitos básicos de representação do conhecimento utilizados neste trabalho. São então descritas as técnicas de modelagem adotadas para permitir a unificação do planejamento e escalonamento, com exemplos da linguagem criada para isso.

A implementação dos componentes da arquitetura é descrita nos Capítulos 6 e 7, e os estudos de caso aplicados a ela, no Capítulo 8. Dados de desempenho e considerações sobre as decisões autônomas tomadas em função dos objetivos recebidos também são apresentados.

As conclusões deste trabalho, suas contribuições e os trabalhos futuros vislumbrados são descritos no Capítulo 9.

2. A OPERAÇÃO DE MISSÕES ESPACIAIS BASEADA EM OBJETIVOS

Desde que computadores de bordo passaram a ser adotados em missões, a operação do segmento espacial baseia-se no envio de sequências de comandos previamente definidas em solo. Esse paradigma tem se mostrado suficiente para satélites cuja operação das cargas úteis é relativamente simples, como são as missões atuais do INPE.

Entretanto, missões já previstas e em diferentes fases de desenvolvimento pelo Instituto apresentam um crescente nível de complexidade. Entre os requisitos para tais missões encontram-se: (i) a detecção e resposta autônoma a eventos em órbita (Monitor e Imageador de Raios X, MIRAX), (ii) o gerenciamento simultâneo de conjuntos de experimentos científicos com fins diversos (*Equatorial Atmosphere Research Satellite*, EQUARS), (iii) câmeras multiespectrais (*Global Terrestrial Ecosystem Observatory*, GTEO e futuros *China-Brazil Earth Resources Satellite*, CBERS) e outros.

Quanto maior a complexidade de uma missão, maiores as limitações apresentadas pelas sequências de comandos. Nesse cenário, a operação baseada em objetivos apresenta-se como uma evolução do paradigma vigente, com potencial para suprir as demandas operacionais de futuras missões. Mais do que isso, o conceito de operações baseadas em objetivos abre caminho para operações total ou parcialmente autônomas do segmento espacial.

Este Capítulo apresenta a evolução do paradigma de operação do segmento espacial.

2.1. O Paradigma de Operação Baseada em Sequências de Comandos

O paradigma atual de operação do segmento espacial baseia-se na geração de sequências de comandos fixas e lineares, previamente definidas pelos operadores da missão em solo. Alguns desses comandos deverão ser executados pelo veículo espacial

de forma imediata assim que recebidos, enquanto outros serão agendados para execução futura, em momentos predeterminados.

A Figura 2.1 traz uma representação dessa forma de operação. A figura não mapeia fielmente os componentes da arquitetura de operação de uma missão, mas apresenta um modelo conceitual útil para o entendimento do paradigma.

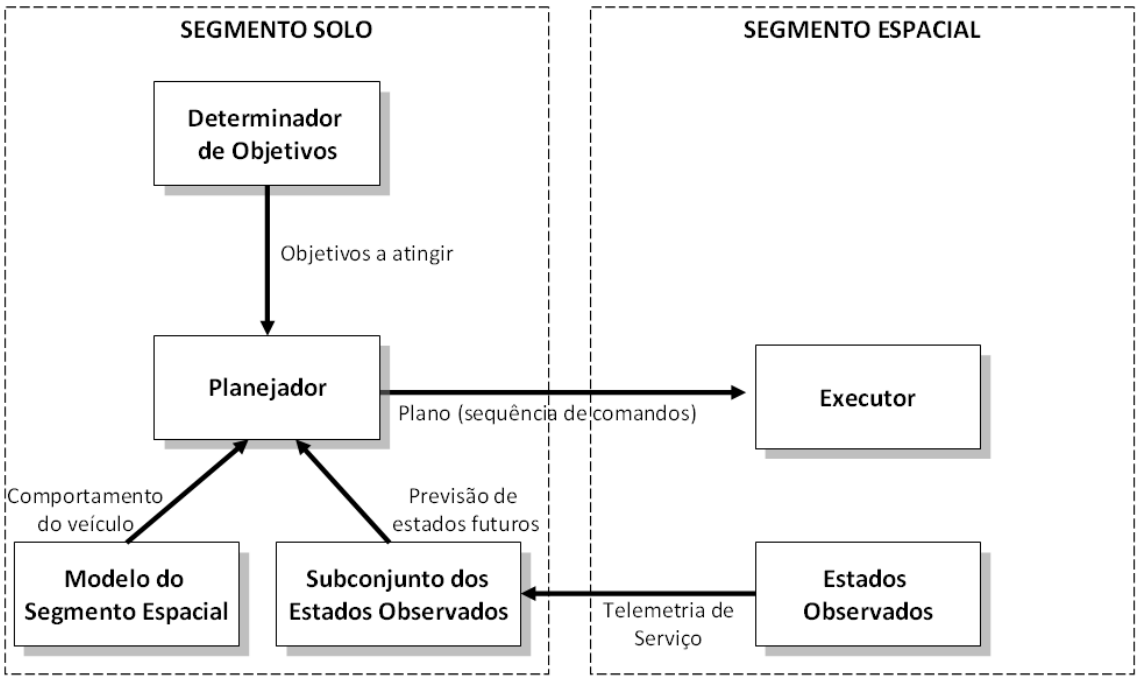


Figura 2.1 - Modelo conceitual da operação baseada em sequências de comandos

Na operação baseada em sequências de comandos, um Determinador de Objetivos, geralmente representado pelos usuários da missão, determina os objetivos a serem atingidos pelo segmento espacial. Um exemplo de objetivo para um satélite de sensoriamento remoto seria *"adquirir imagens de alta resolução da Baía do Almirantado, na Antártica"* (onde se encontra a Estação Antártica brasileira Comandante Ferraz).

Cabe ao Planejador transformar o objetivo recebido em sequências de comandos a serem transmitidas para o satélite. Para isso ele deve identificar os comandos de preparação da câmera adequada para o imageamento de alta resolução, como aqueles

para ligar os aquecedores, determinar o modo de operação da câmera e fazer a obtenção das imagens.

Comandos correspondentes ao gravador de dados de missão também devem ser gerados, para que as imagens obtidas sejam armazenadas para posterior transmissão para solo. Após a obtenção das imagens, a câmera, seus aquecedores e o gravador de dados devem ser desligados ou colocados em seus modos não-operacionais através de novos comandos.

A execução desses comandos deve ser sincronizada com o momento em que o satélite estiver sobrevoando a Baía do Almirantado, e para isso o Planejador deverá consultar os modelos orbitais, obtendo referências temporais a partir de posições orbitais previstas. Talvez seja necessária também uma manobra de ajuste de atitude para realizar o apontamento correto da câmera, que deve ser igualmente sincronizada.

Durante o planejamento de tal operação, o Planejador consulta o histórico de estados passados do satélite, que lhe foram transmitidos via telemetria de serviço, e realiza, a partir de um modelo comportamental do segmento espacial, uma inferência para determinar os estados futuros, aqueles que se espera encontrar no momento de início da operação de imageamento.

O resultado do processo de planejamento é uma sequência de comandos a ser enviada ao satélite, cada comando sendo associado a um momento de execução futuro.

Tal forma de operação pode ser realizada em solo por planejamento automatizado ou não. Quando o planejamento é automatizado, o modelo é explícito e a figura do Planejador é composta pelo operador humano atuando em cooperação com um *software* de planejamento. Quando o planejamento não é automatizado, o Planejador é o próprio operador humano, que trabalha com um modelo mental do comportamento do veículo espacial, baseado na documentação da missão e em sua própria experiência de operação.

A sequência de comandos enviada ao satélite assume que os estados futuros previstos estão corretos, o que nem sempre é verdade. Dvorak et al. (2008, p. 2) afirmam que "para que tal sequência de comandos funcione como esperado, é essencial prever com precisão os estados do veículo espacial e seu ambiente, no momento da execução de cada comando".

Devido às limitações na banda de comunicação de serviço, o Planejador em solo, seja ele automatizado ou não, trabalha com um subconjunto dos estados do satélite, registrados a intervalos maiores do que os intervalos de amostragem a bordo. Os registros de estados disponíveis em solo são defasados temporalmente, com um atraso que pode chegar a horas.

Além da dificuldade em se determinar com precisão o comportamento futuro do satélite, deve-se considerar ainda a possibilidade da ocorrência de imprevistos. Por exemplo, a câmera pode não atingir sua temperatura ideal de operação antes de iniciar a captura de imagens. Independente disso, a sequência de comandos será executada conforme o programado.

Numa situação como essa as imagens adquiridas apresentarão distorções, e toda a operação terá sido perdida. Pode levar semanas até que o satélite passe novamente sobre a mesma região, e quando o fizer, a informação originalmente necessária ao usuário da missão pode não ser mais relevante.

A principal limitação da operação por sequências de comandos é que elas devem ser predeterminadas em solo, com uma antecedência que em alguns casos chega a uma semana, e preveem pouca ou nenhuma flexibilidade. Se algo ocorre de forma diferente do planejado no momento da execução, o máximo que se pode fazer é abortar a sequência e, a depender da falha, colocar o satélite em um modo seguro – o que usualmente causa o desligamento das cargas úteis.

A criação de planos com tamanha antecedência e a falta de flexibilidade durante a

execução faz com que os operadores sejam conservadores e adotem margens de segurança no consumo dos recursos disponíveis a bordo, não otimizando o uso do satélite.

A tendência é que futuras missões sejam cada vez mais complexas e, quanto maior a complexidade da operação, maiores as limitações apresentadas pelas sequências de comandos. Em algum momento as sequências passarão a ser um gargalo operacional.

Outras formas de sequências de comandos vêm sendo implementadas nos últimos anos, como os comandos em resposta a eventos (ECSS, 2003) e os procedimentos de controle a bordo (ECSS, 2010). Essas formas adicionais de operação ajudam na flexibilização dos comandos, mas sofrem das mesmas limitações apresentadas pelas sequências de comandos 'simples', uma vez que também são planejadas em solo.

2.2. O Paradigma de Operação Baseada em Objetivos

O presente trabalho adota o conceito de objetivo conforme estabelecido por Dvorak et al. (2007, p. 5): "um objetivo é uma especificação da intenção do operador, ou seja, uma especificação do que se espera que um sistema faça". A operação de missões baseada em objetivos consiste em comandar o segmento espacial dizendo *o que* deve ser feito, ao invés de *como* fazê-lo.

O conceito foi colocado em prática pela primeira vez em 1999, quando a sonda *Deep Space One* da NASA gerou, em duas ocasiões, planos de operação detalhados a partir de objetivos recebidos de solo para a realização de tarefas de comunicação, propulsão e outras (MUSCETTOLA et al., 2002).

A Figura 2.2 ilustra o modelo conceitual de operação baseada em objetivos.

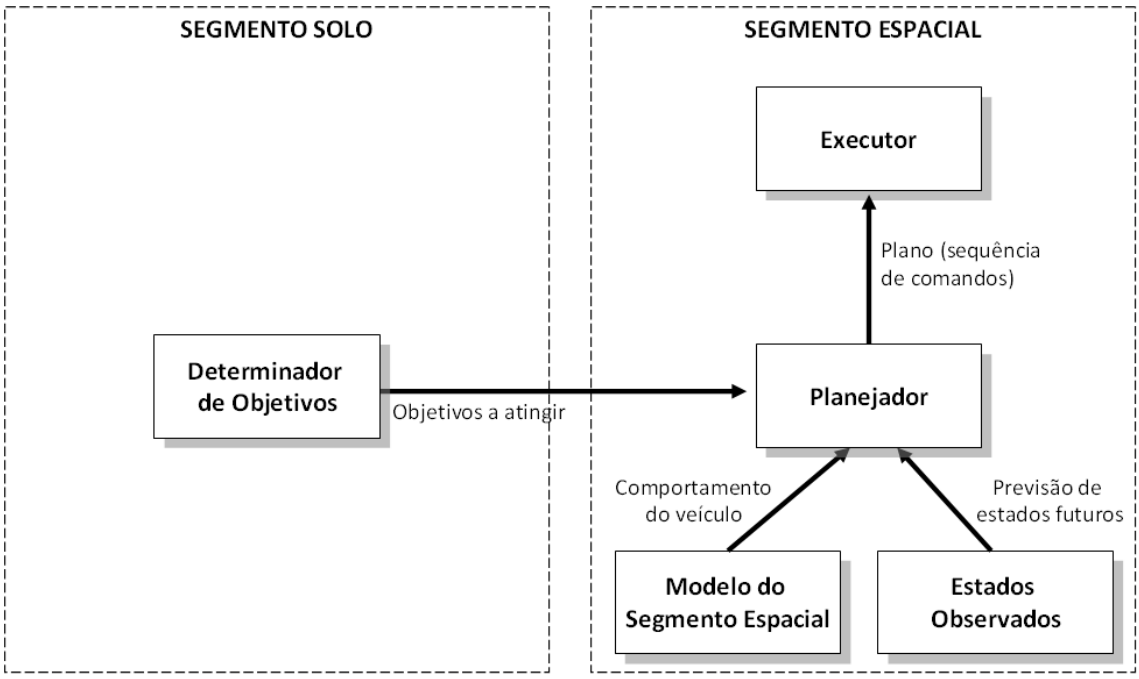


Figura 2.2 - Modelo conceitual da operação baseada em objetivos

Nota-se que os componentes e as interações são basicamente os mesmos presentes na operação baseada em sequências de comandos. A ideia central da operação baseada em objetivos é a transferência do processo de planejamento, ou ao menos parte dele, para o segmento espacial.

O segmento espacial passa a receber objetivos a atingir (*o que fazer*), ao invés de comandos a executar. Fica a cargo dele a determinação de quais comandos levarão ao cumprimento dos objetivos (*como fazer*). Dvorak et al (2007, p. 2) afirmam que "o princípio chave por trás dessa evolução [a operação baseada em objetivos] é a especificação da intenção do operador. Prover um sistema da especificação da intenção é dotá-lo de capacidade para verificar o cumprimento bem-sucedido de objetivos, e de utilizar métodos alternativos para cumpri-los, caso necessário".

Sendo executado a bordo, o Planejador tem acesso completo e em tempo real aos estados do satélite, o que potencialmente melhora a qualidade do processo de inferência de estados futuros. Além disso, os estados inferidos podem ser averiguados

regularmente, disparando correções caso sejam detectados desvios entre o que foi previsto e o que se está aferindo. Reduzem-se assim as limitações impostas pelo planejamento antecipado em solo e também pela falta de flexibilidade dos planos – uma vez que um plano pode ser adaptado a bordo em tempo real, em função de anormalidades ou imprevistos detectados durante sua execução.

Os benefícios imediatos dessa abordagem são uma maior abstração na operação de missões, reduzindo o esforço do operador em solo, e portanto o custo de operação, bem como uma potencial melhora na qualidade dos planos gerados, devido a uma maior observabilidade dos estados a bordo. De acordo com Dvorak et al. (2007), é mais fácil especificar o que fazer do que como fazer; o objetivo é explícito, compacto e inspecionável, o que facilita também a sua verificação.

Outro benefício de operar missões por objetivos é que a transferência do processo de planejamento para bordo abre caminho para a operação total ou parcialmente autônoma do segmento espacial, o que traz uma série de novas possibilidades para as missões.

2.3. Operação Autônoma do Segmento Espacial

O *European Committee for Space Standardization* (ECSS) aponta a execução a bordo de 'operações orientadas a objetivos' como o maior nível de autonomia para a operação nominal de missões (ECSS, 2008).

Dotar o segmento espacial de liberdade para decidir a melhor forma de atingir os objetivos que lhe são passados representa, certamente, um aumento considerável em sua autonomia – mas é possível ir além.

Um determinador de objetivos embarcado, associado ao planejador, pode prover ao segmento espacial a capacidade de reagir autonomamente a falhas, situações imprevistas e oportunidades, tirando proveito de seu acesso completo e em tempo real aos estados observados.

A Figura 2.3 apresenta uma variação do modelo conceitual da operação baseada em objetivos, onde se adiciona ao segmento espacial um Determinador de Objetivos. Tal modificação é o que permite a operação total ou parcialmente autônoma do segmento espacial.

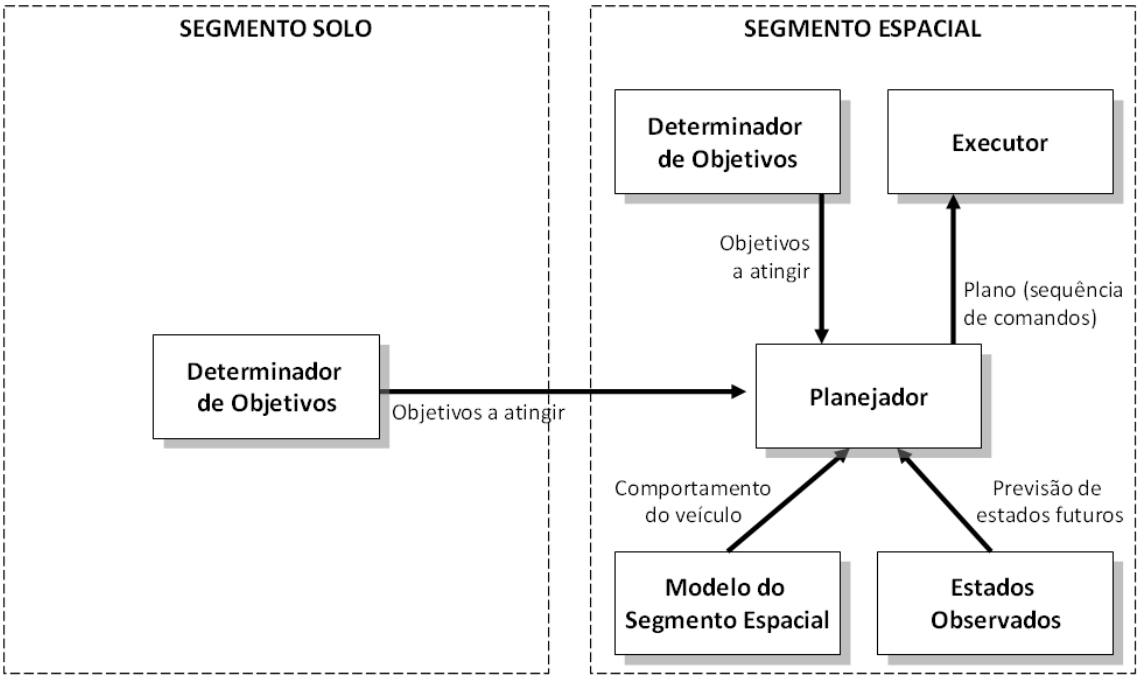


Figura 2.3 - Modelo conceitual da operação autônoma do segmento espacial

A principal razão para se adicionar um Determinador de Objetivos ao segmento espacial é reduzir a dependência de comunicação com o segmento solo em determinadas situações, que demandem respostas rápidas.

Retornemos ao exemplo de operação de um satélite de sensoriamento remoto dado na seção 2.1. Caso ocorra alguma falha durante a execução do plano, como a câmera não atingir sua temperatura ideal de operação, um Determinador de Objetivos embarcado poderia tentar corrigir a situação em tempo hábil, enviando ao Planejador um novo objetivo: *"levar a câmera à temperatura ideal de operação em 'n' minutos"*.

Isso poderia determinar a ativação de um conjunto de aquecedores redundante, ou alguma reconfiguração da câmera. Se ainda assim não se atingir a temperatura ideal a

tempo, o objetivo original poderia ser modificado, postergando o início da operação em alguns segundos.

Manter um Determinador de Objetivos e um Planejador próprios dá ao segmento espacial uma capacidade de reação muito superior à das missões atuais. Satélites de monitoramento ambiental poderiam modificar imediatamente seus planos de forma a obter mais imagens de queimadas, enchentes ou tempestades em formação que fossem detectadas. Satélites científicos teriam a capacidade de reagir a fenômenos físicos de curta duração, como a ocorrência de fenômenos elétricos na alta atmosfera ou erupções de raios-x provenientes de galáxias distantes.

É importante enfatizar que os conceitos de operação apresentados neste Capítulo são evoluções uns dos outros, não sendo, portanto, mutuamente exclusivos. Uma dada missão pode adotar qualquer combinação possível entre as operações baseadas em sequências de comandos, em objetivos ou operações autônomas, de acordo com suas necessidades.

2.4. Riscos versus Benefícios do Aumento da Autonomia no Segmento Espacial

Wertz e Reinert (1999, p. 29) afirmam que “com maior capacidade computacional a bordo, é claramente possível criar satélites completamente autônomos. A questão é: nós deveríamos fazer isso, ou deveríamos continuar a controlar os satélites predominantemente a partir de solo?”.

A pergunta é colocada porque deve existir uma solução de compromisso entre os potenciais benefícios do aumento da autonomia e os riscos que as decisões autônomas podem trazer a uma missão. Uma decisão incorreta pode colocar a missão em risco.

No geral, e ao contrário do que normalmente se acredita, veículos espaciais possuem baixo nível de autonomia. Um livro recente da NASA dedicado ao tema de sistemas autônomos na área espacial (TRUSZKOWSKI et al., 2009) apresenta estimativas aproximadas dos níveis atuais de autonomia em missões espaciais, que seguem

transcritas na Tabela 2.1.

Tabela 2.1 - Estimativas aproximadas do nível de autonomia em voo

Atividade	Nível atual de autonomia
Planejamento e escalonamento de tarefas	Baixo
Diagnóstico de falhas	Baixo
Correção de falhas	Baixo
Análise de dados de engenharia / calibração	Baixo
Processamento de dados científicos / calibração	Baixo
Execução da agenda científica (comandos a carga úteis)	Médio
Execução de atividades de suporte à ciência (cargas úteis)	Médio
Monitoramento de dados e desempenho	Médio
Atividades de suporte à engenharia a bordo	Alto

Fonte: adaptado de Truszkowski et al. (2009, Table 3.2).

Nota-se que quanto maior o nível de conhecimento e raciocínio necessário para a realização de uma dada tarefa, menor o nível de autonomia. Por esse motivo a única atividade operacional com alto nível de autonomia é o suporte à engenharia de bordo, que engloba tarefas repetitivas e rotineiras. Já o planejamento e escalonamento, o diagnóstico e a correção de falhas, por exemplo, apresentam baixo nível de autonomia.

Do ponto de vista do segmento solo, o mais imediato benefício de um veículo espacial com maior autonomia é a redução do número e da complexidade das atividades necessárias para sua operação de rotina.

Quanto aos riscos, cabe aos engenheiros da missão determinar quais decisões podem ser tomadas de forma autônoma sem que isso implique em risco para a missão. O aumento da autonomia deve ser gradual e baseado no ganho de confiança sobre os sistemas, possivelmente habilitando uma a uma as funções autônomas de um sistema embarcado. Toda decisão autônoma deve ser desabilitável, e os comandos provenientes de solo devem sempre ter precedência com relação aos comandos gerados a bordo.

Há que se considerar ainda que se uma decisão autônoma pode colocar uma missão

em risco, o mesmo se aplica a qualquer decisão do operador em solo. Um sistema de *software* autônomo, entretanto, não está sujeito ao cansaço ou a mudanças de humor. Uma vez que seu funcionamento tenha sido validado, ele se comportará da mesma forma, sempre – afinal, como todo *software*, é um sistema determinístico.

O aumento da autonomia libera o operador para tarefas menos repetitivas. Com mais tarefas sendo realizadas a bordo, ele passa a ter o papel de monitorar se o sistema autônomo executa corretamente aquilo que lhe foi incumbido.

Isso é algo que já ocorre nas missões atuais, com o subsistema de controle de atitude.

Atualmente, todo satélite com controle de atitude ativo realiza essa tarefa em malha fechada, coletando dados de um conjunto de sensores, processando tais dados e determinando comandos de correção e manutenção de atitude a serem executados pelos atuadores. Cabe a solo a tarefa de calcular e enviar ao satélite os parâmetros orbitais utilizados pelo algoritmo de controle, monitorar se a atitude encontra-se dentro do esperado e atuar apenas em caso de falha.

A operação das cargas úteis não é mais complexa que o controle de atitude; trata-se apenas de um tipo diferente de problema – um que envolve não apenas cálculos matemáticos, mas também decisões lógicas.

Os próprios Wertz e Reinert (1999, p. 31) concluem que “o maior problema encarado por satélites autônomos é a tradição. [A equipe de operações em] Solo trabalha dessa forma principalmente porque sempre o fez assim. Há, entretanto, sinais de mudança”.

3. PROJETOS VOLTADOS À OPERAÇÃO BASEADA EM OBJETIVOS NA ÁREA ESPACIAL

Como colocado anteriormente, o conceito de operação baseada em objetivos implica em inferência de estados e planejamento realizados no segmento espacial. Embora a literatura registre que apenas duas missões tenham utilizado *software* com tais características até a presente data, existem outros projetos desenvolvidos ou em desenvolvimento visando aplicações futuras.

Este Capítulo traz um histórico de sistemas desenvolvidos e em desenvolvimento com inferência de estados e planejamento, voltados ao segmento espacial. São ilustradas as arquiteturas de cada sistema, de forma a identificar os componentes comuns e enfatizar as particularidades de cada solução. Apresentam-se as aplicações, lógica de funcionamento e dados de desempenho dos sistemas. Alguns trabalhos em áreas correlatas à espacial também são brevemente descritos.

3.1. Projetos e Sistemas Desenvolvidos pela NASA

3.1.1. O Estudo New Millennium Autonomy Architecture Prototype

O início do desenvolvimento de planejadores embarcados em satélites data de 1995, quando foi realizado um estudo conjunto entre o *Ames Research Center* (ARC) e o *Jet Propulsion Laboratory* (JPL), ambos da NASA. Tal estudo fazia parte do programa *New Millenium* de validação tecnológica e foi batizado de '*New Millennium Autonomy Architecture Prototype*' (NewMAAP) (DVORAK et al., 2000).

O NewMAAP unificou um conjunto de conceitos, como o de comandos baseados em objetivos, controle em ciclo fechado, diagnóstico baseado em modelos, gerenciamento de recursos a bordo e planejamento e escalonamento baseados em Problemas de Satisfação de Restrições (*Constraint Satisfaction Problems*, CSP). Esses conceitos foram

integrados aos mecanismos tradicionais de monitoramento e controle em tempo real, resultando numa arquitetura de *software* para sistemas de controle autônomo de veículos espaciais.

Um protótipo da arquitetura foi implementado e demonstrado em uma simulação de inserção orbital em Saturno para a missão Cassini (PELL et al., 1996), cujos resultados foram considerados um sucesso.

Os principais autores desse estudo lideraram posteriormente o desenvolvimento dos únicos sistemas com planejamento embarcado a serem executados no espaço até a presente data: o *Remote Agent Experiment* e o *Autonomous Sciencecraft Experiment*, ambos descritos neste Capítulo.

3.1.2. O Remote Agent Experiment

Quando a sonda *Deep Space One* (DS-1) foi anunciada como uma missão de validação tecnológica do programa *New Millenium*, o estudo NewMAAP foi rapidamente convertido no projeto *Remote Agent Experiment* (RAX), cujo desenvolvimento ficou a cargo do ARC, com colaboração do JPL.

A missão da sonda DS-1, lançada em 1998, era realizar encontros com corpos celestes e obter imagens durante os sobrevoos. A principal tecnologia validada pela DS-1 foi seu sistema de propulsão iônica, mas diversas outras foram testadas, como o RAX.

A função do RAX foi gerar planos de operação a partir de objetivos recebidos de solo. O experimento ocorreu em duas ocasiões no mês de maio de 1999, totalizando cerca de vinte e seis horas de execução. Nesse período, o RAX gerou planos para tarefas de controle de atitude, comunicação, propulsão, observações e navegação, tornando-se o primeiro *software* a realizar planejamento embarcado no segmento espacial. Alguns erros na criação dos planos foram detectados – relatados por Bernard et al. (1999) –, mas o experimento foi considerado bem-sucedido.

A arquitetura do RAX, apresentada na Figura 3.1, integra três camadas de funcionalidade: um planejador/escalonador baseado em restrições, um executivo reativo, e um sistema de identificação de modos e recuperação de falhas.

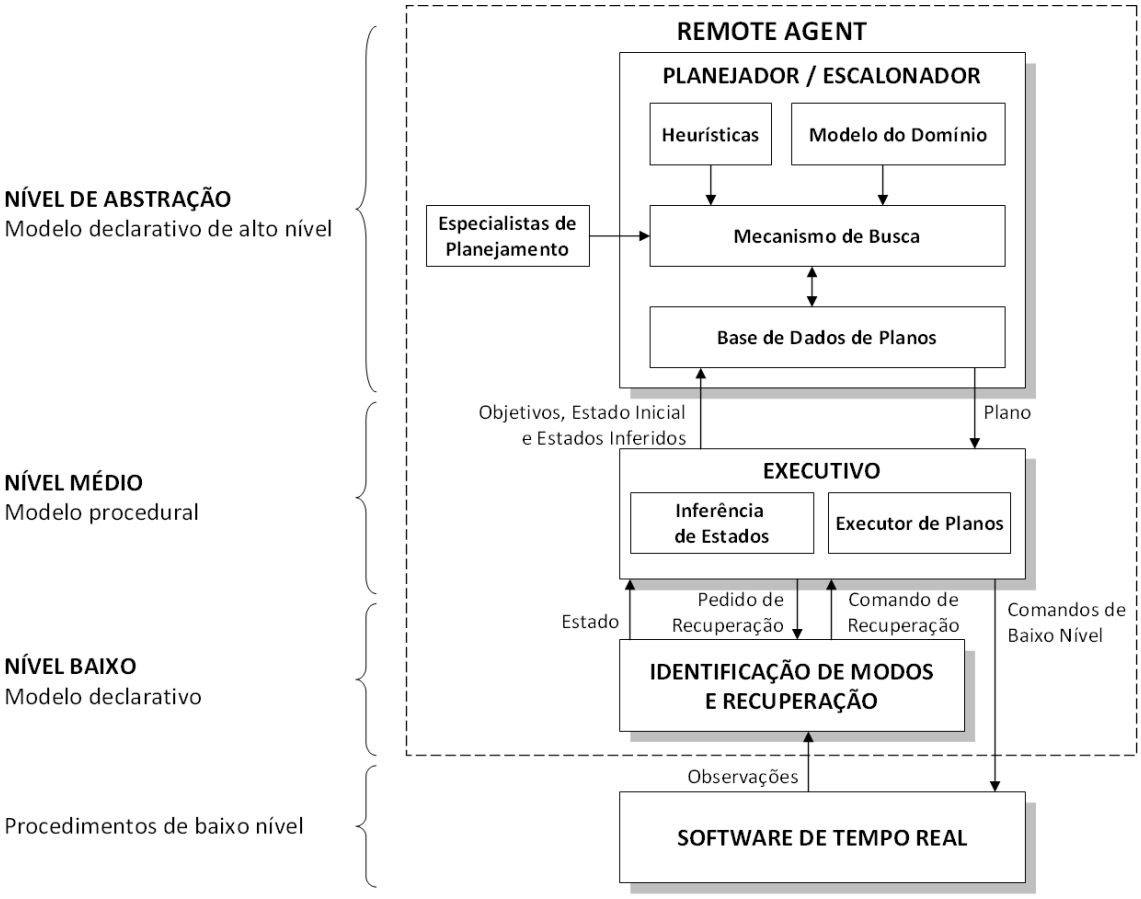


Figura 3.1 - Arquitetura do Remote Agent Experiment

Fonte: adaptado de Jónsson et al. (2000, p. 1) e Muscettola et al. (2002, p. 2).

O processo de planejamento é iniciado pelo módulo Executivo, que envia ao Planejador um conjunto de objetivos recebidos de solo para serem cumpridos, o estado inicial e os estados inferidos para a sonda.

O Planejador do RAX manipula objetivos em um CSP, mais especificamente em uma especialização de CSP chamada de Redes de Tarefas Hierárquicas (*Hierarchical Task Network*, HTN). Ele foi baseado em um sistema de solo originalmente desenvolvido para o telescópio espacial Hubble, chamado *Heuristic Scheduling Testbed System* (HSTS

– MUSCETTOLA et al., 1995).

Tal sistema gera planos temporalmente flexíveis a partir dos objetivos recebidos, consultando para isso um Modelo do Domínio e fazendo uso de Heurísticas durante o processo de planejamento. Componentes '*ad-hoc*' chamados Especialistas de Planejamento permitem a comunicação com sistemas externos.

O Modelo do Domínio descreve a dinâmica do sistema para o qual o plano é criado. Uma solicitação de plano, consistindo de um estado inicial e um conjunto de objetivos a serem atingidos, inicializa a Base de Dados de Planos, que armazena o plano em desenvolvimento.

Quando um plano válido é gerado, ele é enviado ao Executivo na forma de atividades de alto nível. Cabe ao Executivo decompor as atividades em comandos de baixo nível, enviando-os para execução e monitorando o resultado. O Executivo possui um modelo das atividades e alguma liberdade na sua decomposição, podendo tentar métodos alternativos caso algum comando falhe.

Se isso não for suficiente para corrigir a falha, o plano é abortado e um pedido de recuperação é enviado ao módulo de Identificação de Modos e Recuperação. Esse módulo incorpora uma ferramenta chamada Livingstone (WILLIAMS et al., 1996), que possui um modelo próprio do domínio utilizado para inferir os estados do satélite e monitorá-los.

Esse modelo é diferente daquele utilizado para o planejamento. Verma et al. (2005a, p. 94) relatou que "uma das duas falhas experimentadas pelo RAX foi originada por diferenças sutis e não documentadas nas semânticas entre os modelos das camadas de planejamento, execução e diagnóstico".

O tempo de resposta do RAX, entre a recepção dos objetivos e o envio de um plano para atingi-los, variou de 30 minutos a 4 horas, dependendo das características do problema de planejamento. Tal resultado foi obtido com alocação de 50% do tempo de

um processador PowerPC RAD6000 operando a 20 MHz, com 32 MB de memória RAM disponível (MUSCETTOLA et al., 2002).

3.1.3. O Autonomous Sciencecraft Experiment

Outro fruto direto do estudo NewMAAP foi o planejador embarcado *Continuous Activity Scheduling, Planning, Execution and Replanning* (CASPER), desenvolvido pelo JPL. O CASPER consiste de uma versão embarcada do *Automated Scheduling and Planning Environment* (ASPEN – FUKUNAGA et al., 1997), um planejador automatizado para o segmento solo utilizado em diversas missões da NASA.

O CASPER foi desenvolvido originalmente para aplicação na constelação de satélites universitários *Three Corner Sat* (3CS) e no Techsat-21 da Força Aérea Norte-Americana (KNIGHT et al., 2001).

A função do CASPER nos 3CS, que possuíam câmeras como carga útil, era o gerenciamento das imagens adquiridas. Baseado na análise da qualidade de uma imagem adquirida, o CASPER determinava se a mesma seria armazenada ou descartada, e qual a sua prioridade na transmissão para solo. Todas as tarefas relacionadas ao armazenamento e transmissão eram então planejadas a bordo.

No Techsat-21, que também possuía câmeras, o CASPER seria responsável pelo replanejamento da missão em função de anomalias ou de eventos detectados pela análise das imagens a bordo, como parte de um experimento chamado *Autonomous Sciencecraft Experiment* (ASE).

Entretanto, não houve oportunidade para se averiguar o funcionamento do CASPER em voo nessas missões: os satélites do 3CS foram perdidos durante o lançamento em 2004, e a missão Techsat-21 foi cancelada.

A terceira missão do CASPER, e a única a conseguir executá-lo, foi a do satélite de sensoriamento remoto *Earth Observing One* (EO-1) do *Goddard Space Flight Center*

(GSFC), lançado em 2000. Esse satélite, assim como a sonda DS-1, faz parte do programa *New Millenium*. Isso, e o fato de o EO-1 já haver completado sua vida útil prevista de um ano, foram decisivos para a aceitação do experimento.

A função do ASE no EO-1 é uma variação das suas duas missões anteriores: possibilitar a resposta autônoma do satélite à enchentes, derretimento de gelo ou erupções vulcânicas detectadas nas imagens adquiridas.

Para isso, algoritmos de reconhecimento de padrões analisam as imagens obtidas a bordo na tentativa de identificar eventos de interesse, como por exemplo uma erupção. Em caso positivo, novos objetivos são gerados e enviados ao planejador CASPER, que deve modificar os planos de operação atuais de forma a adquirir mais imagens sobre o evento nas próximas órbitas.

O CASPER não se propõe a ser um sistema de uso geral; ele foi adaptado a diferentes missões, em função dos contratempos e oportunidades que surgiram, e todas suas aplicações foram voltadas a um mesmo tipo de objetivo. Parte do esforço de adaptação do CASPER ao satélite EO-1 é relatado por Tran et al. (2004).

A Figura 3.2 ilustra a arquitetura do ASE no EO-1.

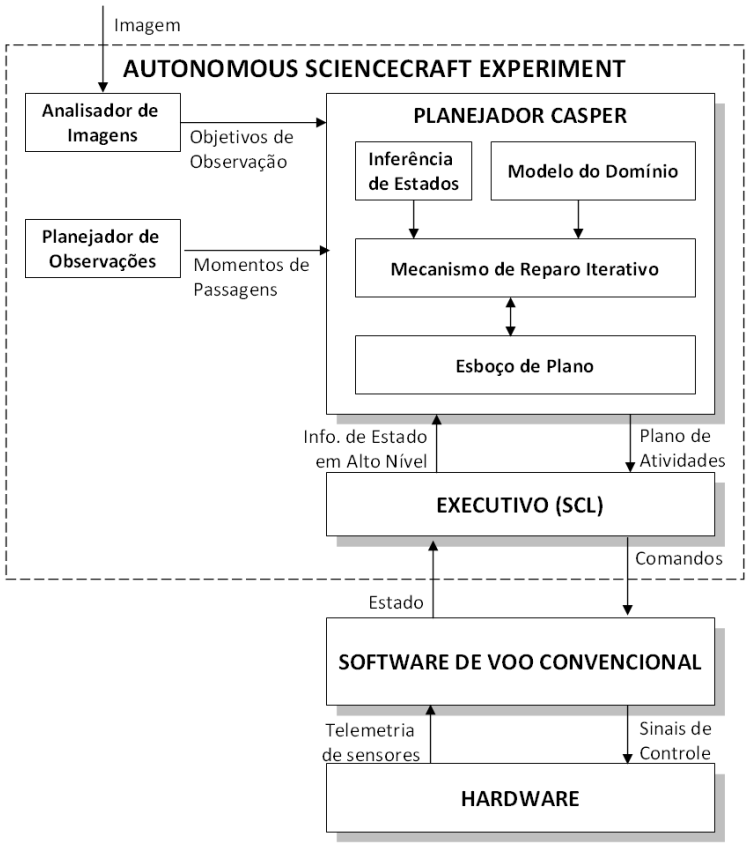


Figura 3.2 - Arquitetura do Autonomous Sciencecraft Experiment no EO-1

Fonte: adaptado de Chien et al. (2003, p. 2).

Um Analisador de Imagens é o responsável por determinar se há eventos de interesse nas imagens que foram adquiridas. Em caso positivo, o Analisador determina novos objetivos de observação, para a aquisição de mais imagens da mesma área nas órbitas seguintes, e os envia ao planejador CASPER.

Assim como o RAX, o CASPER trata o problema de planejamento como uma HTN.

Para gerar respostas autônomas em tempo hábil para as novas observações, o CASPER adota uma abordagem de planejamento chamada de 'reparo iterativo'. Nessa abordagem, o processo de planejamento é iniciado com um Esboço de Plano com falhas. A partir desse esboço, o CASPER trabalha para reparar as falhas, uma a uma, utilizando um algoritmo de busca local. A correção encontrada para uma falha no plano pode inserir novas falhas, e assim o CASPER inicia um novo ciclo de reparo. Esse

ciclo se repete até que surja um plano válido, sem falhas.

Um Executivo faz a interface entre o CASPER e o *software* de voo, enviando informações de estado do satélite para a inferência de estados e recebendo um plano de atividades a executar. Esse Executivo faz uso de uma linguagem chamada *Spacecraft Command Language* (SCL), para transformar o plano em sequências de comandos.

Através do reparo iterativo no planejamento, o ASE é capaz de gerar respostas em algumas dezenas de minutos (SHERWOOD et al., 2004), tempo suficiente para colocar o plano em execução na próxima passagem sobre o evento observado. Tal desempenho foi aferido em um processador R3000 Mongoose V operando a 8 MIPS, com alocação de CPU de 50% e consumo de 40 MB de memória RAM (TRAN et al., 2004).

Os primeiros testes em voo do ASE ocorreram em outubro de 2003. Testes incrementais foram realizados desde então, até que o sistema foi aprovado para uso em operações nominais em abril de 2005. O EO-1 continua operacional até a presente data, e o ASE segue sofrendo atualizações e otimizações.

Estima-se que o ASE tenha contribuído para uma economia de cerca de um milhão de dólares por ano na operação do EO-1, ao mesmo tempo em que propiciou um aumento considerável no volume de dados obtidos da missão (CHIEN et al., 2010).

3.1.4. Intelligent Distributed Execution Architecture

Com base na experiência obtida com o RAX, o ARC iniciou por volta de 2001 o desenvolvimento de uma nova arquitetura para sistemas de planejamento embarcado, chamada *Intelligent Distributed Execution Architecture* (IDEA - MUSCETTOLA et al., 2002).

Na IDEA, as camadas funcionais (de monitoramento, planejamento e execução)

presentes nas arquiteturas do RAX e ASE são substituídas pelo conceito de ‘agente’. Cada agente possui uma mesma estrutura fundamental, e todos utilizam um mecanismo comum para troca de informações na forma de mensagens.

A arquitetura IDEA, representada na Figura 3.3, replica-se para cada agente criado para uma dada aplicação.

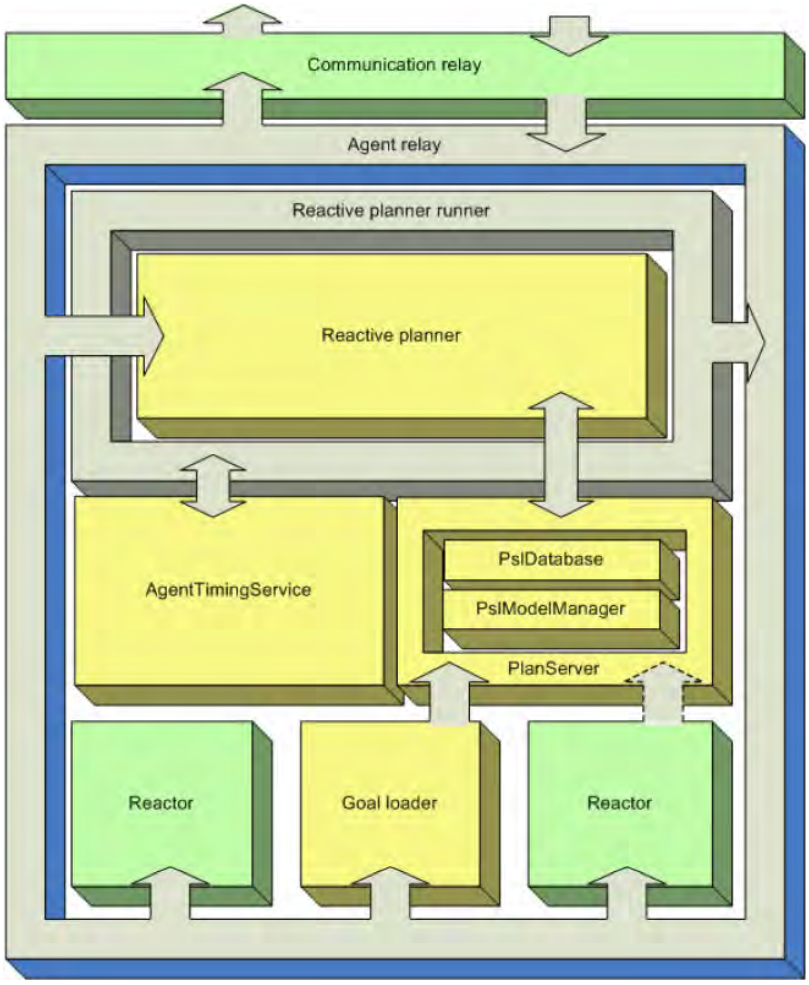


Figura 3.3 - Arquitetura de um agente da IDEA

Fonte: Aschwanden et al. (2006, p. 4).

O *Communication Relay* realiza as trocas de mensagens entre os agentes de uma aplicação. Cada agente possui um *Agent Relay*, responsável por encaminhar as mensagens recebidas para o devido componente interno, e as mensagens enviadas para outros agentes, pelo *Communication Relay*. O *Agent Timing Service* tem a função

de sincronizar todos os agentes da IDEA com uma mesma referência temporal.

Reactors são componentes que associam mensagens recebidas a ações, o que permite reagir imediatamente a situações de falha ou contingências detectadas, que demandam resposta rápida. Um tipo de *Reactor* é o *Goal Loader*, que recebe objetivos, os converte em restrições e os encaminha ao *Plan Service Layer* (PSL) *Database*.

A principal inovação da IDEA é o uso de um único modelo para todas as camadas/agentes. Enquanto o RAX e o CASPER possuem modelos de domínio distintos para cada camada (um modelo de inferência, um de planejamento e um de execução), a IDEA possui apenas um modelo, compartilhado por todos os agentes.

A unificação do modelo evita a ocorrência de problemas como o detectado durante a execução do RAX onde, de acordo com Verma et al. (2005a), diferenças de semântica entre modelos distintos levaram cada camada a prever um comportamento ligeiramente diferente do domínio. Isso originou um erro de difícil identificação, uma vez que cada modelo foi considerado válido, separadamente.

É o PSL que mantém o modelo do domínio e o motor de inferência da IDEA. Ele atualiza constantemente os estados representados no modelo em resposta a eventos ocorridos, mudança de valores de parâmetros, ou pela passagem do tempo. Os efeitos de cada atualização são propagados para todos os elementos do modelo. O PSL provê mecanismos para verificar a consistência do modelo, comparando os estados inferidos aos estados observados do domínio.

O núcleo do PSL é composto pela *Extendable Uniform Remote Operations Planning Architecture* (EUROPA – FRANK et al., 2003), usada para a modelagem e inferência de estados. A EUROPA é uma arquitetura para a resolução de CSPs com ênfase em redes de restrições temporais relacionadas a HTN. Ela é uma evolução do HSTS, e é atualmente utilizada em diversos sistemas de planejamento em solo de missões da NASA.

O planejador da IDEA pode ser deliberativo (criando planos completos, como no caso do RAX) ou reativo (modificando planos existentes, como no CASPER). No primeiro caso, o horizonte de planejamento deve ser mais longo, e o tempo de resposta será maior. No segundo, o horizonte deve ser extremamente curto para permitir respostas em tempo real. Dessa forma, um agente IDEA se propõe a realizar o replanejamento contínuo, unificando as tarefas de planejamento e execução para operações nominais.

Nessa abordagem unificada, partes de um plano são liberadas para execução sem que o plano esteja completo. O resultado do trecho do plano executado é lido por sensores e realimentado no processo de planejamento, que gera com isso um novo trecho do plano a executar. Assim, o planejador reage aos efeitos das partes de seu plano já executadas, adaptando-se aos novos estados lidos. Um plano gerado dessa forma não é ótimo, mas em princípio permite lidar em tempo real com situações inesperadas.

A IDEA foi testada em *rovers* de pesquisa da NASA em terrenos simulados da superfície de Marte. Os objetivos enviados nos testes envolviam atividades de aquisição e rastreamento de alvos a estudar (rochas), posicionamento de instrumentos e captação de imagens.

Embora a IDEA se proponha a ser um sistema embarcado de planejamento, sua implementação não reflete isso; os *rovers* utilizados nos testes possuem microcomputadores comerciais com diferentes configurações (com processadores Pentium de 300 MHz e 1.3 GHz), e possuindo como sistema operacional o *Linux*. Em tais condições, dados de desempenho não são representativos para uma aplicação embarcada.

Apesar de não ter sido adotado por nenhuma missão espacial, o IDEA foi usado recentemente como base para outros projetos voltados à operação baseada em objetivos, conforme descrito nos itens 3.2.4 e 3.5.2 deste Capítulo.

3.1.5. Coupled Layer Architecture for Robotic Autonomy

Outra arquitetura criada a partir da experiência obtida com o RAX e o ASE é a *Coupled Layer Architecture for Robotic Autonomy* (CLARAty – NESNAS et al., 2003), cujo projeto foi iniciado por volta de 2000 em um esforço conjunto do JPL, ARC, *Carnegie Mellon University* e *University of Minnesota*.

A CLARAty é uma arquitetura de *software* reutilizável, projetada para reduzir o esforço de integração, teste e amadurecimento de tecnologias robóticas para missões futuras da NASA (ESTLIN et al., 2007).

Inicialmente a CLARAty funcionou como uma referência sobre como unificar sistemas preexistentes de inferência de estados, diagnóstico, planejamento e execução, não sendo ela própria uma implementação. Para uma dada missão, a arquitetura depende da composição de diferentes módulos e interfaces *ad-hoc* para funcionar.

A arquitetura se baseia em duas camadas: uma Camada Funcional de primitivas robóticas (baixo nível) e uma Camada de Decisão com capacidades de planejamento e execução (alto nível). A arquitetura prevê também interfaces padrão para a comunicação entre os elementos de cada camada.

A Camada Funcional deve prover um conjunto de capacidades robóticas genéricas, organizadas em *software* de forma hierárquica, que fazem interface com o *hardware* do sistema.

A Camada de Decisão deve prover recursos para criar e executar de forma autônoma sequências de ações para atingir objetivos operacionais. Isso envolve a inferência de estados, planejamento, escalonamento de tarefas e execução monitorada.

A Figura 3.4 apresenta as camadas da CLARAty e suas interfaces.

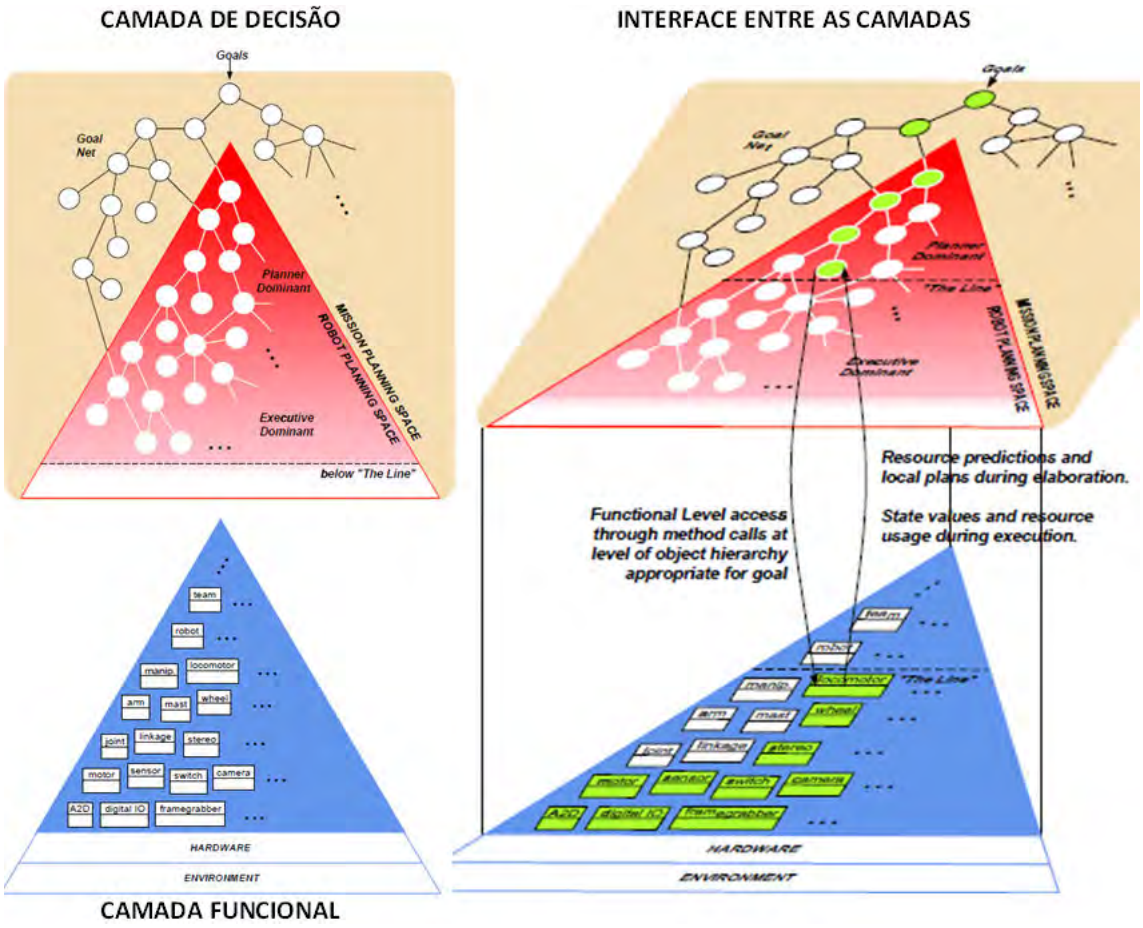


Figura 3.4 - Camadas da CLARAty e suas interfaces

Fonte: Adaptado de Volpe et al. (2001, p. 5, 7 e 8).

Embora a CLARAty não seja por si só uma implementação de planejador embarcado e motor de inferência, ela permite a integração de diferentes tecnologias com um fim comum: aumentar a autonomia de missões robóticas.

Um trabalho desenvolvido pelo *Research Institute for Advanced Computer Science* (RIACS) com a finalidade de estudar a verificação e validação de sistemas espaciais autônomos (JÓNSSON et al. (2005) e BRAT et al. (2006)) adotou a EUROPA como mecanismo de planejamento e um sistema de execução chamado *Plan Execution Interchange Language* (PLEXIL – VERMA et al., 2005b) para a Camada de Decisão da CLARAty.

Outra implementação da arquitetura resultou no sistema *Closed Loop Execution and Recovery* (CLEaR – CHOUINARD et al., 2003). O CLEaR implementa a Camada de Decisão da CLARAty de forma a reproduzir a estrutura de planejamento e execução do ASE no EO-1, adotando o CASPER e substituindo o SCL por um sistema executor chamado *Task Description Language* (TDL – SIMMONS e APFELBAUM, 1998). A TDL se baseia em um modelo próprio de representação de tarefas e transições entre tarefas.

Assim como a IDEA, o CLEaR foi executado em *rovers* de pesquisa da NASA, em ambientes simulados. Os cenários de teste iniciavam com um plano de atividades onde o *rover* precisaria alcançar certo número de rochas a imagear, seguindo um caminho predeterminado. Ao executar o plano original, obstáculos não antes visualizados eram detectados pelo *rover*, o que disparava um processo de replanejamento do caminho a percorrer para que o maior número de rochas possível ainda fosse alcançado e analisado até o fim do dia.

Os cenários, executados pela primeira vez em 2001, deram início a uma série de novos desenvolvimentos baseados na CLARAty e no CLEaR, que culminaram em um aumento significativo da autonomia dos *Mars Exploration Rovers* (MER) em 2009.

3.1.6. Onboard Autonomous Science Investigation System

Os resultados obtidos dos cenários de simulação do CLEaR levaram à criação do projeto *Onboard Autonomous Science Investigation System* (OASIS – ESTLIN et al., 2008). O OASIS é uma implementação da Camada de Decisão da CLARAty que tem por objetivo permitir que o segmento espacial identifique e responda de forma autônoma a possíveis elementos de interesse científico.

Ele é capaz de extrair elementos de interesse de imagens adquiridas, analisar e priorizar o potencial valor científico dos elementos extraídos. Em função disso, o OASIS planeja e executa uma nova sequência de comandos para adquirir mais imagens. O planejador também é responsável por priorizar a transmissão das imagens para o

segmento solo.

Atualmente o OASIS é capaz de detectar e reagir a nuvens, redemoinhos pequenos e rápidos (conhecidos como '*dust devils*') e rochas com características distintas das demais em uma mesma cena. Elementos do OASIS foram sendo gradualmente testados e integrados aos *rovers* da missão MER, aumentando sua capacidade de resposta autônoma.

Em 2006, algoritmos de detecção de *dust devils* foram incorporados ao *software* de voo de ambos os *rovers*, *Spirit* e *Opportunity* (CASTAÑO et al., 2006). O resultado foi um aumento no envio de imagens com flagrantes de *dust devils* na superfície marciana.

Em dezembro de 2009, outro elemento do OASIS chamado *Autonomous Exploration for Gathering Increased Science* (AEGIS – JPL, 2009), foi adicionado ao *software* de voo do *rover Opportunity*. O AEGIS faz uso de um conjunto de algoritmos para detectar rochas que apresentem propriedades de interesse científico, durante travessias realizadas pelo *Opportunity* entre um ponto e outro da superfície marciana. Caso algo seja detectado, o *rover* pode se deter, obter imagens da rocha, e seguir seu caminho conforme o planejado pela equipe de operações em solo.

O AEGIS é executado junto ao *software* de voo do *Opportunity*, sem interferir com outras tarefas do *rover*. Embora implemente o CASPER para o planejamento de atividades a bordo, tal componente não foi enviado ao *Opportunity*, sendo substituído por sequências de comandos previamente definidas em uma linguagem procedural chamada *Virtual Machine Language* (VML).

Isso, segundo os autores, deve-se ao fato de eles não haverem conseguido fazer com que o planejador se adeque às restrições computacionais do *rover*. Os autores, entretanto, esperam conseguir embarcar o componente de planejamento em versões futuras do AEGIS (ESTLIN et al., 2010).

O tempo de execução de um ciclo de detecção-resposta (sem planejamento embarcado) em um processador PowerPC RAD6000 a 20 MHz, alocando 4 MB de memória RAM sob o sistema operacional VxWorks, varia de 15 a 20 minutos (BORNSTEIN et al., 2010).

Os desenvolvedores do OASIS têm planos de embarcar uma versão completa do sistema no *Mars Science Laboratory* (MSL), lançado em novembro de 2011 e com chegada a Marte prevista para agosto de 2012. Esse novo experimento provavelmente será tentado após o MSL completar sua vida útil prevista, de pouco menos de dois anos – mas poderá ser postergado caso o MSL demonstre longevidade similar à dos MER, adiando um eventual exercício do OASIS, completo, no segmento espacial.

Há também estudos conceituais para a aplicação do OASIS em uma futura sonda aérea, voltada à exploração da lua saturnina Titã, descritos por Gaines et al. (2010).

3.1.7. O Projeto Mission Data System

Em 1998 o JPL iniciou o projeto *Mission Data System* (MDS) com o objetivo de repensar o ciclo de vida de *software* em missões espaciais e propor mudanças para melhorar a arquitetura de *software* e seus processos de desenvolvimento.

Desde o início o projeto foi focado no conceito de operação baseada em objetivos, havendo realizado estudos em diversas áreas correlatas: modelagem de sistemas espaciais, planejadores em solo, replanejadores embarcados, interfaces com usuários e outros.

Embora o projeto tenha elaborado o conceito de operação baseada em objetivos e analisado suas implicações além de qualquer outro, ele nunca teve como meta gerar *software* embarcado. Entretanto, um conjunto de padrões de projeto para esse tipo de *software* foi proposto por Bennett et al. (2008), dentro do escopo do MDS.

Ao longo dos anos o MDS foi capturando conceitos e experiências dos demais projetos

da NASA descritos neste Capítulo, agregando-os em uma única arquitetura de referência, similar àquelas nas quais foi inspirada. A arquitetura é apresentada na Figura 3.5.

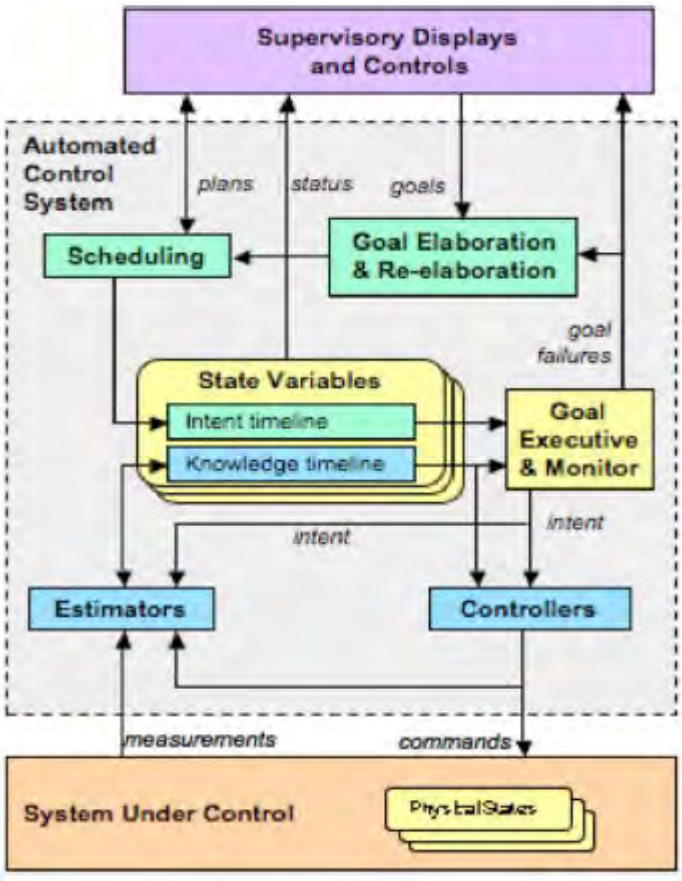


Figura 3.5 - A arquitetura de controle proposta pelo Mission Data System

Fonte: Wagner et al. (2009, p. 4).

Mais recentemente o grupo de trabalho do MDS vem deixando o estudo de missões robóticas autônomas e concentrando esforços no suporte robótico a futuras missões tripuladas, como pode ser verificado em Wagner et al. (2009), Wagner et al. (2010), Braman e Wagner (2011).

3.2. As Iniciativas Europeias

3.2.1. O Estudo Mars Mission On-Board Planner and Scheduler da ESA

Os conceitos de operações baseadas em objetivos e autonomia do segmento espacial

vêm sendo exercitados pela ESA no contexto da missão *ExoMars*. Essa missão é uma cooperação da ESA com a NASA para levar a Marte um *rover* europeu similar aos MER. O lançamento foi adiado algumas vezes, e atualmente está previsto para 2018.

Visando tal missão, a ESA contratou em 2003 um estudo chamado *Mars Mission On-board Planner and Scheduler* (MMOPS – WOODS et al., 2006), com o objetivo de determinar um nível adequado de autonomia para a *ExoMars* e estudar a viabilidade da adoção do planejamento embarcado no segmento espacial.

O estudo, encerrado em 2006, culminou no desenvolvimento de um protótipo para uma ferramenta de monitoramento da execução de planos, e reparos de falhas detectadas nesses planos, chamada *Timeline Validation, Control and Repair* (TVCR). A arquitetura da TVCR é apresentada na Figura 3.6.

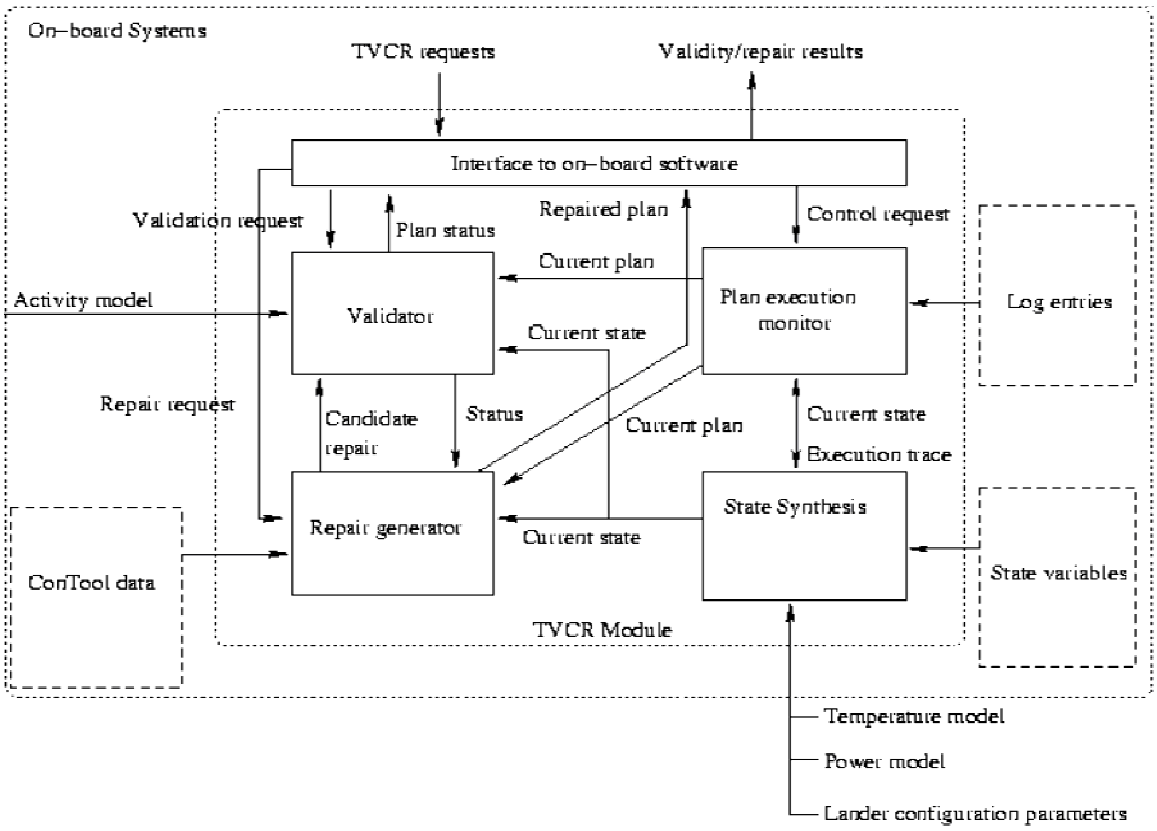


Figura 3.6 - A arquitetura do TVCR

Fonte: Woods et al. (2006, p. 7).

O componente central é o *Validator*. Esse componente recebe do *Plan Execution Monitor* o plano de atividades atual e, a partir do estado corrente do domínio enviado pelo *State Synthesis*, realiza a inferência de estados para determinar se há alguma falha no plano.

Para isso, o *Validator* depende de um modelo de sequências de atividades, que descrevem as precondições necessárias para que uma atividade execute com sucesso, e o efeito de tal atividade nos estados do domínio. O modelo é descrito em *Planning Domain Description Language* (PDDL – FOX e LONG, 2003).

Caso uma falha seja detectada, o *Repair Generator* é ativado para corrigi-la. O *Repair Generator* não executa planejamento; basicamente, ele tenta ordenar sequências de ações previamente definidas em solo (tratadas como 'fragmentos de plano'), de forma a corrigir a falha. O processo de validação, controle e reparo ocorre de forma contínua.

Os trabalhos publicados não deixam claro se o TVCR foi executado em uma simulação do *hardware* de voo, não integrado ao restante do *software*, ou em microcomputadores comerciais fazendo interface com a simulação do *hardware* de voo. Dados de desempenho também não foram apresentados.

Um trabalho recente (WOODS et al., 2009) descreve uma proposta que adota o TVCR como o núcleo de um sistema a ser embarcado no *rover* da missão *ExoMars*. Nessa proposta o objetivo básico é replicar as funções do OASIS/AEGIS nos MER da NASA, inclusive passando a realizar planejamento embarcado. Essa seria, entretanto, uma proposta ainda sem resultados publicados.

3.2.2. O Estudo Autonomy Generic Architecture: Tests and Applications

Em 2004 o CNES e o *Office National d'Etudes et de Recherches Aerospatiales* (ONERA) iniciaram um estudo conjunto sobre autonomia em sistemas espaciais. O estudo tinha entre os seus objetivos a identificação da viabilidade de altos níveis de autonomia no segmento espacial, a análise do impacto da autonomia nas operações, e o

desenvolvimento de um protótipo de sistema autônomo.

O estudo dividiu-se em tópicos como arquitetura, planejamento e diagnóstico autônomo. Os resultados das pesquisas foram aplicados a uma 'bancada de laboratório' (termo utilizado pelos autores) chamada *Autonomy Generic Architecture: Tests and Applications* (AGATA – CHARMEAU e BENSANA, 2005).

Alguns cenários hipotéticos de aplicações de autonomia em missões espaciais foram explorados durante o projeto. O cenário mais estudado baseou-se em um satélite de sensoriamento remoto equipado com um detector de nuvens. Em função da presença de nuvens sobre uma área a imagear, um planejador embarcado modifica o plano de operações corrente.

O estudo, que teve grande foco na modelagem baseada em restrições, apresentou uma proposta de arquitetura similar à do CASPER, e implementou um protótipo de planejador baseado num algoritmo de busca local gulosa (‘*greedy search*’, BEAUMET et al., 2011). Tal protótipo foi executado em microcomputadores comerciais, uma vez que a AGATA não tinha nenhuma preocupação com as restrições de *hardware* do segmento espacial.

3.2.3. O LAAS Autonomy Architecture

O *Laboratoire d'Analyse et d'Architecture des Systèmes* (LAAS) iniciou ainda na década de 1990 o desenvolvimento do *LAAS Autonomy Architecture* (ALAMI et al., 1998), que precedeu e provavelmente inspirou a definição da arquitetura CLARAty.

A arquitetura é ilustrada na Figura 3.7.

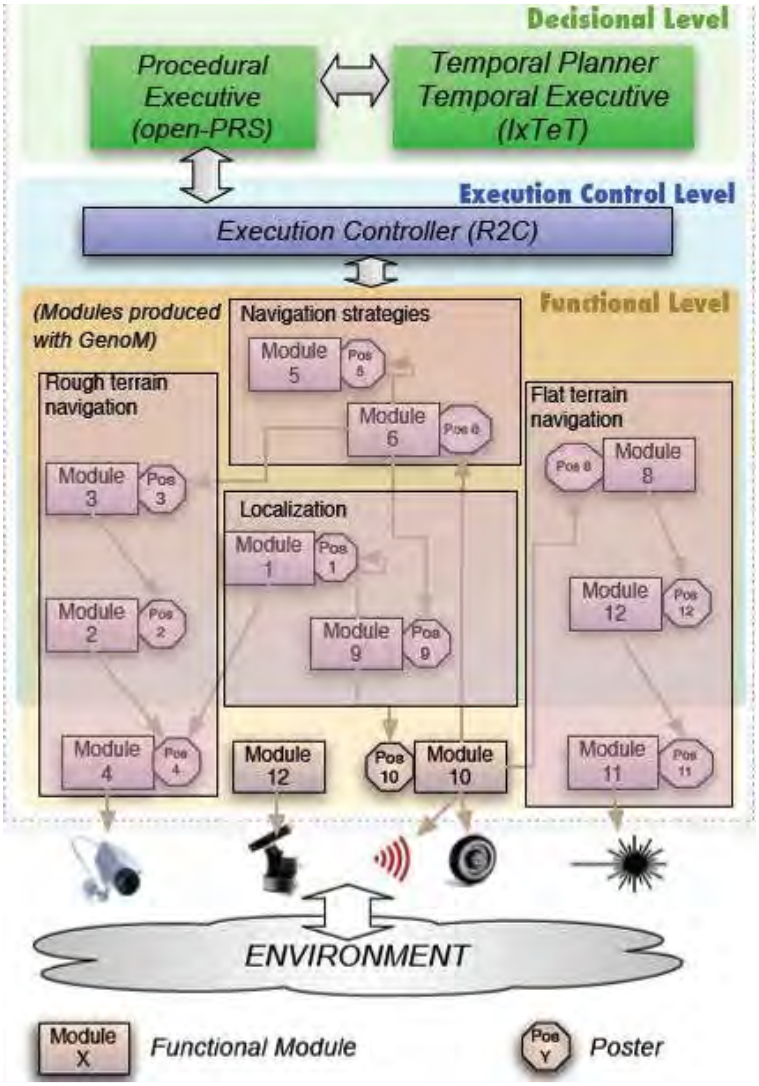


Figura 3.7 - A LAAS Autonomy Architecture

Fonte: Ingrand et al. (2007, p. 4).

Nota-se que a arquitetura possui as mesmas camadas de decisão e funcional da CLARAty, inserindo entre elas uma camada de controle de execução, equivalente aos executivos presentes no RAX e CASPER. O planejador temporal é capaz de realizar reparo iterativo baseado em CSP e redes temporais simples (*Simple Temporal Networks*, STN).

O principal cenário de estudos da *LAAS Autonomy Architecture* é a navegação autônoma de *rovers* em terrenos rochosos. Para isso, casos de testes foram executados em *rovers* de pesquisa, que possuíam um microcomputador Pentium IV de

3 GHz com 1 GB de RAM, rodando o sistema operacional *Linux*.

Embora o *hardware* dos *rovers* de pesquisa não seja compatível com o *hardware* de voo das missões atuais, uma versão simplificada da *LAAS Autonomy Arquitecture* foi executada, como um experimento, em um processador LEON (de uso espacial) operando a 40 MHz, com 32 MB de memória RAM disponíveis.

O estudo de caso envolvia um satélite de sensoriamento remoto voltado ao monitoramento de vulcões e queimadas, com características similares às do ASE no satélite EO-1. Esse experimento indicou um tempo de planejamento da ordem de 30 segundos. Os autores, entretanto, ressalvam que "os testes não foram muito representativos" (LEMAI et al., 2006, p. 7).

3.2.4. O Projeto Goal-Oriented Autonomous Controller

Em 2009 a ESA iniciou um projeto de dois anos de duração, similar em escopo e objetivos ao MMOPS (mas sem relação com este), chamado *Goal-Oriented Autonomous Controller* (GOAC – FRATINI et al., 2011). Esse novo projeto foi realizado em parceria com o LAAS, o *Instituto di Scienze e Tecnologie della Cognizione* (ISTC), o *Monterey Bay Aquarium Research Institute* e o laboratório de pesquisas francês Verimag.

Assim como feito com a arquitetura CLARAty, que culminou no uso do sistema OASIS a bordo dos *rovers* da missão MER, o projeto GOAC se propôs a integrar um conjunto de tecnologias existentes sob a *LAAS Autonomy Architecture*.

Para a camada de decisão da arquitetura foi adotado um sistema de planejamento embarcado chamado T-REX, desenvolvido originalmente para aplicações submarinas a partir da IDEA do ARC/NASA (para maiores informações, consulte o item 3.5.2 neste Capítulo).

O projeto, encerrado em 2011, gerou poucas publicações. Dois estudos de caso,

voltados a *rovers* de exploração planetária, foram propostos: um executado em um *rover* de pesquisa do LAAS, e outro em uma simulação computadorizada. Em ambos os casos os módulos componentes da GOAC foram integrados e submetidos apenas a um teste simples (CEBALLOS et al., 2011), como uma prova de conceito.

3.3. Um Protótipo para o Planejamento Embarcado no INPE

Um primeiro estudo sobre planejamento embarcado no segmento espacial das missões do INPE foi iniciado em 2005, e resultou no protótipo de planejador embarcado baseado em CSP chamado *Resources Allocation Service for Scientific Opportunities* (RASSO – KUCINSKIS, 2007).

O estudo de caso selecionado para o protótipo foi o satélite EQUARS (atualmente uma das duas missões do satélite Lattes), composto por um conjunto de experimentos científicos para a pesquisa da alta atmosfera terrestre.

O RASSO visava a realocação de recursos, especificamente energia e memória, para experimentos científicos que detectassem a ocorrência de fenômenos físicos de curta duração como *sprites* (fenômenos elétricos) ou perturbações ionosféricas.

Operando com mais recursos do que o originalmente programado, o experimento poderia funcionar por mais tempo e armazenar mais dados, fazendo uma melhor observação do fenômeno. Tal realocação deveria ser feita de forma a afetar o mínimo possível a operação dos demais experimentos.

A arquitetura do RASSO baseia-se em três módulos, ilustrados na Figura 3.8: o Modelo do Domínio, o Compositor de Problemas e o Planejador.

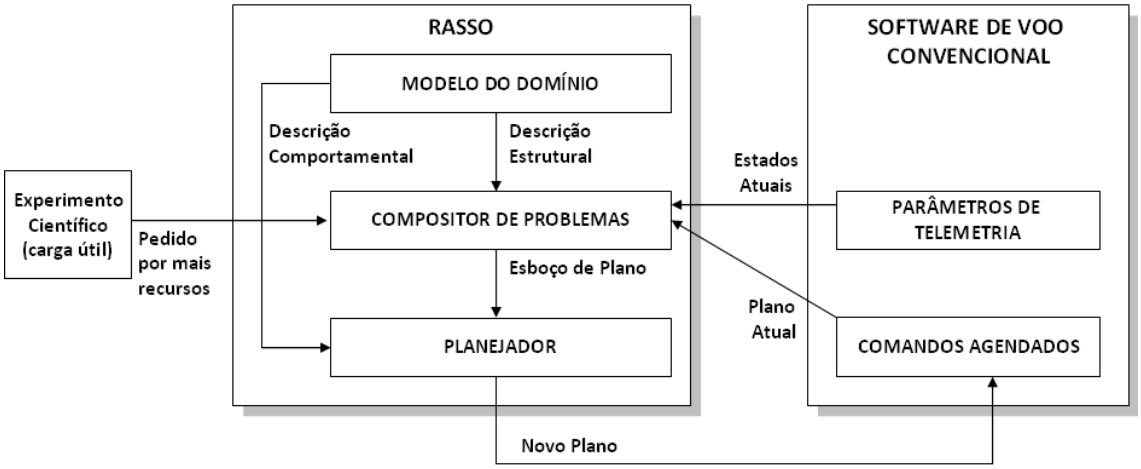


Figura 3.8 - A arquitetura do RASSO

Fonte: Adaptado de Kucinskis (2007, p. 102).

O Modelo do Domínio descreve a lógica e restrições de operação dos experimentos, e seu consumo de recursos quando ativados. Ele é composto por duas descrições: uma estrutural, com os elementos que fazem parte do domínio e seus relacionamentos, e uma comportamental, com a descrição dos efeitos das ações (comandos) e eventos exógenos (eventos externos que afetam o domínio, cujos efeitos estão fora do controle do sistema) sobre o domínio.

O processo de planejamento é iniciado quando um experimento científico envia um novo objetivo ao RASSO, na forma de um pedido por mais recursos. Nesse pedido o experimento informa os recursos necessários, sua quantidade, e por quanto tempo eles devem ser alocados.

Ao receber o objetivo, o Compositor de Problemas o converte em um conjunto de restrições aplicadas a um conjunto de variáveis, criadas a partir da Descrição Estrutural do domínio. As variáveis são inicializadas com os estados atuais obtidos do *software* de voo, e utilizadas durante o processo de planejamento para a realização de inferência de estados e análise de cenários.

O Compositor de Problemas então solicita do *software* de voo o plano de operações atual, que é enviado junto às variáveis ao Planejador, como um CSP, na forma de um

esboço de plano a ser trabalhado.

O Planejador inicia a busca por uma sequência de ações que levem a um plano válido, ou seja, que permita a realocação de recursos para o experimento que os solicitou, respeitando as restrições operacionais dos demais.

Assim como o CASPER, o RASSO implementa o reparo iterativo de planos, e utiliza um algoritmo de busca local guiada por restrições de escalonamento, sem heurísticas. Quando necessário, são adicionadas perturbações ao plano para escapar de mínimos locais e platôs durante a busca.

O RASSO foi executado em um protótipo de computador de bordo para a Plataforma Multimissão (PMM) do INPE, que possui um processador SPARC ERC32 operando a 12 MHz, com 4 MB de memória RAM, e utiliza o sistema operacional de tempo real RTEMS.

O tempo médio de resposta, desde a recepção do pedido por recursos até o envio de um novo plano, foi de cerca de 2 minutos, com 100% de utilização da CPU.

3.4. Linha do Tempo de Sistemas com Inferência de Estados e Planejamento Voltados ao Segmento Espacial

Com o objetivo de consolidar o histórico e a evolução dos estudos e sistemas apresentados neste Capítulo, foi criada uma 'linha do tempo'.

A evolução rumo a um sistema embarcado no segmento espacial foi categorizada em níveis, da seguinte forma:

1. Início de Estudos: marco do início de estudos conceituais e teóricos;
2. Início do Desenvolvimento: início do desenvolvimento dos sistemas;
3. Protótipo Não-Embarcado: criação de um protótipo de sistema executado em

computadores comerciais. Sistemas executados em *rovers* de pesquisa encontram-se nessa categoria, devido à grande capacidade de processamento e memória disponíveis, e ausência de restrições aplicáveis ao *software* de voo;

4. Protótipo em *Hardware* de Voo: criação de um protótipo de sistema executado em processadores de uso espacial, com capacidade de processamento e memória bastante limitados;

5. Integrado ao *Software* de Voo: sistemas integrados ao *software* de voo regular do segmento espacial, embarcados no *hardware* de voo;

6. Validado em Voo: sistemas executados em voo durante as missões.

Devido à falta de informação detalhada em algumas das referências utilizadas, algumas datas são aproximadas, com um erro estimado de não mais que um ano na maioria dos casos. A linha do tempo segue na Figura 3.9.

A figura também adianta a situação da arquitetura desenvolvida como parte do presente trabalho (chamada 'GOESA' e descrita no próximo Capítulo) com relação aos demais.

Figura 3.9 - Linha do tempo de sistemas com inferência de estados e planejamento voltados ao segmento espacial

3.5. Operação Baseada em Objetivos para Veículos Não Tripulados

Missões com veículos não tripulados, aéreos e submarinos, constituem áreas correlatas à operação de satélites. Assim como em missões espaciais, tais veículos são remotamente operados através de planos previamente criados em seus centros de controle.

Outra característica comum com a área espacial é a existência de períodos sem comunicação com o veículo durante o cumprimento da missão. Surgem então requisitos de autonomia similares aos descritos no Capítulo 2 desta Tese.

Encontram-se também nesse tipo de missão os conceitos de planejamento embarcado e operação baseada em objetivos. Os próximos subitens descrevem brevemente alguns trabalhos que adotam tais conceitos nas áreas de veículos não tripulados.

3.5.1. Veículos Aéreos Não Tripulados

Existem diversos trabalhos sobre planejamento embarcado para Veículos Aéreos Não Tripulados (VANTs). Entretanto, a grande maioria é focada no problema de cálculo e correção de rotas a serem percorridas ('*path planning*'), como em Wu et al. (2009), Chanthery et al. (2005) e Castro et al. (2009).

O planejamento de rotas envolve o processamento de mapas em três dimensões e em alguns casos a análise em tempo real de imagens – como no projeto Processamento de Imagens em Tempo Real (PITER – MARTINS et al., 2006) desenvolvido pelo Instituto de Estudos Avançados (IEAv) do Departamento de Ciência e Tecnologia Aeroespacial (DCTA).

Tal tipo de planejamento requer uma grande capacidade de processamento de dados a bordo do veículo. Assim, o ambiente computacional embarcado torna-se muito diferente do existente na área espacial. A pesquisa realizada como parte desta Tese encontrou poucos trabalhos voltados ao planejamento embarcado da operação das

cargas úteis de VANTs.

Um deles é o *Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance* (OPS-USERS). O OPS-USERS é um sistema de operação de VANTS proposto para dar suporte a operações militares de inteligência e combate da Marinha Norte-Americana (PADUANO, 2006).

Um dos objetivos do sistema é permitir que um único operador humano seja capaz de controlar múltiplos veículos para a execução de missões militares complexas, sendo para isso auxiliado por um módulo de planejamento embarcado e coordenação de ações entre os veículos.

Por se tratar de um produto militar, não são divulgados dados sobre técnicas ou soluções adotadas para o planejamento embarcado no OPS-USERS. Também não há informações a respeito da compra ou uso de tal sistema pela Marinha.

A NASA também iniciou um projeto, chamado *Intelligent Mission Management / Autonomous Robust Aviation*, para a criação de um sistema de operação autônoma e coordenação de frotas de VANTs com planejamento embarcado (WEGENER et al., 2004).

Esse projeto propôs uma arquitetura de gerenciamento da operação de VANTS em missões científicas e de monitoramento ambiental e adaptaria sistemas existentes para missões espaciais, como o RAX. O projeto chegou a gerar um plano de atividades para o período entre 2005 e 2009, mas não foi dada continuidade a ele.

3.5.2. Veículos Autônomos Subaquáticos

Os Veículos Autônomos Subaquáticos (*Autonomous Underwater Vehicles*, AUVs) são uma categoria relativamente nova de Veículo Submarino Não Tripulado (VSNT), voltada majoritariamente para a pesquisa do ambiente marinho. Eles são constituídos por submarinos robóticos com sensores diversos para resolver a interação com

fenômenos físicos, químicos e biológicos.

Sistemas de controle atuais para AUVs dependem de planos gerados manualmente e a priori (RAJAN et al., 2010). Um controlador embarcado gera respostas ao seu ambiente imediato, disparando ações sem considerar suas consequências. Assim como no caso espacial, há a necessidade de adaptação do veículo a oportunidades e situações inesperadas para um melhor aproveitamento da missão de pesquisa.

Um trabalho recente, surgido como um *spin-off* da área espacial, trouxe para os AUVs o planejamento embarcado voltado ao atendimento de objetivos de alto nível. Trata-se do *Teleo-Reactive Executive* (T-REX).

O T-REX vem sendo usado para encontrar, mapear e coletar amostras de camadas de partículas em suspensão transportadas do fundo do mar para a superfície, em regiões costaneiras. Para isso ele implementa uma arquitetura de *software* de planejamento e controle em ciclo fechado baseado no projeto IDEA do ARC/NASA (RAJAN et al., 2009).

O AUV utilizado possui dois computadores de bordo: um computador principal, baseado em um processador de 244 MHz com o sistema operacional de tempo real QNX, e um computador dedicado à execução do T-REX, com processador de 367 MHz e sistema operacional *Red Hat Linux*. Dados de desempenho do ciclo de planejamento não foram encontrados nos trabalhos publicados.

Os autores afirmam que esse é o único sistema que executa planejamento a bordo de AUVs. Um relato detalhado da aplicação e análise dos resultados do T-REX foi publicado por Das et al. (2012).

4. UMA ARQUITETURA DE SOFTWARE EMBARCADO EM SATÉLITES PARA AS OPERAÇÕES BASEADAS EM OBJETIVOS

Este Capítulo apresenta considerações sobre a pesquisa descrita no Capítulo anterior. A partir dessas considerações, são destacadas características importantes no segmento espacial para que se viabilize a operação baseada em objetivos como um paradigma de fato, independente de missão.

Uma arquitetura de *software* embarcado que possui tais características é proposta, e seus componentes são apresentados de forma geral. É colocada a preocupação com a integração da arquitetura com as próximas missões do INPE, e quais decisões foram tomadas para garantir que tal integração seja possível.

4.1. Considerações Sobre a Pesquisa Realizada

A pesquisa sumarizada no Capítulo 3 permitiu identificar um conjunto de características comuns nos sistemas estudados, discutidas nesta seção.

Primeiramente, todos os sistemas são baseados em modelos embarcados que descrevem o comportamento do veículo espacial. Tais modelos, e os motores de inferência de estados que operam sobre eles, são centrais em todas as arquiteturas.

Uma ou outra forma de CSP, em especial as Redes de Tarefas Hierárquicas, é adotada em todos os sistemas para a representação dos objetivos, e algoritmos de busca para a solução de CSPs são implementados para o planejamento. Isso certamente se deve à opção pioneira do NewMAAP pela adoção de tal técnica.

Embora isso não fique claro na representação de algumas das arquiteturas, quase todos os sistemas, à exceção do IDEA e do RASSO, separam o processo de busca por um plano em duas etapas: o planejamento da sequência de atividades e o

escalonamento das mesmas.

Conforme já descrito na introdução desta Tese, o planejamento determina a sequência de atividades para chegar a um estado objetivo, enquanto o escalonamento atribui recursos necessários à execução dessas atividades e as posiciona temporalmente em função de um conjunto de restrições dadas. Tal separação em sistemas embarcados é uma herança de planejadores de solo como o HSTS, usado como base para o RAX, e o ASPEN, que originou o CASPER. Aparentemente, os planejadores que os sucederam apenas seguiram a solução adotada por eles.

A separação entre planejamento e escalonamento impõe a existência de ao menos dois modelos distintos que descrevam um mesmo conjunto de atividades: um voltado ao comportamento do domínio, e outro às restrições temporais e recursos. As arquiteturas do RAX, ASE e CLARAty apresentam ainda um terceiro modelo, o de execução de planos.

Como detectado já na primeira experiência embarcada com o RAX, a existência de múltiplos modelos descrevendo características distintas e inter-relacionadas de um mesmo domínio abre a possibilidade para a ocorrência de erros de difícil detecção, gerados por diferenças sutis entre os comportamentos previstos por cada modelo. Surpreendentemente, o único sistema da NASA a levar isso em conta foi o IDEA.

Outra constatação é que sistemas desenvolvidos originalmente para plataformas do tipo PC não foram posteriormente adaptados para execução a bordo. Isso indica que as limitações do segmento espacial devem ser consideradas desde a concepção de um *software* que se proponha a ser embarcado.

É importante notar que, entre os poucos sistemas que realmente executaram em *hardware* de voo, bastante limitado computacionalmente, nenhum se propõe a ser de uso geral – esses foram desenvolvidos visando uma única missão ou um tipo de aplicação específica. As adaptações, quando existiram, foram para aplicações muito

similares, e assim mesmo à custa de considerável esforço, como o descrito por Tran et al. (2004).

Portanto, nota-se que não há um sistema embarcado em missões espaciais voltado à operação baseada em objetivos, que se proponha a ser aplicável a diferentes tipos de missões, e que unifique o planejamento e escalonamento, propiciando a existência de um único modelo de domínio.

4.2. Visão Geral da Arquitetura Proposta

O estudo dos projetos voltados ao segmento espacial para a operação baseada em objetivos demonstra que, para habilitar tal tipo de operação, o veículo espacial deve ser provido de:

- Modelos que descrevam o domínio da operação, em especial o das cargas úteis;
- Um motor de inferência de estados que faça uso desses modelos;
- Uma interface entre o motor de inferência e informações coletadas do satélite, como o estado dos equipamentos e o plano de operações corrente;
- Um gerenciador de objetivos, responsável pela conversão dos objetivos recebidos em dados manipuláveis pelo motor de inferência, bem como pela verificação da consistência e de eventuais conflitos entre múltiplos objetivos;
- Um planejador/escalonador embarcado;
- Para missões que tenham como característica a resposta autônoma a eventos detectados em órbita, um determinador embarcado de objetivos.

Foi desenvolvida como parte do presente trabalho a *Goal-based Operations Enabling Software Architecture* (GOESA), uma arquitetura de *software* que tem por finalidade habilitar o paradigma de operação de missões espaciais baseada em objetivos – em

detrimento do paradigma atual de operação baseada em sequências de comandos.

A arquitetura unifica as tarefas de planejamento e escalonamento, possuindo um modelo único do domínio. Ela se propõe a ser de uso geral, executável em *hardware* qualificado espacialmente, e integrável ao *software* de voo.

A GOESA foi concebida em função da pesquisa realizada sobre arquiteturas e soluções adotadas em sistemas similares, e da experiência anterior com o RASSO. Ela possui componentes para todas as funcionalidades listadas anteriormente nesta seção, como ilustra a Figura 4.1.

Quando em execução, a arquitetura recebe objetivos, que podem ser originados tanto em bordo como provenientes de solo, obtém informações do *Software* de Supervisão de Bordo e realiza inferência dos estados futuros do satélite baseada em um único modelo.

Um Planejador embarcado consulta e opera sobre o modelo de forma a gerar ou modificar um plano para atender aos objetivos recebidos. Ao conseguir isso, o novo plano de operações é enviado para execução pelo *Software* de Supervisão de Bordo.

O restante deste Capítulo descreve de forma geral os componentes da arquitetura.

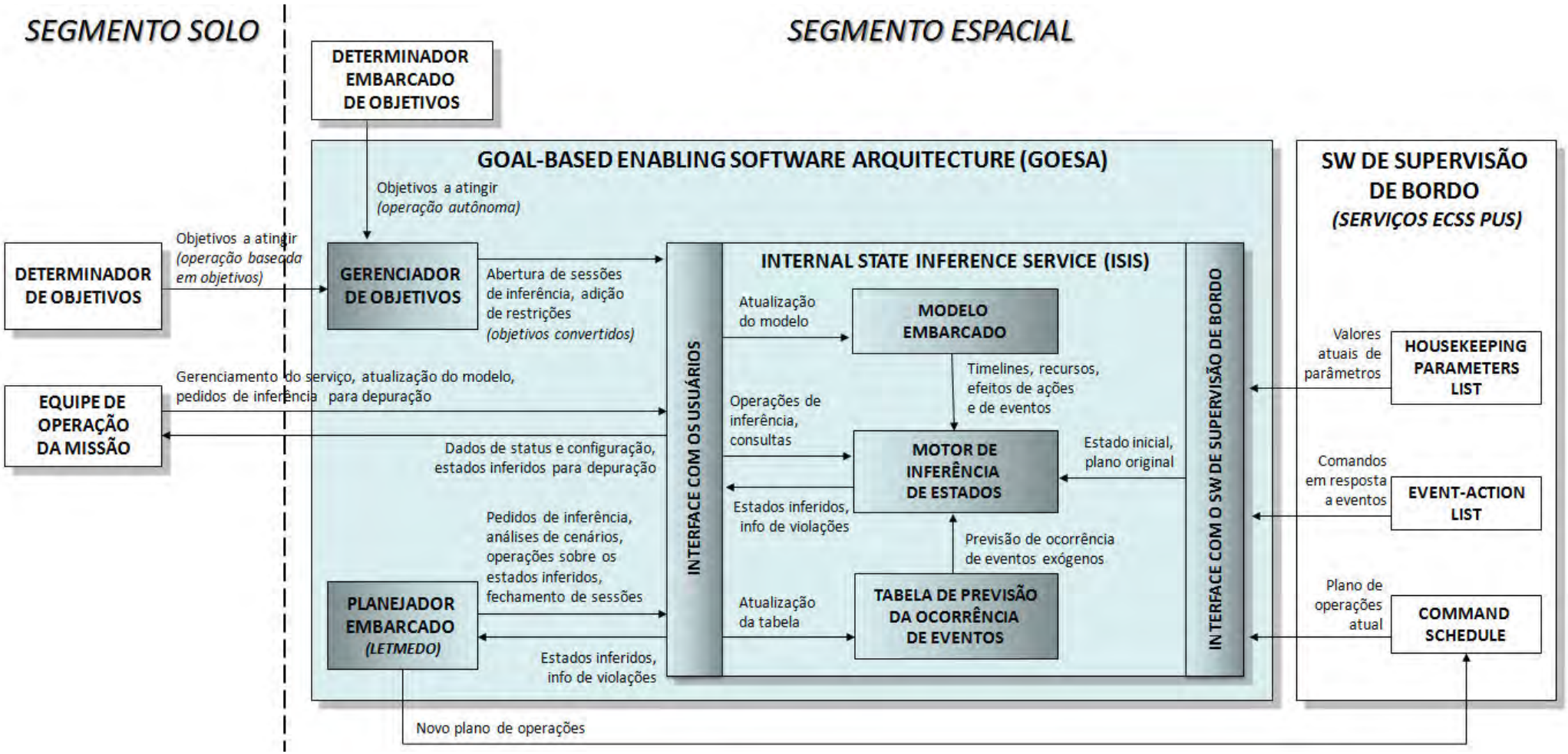


Figura 4.1 - A Goal-based Operations Enabling Software Architecture e suas interfaces

4.2.1. Os Determinadores de Objetivos

A GOESA pode receber objetivos a cumprir tanto do segmento solo (em um paradigma de operação baseada em objetivos) como do próprio segmento espacial (para a operação autônoma).

No segundo caso, a GOESA prevê a existência de um Determinador Embarcado de Objetivos, responsável pela detecção de eventos para os quais gerar respostas e pela definição dos objetivos em função de tais eventos.

A implementação de um Determinador Embarcado depende das características de uma dada missão e em especial de suas cargas úteis. Um exemplo de objetivo gerado autonomamente em um observatório espacial seria *"apontar os instrumentos e adquirir o maior volume de dados possível sobre a fonte de raios X que foi detectada agora"*.

4.2.2. O Gerenciador de Objetivos

O Gerenciador de Objetivos é responsável por receber objetivos, verificar sua consistência, se são operacionalmente factíveis, e se há conflitos entre eles. Após realizadas essas verificações, cabe também ao Gerenciador converter os objetivos em algo manipulável pelo Motor de Inferência de Estados.

Múltiplos objetivos podem ser recebidos, a qualquer momento durante a operação. A origem dos objetivos – se foram determinados em solo ou em bordo – não é relevante aos demais componentes da arquitetura.

4.2.3. O Internal State Inference Service

Com o objetivo de promover o reuso da arquitetura em diferentes missões, seus componentes centrais – um Modelo Embarcado, um Motor de Inferência de Estados e uma Interface com o *Software* de Supervisão de Bordo – foram unidos em um serviço de bordo, chamado *Internal State Inference Service* (ISIS).

Também faz parte do ISIS uma Tabela de Predição da Ocorrência de Eventos, necessária para prover ao Motor de Inferência informações sobre o comportamento futuro do satélite que não estão disponíveis no *Software* de Supervisão de Bordo. É possível fazer uma correlação operacional entre a Tabela de Eventos e os parâmetros orbitais enviados regularmente para um *software* de controle de atitude e órbita.

Os usuários do ISIS são a Equipe de Operação da Missão em solo, o Gerenciador de Objetivos e o Planejador Embarcado. Uma Interface com os Usuários recebe e processa pedidos de operações de inferência de estados, consultas aos estados previstos e tarefas de manutenção, como a configuração de parâmetros do serviço, a atualização do Modelo Embarcado e da Tabela de Eventos.

O Capítulo 6 descreve o ISIS, seu funcionamento interno e as operações e consultas disponibilizadas aos usuários do serviço.

4.2.3.1. O Modelo Embarcado

A GOESA contém um modelo de domínio único. O Modelo Embarcado traz informações sobre a composição do domínio, o comportamento de seus elementos, e dados para a interface com o satélite que estabelecem a relação entre itens modelados e seus correspondentes no *Software* de Supervisão de Bordo: dados obtidos de equipamentos e comandos enviados a eles.

A principal característica necessária para que a GOESA seja reutilizável entre missões é que ela seja independente do modelo que contém. Assim, o modelo é colocado como um componente substituível na arquitetura, podendo inclusive ser atualizado em voo.

Uma nova linguagem de modelagem foi definida especificamente para a criação dos modelos da arquitetura. Isso foi necessário devido à integração ao *software* de voo, e à unificação entre planejamento e escalonamento que a GOESA promove. O Capítulo 5 descreve a linguagem e o processo de modelagem estabelecidos.

4.2.3.2. O Motor de Inferência de Estados

O Motor de Inferência de Estados possui um metamodelo, no qual são consolidados os objetivos recebidos do usuário, o estado atual e o plano de operações corrente obtidos do satélite, e o comportamento apresentado pelo domínio, descrito no modelo.

É sobre essa informação consolidada no metamodelo que o Motor de Inferência opera, inferindo os estados futuros do satélite e disponibilizando os resultados de tal inferência ao Planejador.

O Motor de Inferência permite ao Planejador realizar 'operações de inferência', que consistem em adicionar, remover ou mover temporalmente os comandos presentes no plano de operações. A cada modificação no plano, os estados inferidos são atualizados. Aplicando diferentes operações de inferência, o Planejador exercita variações do plano até encontrar uma que leve ao atendimento completo dos objetivos.

4.2.3.3. A Interface com o Software de Supervisão de Bordo

Para que a inferência de estados possa ser realizada a bordo, é necessário haver uma correlação entre os elementos do modelo e os elementos reais do domínio. Isso é feito por uma Interface com o *Software* de Supervisão de Bordo, a partir de informações providas pelo modelo.

Todos os itens modelados devem estar relacionados a informações disponíveis ao *Software* de Supervisão de Bordo. Essa relação é estabelecida na descrição do domínio que o modelo contém, e é utilizada ao obter o estado atual do domínio modelado, ao ler o plano de operações corrente e ao enviar um novo plano para execução.

4.2.4. O Planejador Embarcado

O Planejador é o responsável por, a partir de consultas e operações de inferência aplicadas ao ISIS, encontrar a sequência de comandos que vai levar ao cumprimento dos objetivos que foram recebidos. Planejamento e escalonamento são realizados

conjuntamente: a sequência encontrada já considera o consumo de recursos e os momentos corretos de execução de cada comando.

Nenhum algoritmo de planejamento específico é imposto na arquitetura. Em princípio qualquer algoritmo pode ser utilizado, desde que seja adequado à execução em *hardware* de voo, e adaptado à interface provida pelo ISIS.

Como parte da presente Tese, um planejador foi implementado como referência para a arquitetura GOESA. Esse planejador é descrito no Capítulo 7.

4.3. Independência de Missões e Integração ao Software de Voo

Para que a GOESA seja realmente adaptável a diferentes missões, ela deve ser capaz de executar em *hardware* de voo ao mesmo tempo em que esteja integrada ao *software* de voo de satélites do Instituto – assim como são hoje os sistemas executores de sequências de comandos.

Em princípio, a integração ao *hardware* e *software* de voo e a independência de missões parecem características mutuamente exclusivas. Entretanto, a compatibilização entre elas é justamente o que vem sendo buscado pelo INPE dentro de seu programa para satélites da Plataforma Multimissão (PMM).

4.3.1. A Plataforma Multimissão do INPE e o Computador Avançado

De acordo com o Plano Diretor 2011-2015 do INPE, “a PMM é uma plataforma de uso múltiplo para satélites de até 500 kg de massa total em órbitas de 600 a 1000 km. A partir de 2012 e até 2020, o INPE planeja lançar oito satélites baseados na PMM, para aplicações de observação da terra (sensoriamento remoto, clima espacial e meteorologia) e científicas (astrofísica e geofísica espacial)” (INPE, 2011a, p. 15). Entre esses satélites encontram-se o Amazonia-1, o Lattes e o *Global Precipitation Measurement Brazil* (GPM-Br).

Um projeto de pesquisa e desenvolvimento vinculado à PMM é o do Computador

Avançado (COMAV). O COMAV é um computador de bordo para satélites que unifica as funções de Supervisão de Bordo (*On-Board Data Handling*, OBDH) e Controle de Atitude e Órbita (*Attitude and Orbit Control*, AOC), usualmente realizadas em outras missões do INPE por dois ou mais computadores (ARIAS et al., 2008).

O COMAV é baseado em um processador de uso espacial ERC32 (arquitetura SPARC V7 de 32 bits), cuja versão de baixo consumo elétrico pode operar a uma frequência de até 15 MHz, com 4 MBytes de memória RAM, e utiliza o sistema operacional de tempo real RTEMS. A GOESA foi projetada para execução nesse computador, havendo inclusive sido seguidas suas recomendações para o projeto e codificação de *software* (INPE, 2011b).

Por fazer parte da PMM, o *software* de voo do COMAV também é adaptável a diferentes requisitos de missão. A principal medida tomada no projeto desse *software* para garantir tal adaptabilidade foi a adoção do padrão *Packet Utilization Standard* (PUS) da ECSS.

4.3.2. O Packet Utilization Standard da ECSS

O PUS (ECSS, 2003) é um padrão da área espacial que define a interface, em nível de aplicação, entre os segmentos solo e espacial. Ele descreve conceitos de operação comuns a qualquer missão, e os agrupa em um conjunto de serviços providos pelo segmento espacial. Sua adoção permite o reuso de grande parte dos sistemas de solo e de bordo, uma vez que as interfaces de operação das missões são padronizadas.

Além dos serviços definidos pelo padrão, o PUS prevê a possibilidade de se agregar serviços adicionais, que se comuniquem com, e façam uso dos demais. Cada novo serviço deve trazer novas capacidades ao segmento espacial, não presentes naqueles já existentes.

O provimento de funcionalidades para habilitar a operação baseada em objetivos se encaixa nessa definição, e por esse motivo os componentes centrais da GOESA foram

projetados como um serviço adicional do PUS, o ISIS. A proposta de implementação do ISIS como um serviço PUS foi tema de um capítulo de livro do *American Institute of Aeronautics and Astronautics* (AIAA), relacionado à presente Tese (KUCINSKIS e FERREIRA, 2011).

A comunicação com outros serviços se dá principalmente através da Interface com o *Software* de Supervisão de Bordo. Ela acessa elementos como o *Housekeeping Parameters List* (um repositório com todos os dados coletados de equipamentos do satélite), o *Command Schedule* (lista de comandos temporizados que contém o plano de operações corrente) e o *Event-Action List* (lista de comandos a serem executados em função da ocorrência de eventos predeterminados).

Sendo compatível com o PUS, a GOESA é aplicável a qualquer missão que adote o padrão, não apenas às da PMM. Atualmente o PUS é utilizado em virtualmente todas as missões da ESA, além de estar presente também em missões canadenses, brasileiras e argentinas.

4.3.3. Reusabilidade dos Componentes da GOESA

A Tabela 4.1 identifica os componentes da GOESA que são de uso geral, e aqueles que precisam ser customizados para uma dada aplicação.

Tabela 4.1 - Reusabilidade de componentes da GOESA

Componente da arquitetura	Reusabilidade
Internal State Inference Service	De uso geral, compatível com qualquer missão que adote o ECSS PUS – inclusive as da PMM.
Modelo Embarcado	Específico para uma dada aplicação, descrito em uma linguagem de modelagem de uso geral. Os mecanismos embarcados de gerenciamento e consulta ao modelo fazem parte do ISIS e são, portanto, de uso geral.
Gerenciador de Objetivos	Específico para uma dada aplicação.
Determinador Embarcado de Objetivos	Específico para uma dada aplicação, quando presente.
Planejador	De uso geral, mas pode ser customizado ou substituído por outro específico, de melhor desempenho.

Frisa-se que mesmo os componentes específicos – ou que podem ser específicos – são

criados sobre interfaces providas pelo ISIS, que é de uso geral. Assim, poderiam ser adaptados a partir de componentes desenvolvidos para missões anteriores.

5. REPRESENTAÇÃO DE OBJETIVOS E MODELAGEM DE DOMÍNIOS PARA A GOESA

Para tratar os objetivos no segmento espacial, é necessário estabelecer como representá-los de forma a permitir sua manipulação por *software*. Conforme demonstrado no Capítulo 3, projetos de planejadores embarcados na área espacial adotam para esse fim técnicas de Problemas de Satisfação de Restrições (do inglês *Constraint Satisfaction Problems*, CSP), em especial o ramo das Redes de Tarefas Hierárquicas (do inglês *Hierarchical Task Network,* HTN).

Este Capítulo descreve os conceitos básicos para essas técnicas, a forma como os objetivos são representados em cada caso, e trata de como as soluções adotadas na área espacial levam à separação de sistemas planejadores em módulos de planejamento e de escalonamento.

Em seguida são apresentadas as formas adotadas na GOESA para a descrição de modelos e representação de objetivos, e demonstra-se como elas promovem a unificação de tais módulos.

5.1. Representação e Manipulação de Objetivos a Bordo

5.1.1. Objetivos em Problemas de Satisfação de Restrições

Os Problemas de Satisfação de Restrições (CSP) surgiram na década de 1970 no campo da Inteligência Artificial (IA) e são objeto de estudos tanto de IA como de Pesquisa Operacional.

Um CSP é definido como uma tripla ⟨X, D, C⟩ onde X é um conjunto de variáveis, D são domínios de valores atribuíveis às variáveis e C é um conjunto de restrições. Cada restrição é um par ⟨t, R⟩, onde t é uma enupla (sequência ordenada de ‘n’ elementos) de variáveis e R é uma relação ‘n-qualquer’ em D.

Uma avaliação das variáveis é uma função do conjunto de variáveis para o domínio de valores, $v : X \rightarrow D$. Uma avaliação 'v' satisfaz a restrição $\langle (x_1, \ldots, x_n), R \rangle$ se $(v(x_1), \ldots, v(x_n) \in R)$. Uma solução é uma avaliação que satisfaz todas as restrições.

Em um domínio modelado na forma de CSP os estados são definidos pelos valores de um conjunto de variáveis, e os objetivos consistem de um conjunto de restrições que tais valores devem obedecer (RUSSELL e NORWIG, 2004).

Cada variável v_i em um CSP possui um domínio D_i de valores que pode assumir. Uma restrição R_i limita as combinações possíveis de valores atribuídos às variáveis, e pode estabelecer uma relação entre variáveis, ou um objetivo.

Uma solução para um CSP define valores para todas as variáveis de forma que todas as restrições impostas sejam satisfeitas.

5.1.2. Exemplo de um CSP

Segue um exemplo simples de CSP para um trajeto a ser percorrido por um automóvel:

Dados os domínios

D_1 = {40, ..., 100}

D_2 = {60, ..., 180}

D_3 = {0, ... }

D_4 = {20, ..., 25}

D_5 = {50, ..., 200}

As variáveis

velocidade $\in D_1$

distância percorrida $\in D_2$

tempo de trajeto $\in D_3$

origem $\in D_4$

destino $\in D_5$

E as restrições (que estabelecem relações)

destino = origem + distância percorrida

tempo de trajeto = distância percorrida / velocidade

Uma solução arbitrária para esse problema, que respeita todos os domínios e restrições impostas, seria:

velocidade = 50

distância percorrida = 150

tempo de trajeto = 3

origem = 25

destino = 175

Objetivos podem ser adicionados ao problema restringindo mais os valores possíveis da solução, como *"percorrer a distância de 100 quilômetros a uma velocidade não maior que 80 km/h"*.

A partir desses conceitos básicos, diversas aplicações podem ser criadas. Por exemplo, múltiplos objetivos podem ser adicionados, e é possível estabelecer que alguns sejam obrigatoriamente atendidos, enquanto outros sejam 'desejáveis', ou secundários.

5.1.3. Operações e Planejamento para Alcançar Objetivos

Em sistemas reais não é possível modificar diretamente os estados como se fossem valores de variáveis. Para isso é necessário utilizar operações.

Cada operação modifica o estado de uma ou mais variáveis. Considerando o exemplo anterior, 'acelerar' e 'frear' seriam operações que modificam diretamente o estado 'velocidade' e indiretamente (através da relação entre variáveis) o estado 'tempo de trajeto'.

Em um CSP que leve tal limitação do mundo real em conta, encontrar a solução para o problema, ou seja, alcançar os objetivos, passa a ser encontrar uma sequência de operações que leve o sistema do estado inicial para um estado final onde todas as restrições sejam atendidas. Essa sequência de operações é um plano, e ao processo de

criação do plano dá-se o nome de planejamento.

Todos os sistemas estudados e apresentados no Capítulo 3, à exceção do RASSO, representam operações através do conceito de tarefas, descrito a seguir.

5.1.4. Objetivos em Redes de Tarefas Hierárquicas

Uma forma de especialização de CSP são as Redes de Tarefas Hierárquicas (HTN). Técnicas de planejamento baseado em HTN tentam chegar a um objetivo a partir da decomposição de tarefas de alto nível até chegarem a tarefas primitivas.

Uma tarefa – qualquer que seja seu nível – representa uma atividade realizada pelo sistema modelado, e possui uma duração temporal definida por seus momentos de início e fim.

Em uma HTN, cada tarefa pode ser decomposta em um conjunto de outras tarefas previamente especificadas (SMITH et al., 2000, p. 53). Há formas alternativas de decompor tarefas, e restrições que estabelecem regras sobre como tarefas de mais baixo nível podem ou não se relacionar. A Figura 5.1 traz um exemplo simples desse tipo de decomposição hierárquica de tarefas em uma HTN.

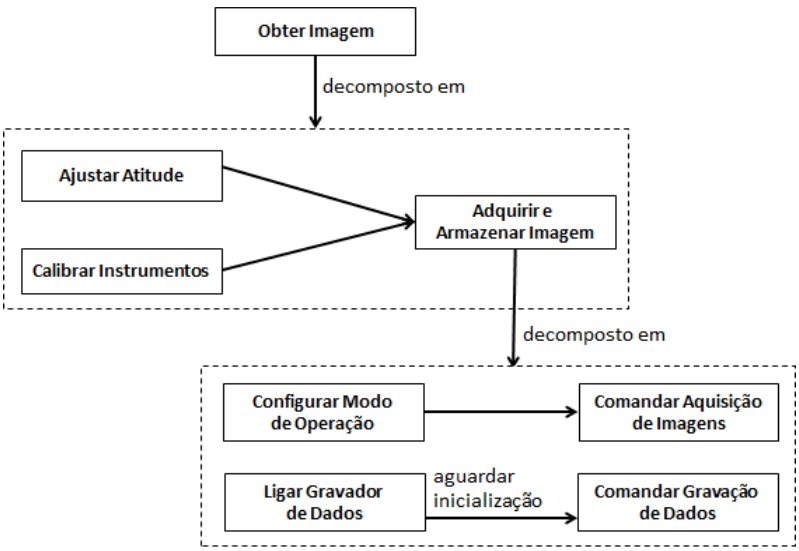


Figura 5.1 - Exemplo de decomposição hierárquica de tarefas em uma HTN

O objetivo dentro de uma HTN é a realização da tarefa de mais alto nível, ao invés de um conjunto de restrições a serem respeitadas. Na Figura 5.1, por exemplo, o objetivo é realizar a tarefa 'Obter Imagem'.

As restrições são impostas não sobre valores de variáveis representando estados, mas sobre como as tarefas podem se relacionar. Assim, definem-se regras de composição, precedência e intervalos entre tarefas.

De forma análoga ao que ocorre com um CSP simples, cabe ao sistema planejador encontrar o conjunto de relações entre tarefas que respeite todas as restrições impostas. Esse conjunto descreve a composição hierárquica e a sequência das atividades em um plano.

De acordo com Smith et al. (2000, p. 53), "virtualmente todos os sistemas de planejamento que foram desenvolvidos para aplicações práticas usam técnicas de HTN". Embora a afirmação não seja recente, ela certamente se aplica aos sistemas descritos no Capítulo 3.

5.1.5. Exemplo da Transformação de Objetivos em Restrições e Tarefas

Para ilustrar como CSP e HTN podem ser aplicados a missões espaciais que adotem o paradigma de operação baseada em objetivos, a Tabela 5.1 traz um exemplo das transformações que um objetivo a ser enviado para um satélite de sensoriamento remoto deveria sofrer. O exemplo é o mesmo apresentado na seção 2.1.

Tabela 5.1 - Transformações sofridas por um objetivo até ser tratável por CSP e HTN

Estabelecido por	Objetivo	Transformações realizadas
Usuário final da missão	"Quero uma imagem da Baía do Almirantado, na Antártica".	-
Operador do satélite	Obter uma imagem entre a (latitude Y_1, longitude X_1) e (latitude Y_2, longitude X_2).	Localização convertida em referências geográficas.

continua

Tabela 5.1 - Conclusão

Estabelecido por	Objetivo	Transformações realizadas
Sistema de solo	Obter uma imagem entre os momentos A e B.	Referências geográficas convertidas em referências temporais (período em que o satélite passa sobre a área a imagear), em função de dados orbitais. Passível de tratamento por HTN.
Software de voo	Ligar e manter a câmera e o gravador de dados ligados entre os momentos A e B.	Objetivo anterior convertido em restrições aos estados de equipamentos, relacionados a informações temporais. Passível de tratamento por CSP.

O objetivo como estabelecido/transformado pelo sistema de solo é colocado como uma atividade a realizar, 'Obter Imagem', com momentos de início e fim. Tal objetivo é diretamente representável como uma tarefa de mais alto nível em um HTN.

Uma transformação adicional feita pelo *software* de voo descreve a mesma tarefa em termos dos estados a serem mantidos para os equipamentos que a realizam. Modelados como variáveis, os estados-objetivo tornam-se restrições aplicáveis a um CSP.

Qualquer que seja a forma escolhida para o objetivo, caberia ao *software* embarcado no satélite buscar e encontrar um plano que o atenda. As técnicas apresentadas anteriormente neste Capítulo, entretanto, tratam de mudanças de estados e relações entre tarefas, mas não lidam com a componente temporal dos objetivos.

5.1.6. Sistemas de Escalonamento para Lidar com o Tempo

A representação de objetivos tanto como restrições a um CSP ou na forma de tarefas em uma HTN permite que se modele um grande número de problemas. Na maioria dos sistemas reais, entretanto, ambas precisam ser complementadas por informações temporais que estabeleçam quando os estados do domínio devem atingir os valores desejados pelo usuário, ou em que períodos dadas tarefas devem ser realizadas. Isso configura um problema de escalonamento.

Um problema de escalonamento envolve a atribuição de recursos, limitados em quantidade, para tarefas distintas durante determinado período de tempo, de forma a otimizar um ou mais objetivos.

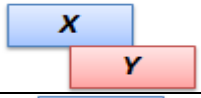
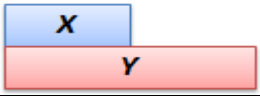
Os estudos sobre técnicas de escalonamento se originaram na área de Pesquisa Operacional e vêm sendo incorporados a projetos em IA, notadamente na última década e meia (KUCINSKIS, 2007).

A principal forma utilizada para a manipulação de informações temporais em sistemas de escalonamento é representando-as como intervalos de tempo.

A representação de tempo como intervalos foi proposta originalmente por Allen (1983). Nela, ações e eventos passam a ocorrer dentro de períodos de tempo que possuem relacionamentos/restrições temporais entre si. Allen introduziu um conjunto de sete relações básicas possíveis entre pares de intervalos.

A esse conjunto de relações, listadas na Tabela 5.2, deu-se o nome de 'Álgebra de Intervalos de Allen'.

Tabela 5.2 - Relações temporais entre pares de intervalos propostas por Allen

Relação	Exemplo
X *before* Y	
X *equal* Y	
X *meets* Y	
X *overlaps* Y	
X *during* Y	
X *starts* Y	
X *finishes* Y	

Uma outra forma de representação do tempo foi proposta por Vilain e Kautz (1986), e é baseada na ideia de ocorrências atômicas, sem duração, chamadas 'pontos'.

Um ponto pode ser o início ou fim de algum evento, assim como um momento qualquer no tempo. As relações temporais são expressas como comparações de números inteiros através dos operadores >, <, ≥, ≤, = e ≠.

5.1.7. Planejamento e Escalonamento em Cascata

Nota-se uma prevalência, entre os sistemas planejadores descritos no Capítulo 3, da separação entre a resolução das restrições impostas aos estados, e da resolução das restrições temporais e de recursos impostas a tarefas.

Usualmente executa-se um planejador baseado em HTN para determinar uma sequência de tarefas que leve ao atendimento de todas as restrições de estados impostas ao problema.

Com uma sequência definida, um escalonador que implementa álgebra de Allen é chamado para resolver um conjunto de restrições temporais e de consumo de recursos que se aplicam às tarefas. Se o escalonador não conseguir cumprir seu papel, o planejador é solicitado a gerar uma sequência alternativa de tarefas.

Sobre essa separação, um estudo contratado pelo LAAS e ONERA para definir requisitos de missões autônomas futuras afirma que "é muito ineficiente criar planos que posteriormente se mostram inviáveis devido a questões de tempo e recursos" (POLLE, 2002, p. 44).

Apesar disso, essa composição particular de técnicas – HTN e álgebra de Allen – é adotada pelos projetistas de sistemas de planejamento devido à compatibilidade entre os conceitos de tarefas de HTN e intervalos de Allen, sendo ambos descritos como um par de momentos no tempo.

Um efeito adverso da modelagem de um domínio como uma rede de tarefas é que há

um alto nível de abstração. Ações momentâneas que afetam o estado de variáveis devem ser convertidas em tarefas com duração fixa. Deixa-se de representar diretamente o estado, mesclando-se assim a operação e seus efeitos. Embora um alto nível de abstração seja desejável para a operação dos satélites, o mesmo não é verdade para os modelos usados nessa operação; idealmente um modelo deveria ser o mais próximo possível de uma descrição declarativa do sistema real.

O modelo em HTN distancia a descrição do domínio do sistema real que ele se propõe a representar. Smith et al. (2000, p. 54), que participaram da criação do primeiro planejador embarcado no segmento espacial, o RAX, assim como do projeto IDEA, chamam a atenção para o fato de que "diversos pesquisadores estão insatisfeitos com o planejamento baseado em HTN porque ele é mais próximo de 'programar' uma aplicação em particular do que prover uma descrição declarativa das ações disponíveis e usar técnicas gerais para o planejamento".

O presente trabalho evita a adoção de HTN na representação do domínio como forma de promover a unificação entre planejamento e escalonamento. O modelo, tratado como um CSP que não releva a componente temporal do problema, se mantém simples e próximo da 'descrição declarativa das ações disponíveis' desejada.

5.2. Conceitos Básicos para a Representação do Conhecimento na GOESA

A representação do conhecimento na GOESA é baseada na modelagem de domínios como CSPs.

A evolução dos estados das variáveis do problema é modelada através do conceito de 'linha do tempo'. As operações são instantâneas, e os objetivos são impostos sobre os estados no tempo, como restrições binárias em um CSP. Mesclam-se assim os conceitos de planejamento e escalonamento.

O restante deste Capítulo apresenta tais conceitos em maiores detalhes.

5.2.1. Representação da Evolução de Estados no Tempo

Em um CSP que possua informações temporais, uma variável passa a assumir não um, mas um conjunto de valores no tempo, todos eles respeitando o mesmo domínio de valores possíveis.

Diversos trabalhos (como alguns dos apresentados no Capítulo 3 e Verfaillie et al., 2008) dão a esse tipo de elemento o nome de 'linha do tempo' que será referenciado nesta Tese pelo termo em inglês '*timeline*'.

A GOESA adota o conceito de *timelines* para a representação da evolução dos valores das variáveis (estados) no sistema modelado.

Ingham et al. (2005, p. 511) descrevem *timelines* como "repositórios conceituais para o conhecimento do estado", sendo uma "representação da história passada e futura de um sistema". A Figura 5.2 ilustra o conceito de *timelines* usado tanto para aferir a acurácia de um modelo, comparando-as com o histórico de valores passados, como para inferir os estados futuros de um sistema modelado.

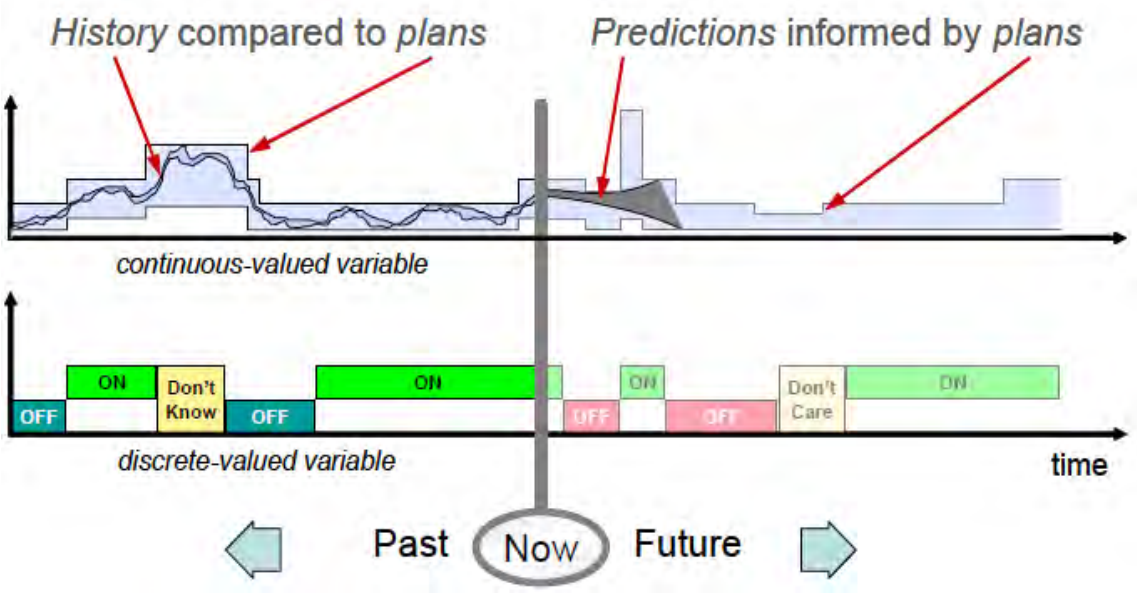


Figura 5.2 - Descrição da dinâmica de timelines contra os valores reais

Fonte: Ingham et al. (2005, p. 511).

Timelines podem representar tanto valores discretos (*timeline* inferior na figura), como valores contínuos discretizados (*timeline* superior).

As *timelines* são os elementos básicos de modelagem de domínios para a GOESA, sobre os quais todos os demais foram criados. Os estados das *timelines* são modificados pela aplicação de operações sobre elas. O próximo passo, portanto, é definir como representar as operações.

5.2.2. Operações Instantâneas em Detrimento de Tarefas com Duração Fixa

Os equipamentos de um satélite são operados por comandos. Os comandos são de execução instantânea, e seus efeitos passam a ser notados a partir do momento em que ocorrem.

A modelagem de operações de planejamento como tarefas com duração fixa, que demandam o consumo de uma quantidade também fixa de recursos, constitui uma abstração do sistema real onde a relação entre comandos e tarefas não é direta; cada tarefa precisa ser mapeada para, no mínimo, um par de comandos.

Ao se criar uma rede de tarefas hierárquicas, o nível de abstração aumenta e torna-se mais difícil a identificação entre o sistema real e seu modelo. Por esse motivo a GOESA adota o conceito de operações sem duração, similar às ocorrências atômicas, ou 'pontos' no tempo, de Vilain e Kautz.

Na GOESA, uma operação é diretamente relacionada a um comando. As operações modificam os estados de *timelines*, que se mantêm até que outra operação seja aplicada sobre elas. Não há restrições temporais entre as operações, e sim entre os estados de *timelines*.

Há, portanto, um relacionamento temporal indireto entre operações, o que é mais próximo do que ocorre na operação real de um satélite. Nela, a preocupação principal é com os estados, sendo os momentos de execução dos comandos uma consequência.

Recursos são modelados como um tipo especial de *timeline*. Uma operação pode alocar uma quantidade fixa de recursos, ou iniciar o consumo de recursos em taxas (unidades consumidas por tempo). Operações subsequentes podem cessar o consumo do recurso, ou iniciar sua liberação também a uma dada taxa. Essa forma de consumir recursos mapeia mais fielmente o comportamento do sistema real.

5.2.3. Objetivos para Estados no Tempo como Restrições Binárias

Uma propriedade de um CSP é a aridade de suas restrições. A aridade refere-se à cardinalidade, ou ao tamanho, do escopo das restrições. Uma restrição unária é imposta sobre uma única variável; já uma restrição binária é imposta sobre um par de variáveis (DECHTER, 2003), criando assim uma relação entre elas.

Restrições binárias são utilizadas em sistemas planejadores com HTN para estabelecer as relações temporais entre tarefas. Da mesma forma, são usadas por Vilain e Kautz para definir as relações entre seus pontos no tempo.

Na GOESA, o tempo é representado como uma variável e está sempre associado a uma *timeline* através de uma restrição binária. Seguem exemplos de como isso se aplica aos objetivos.

Para um objetivo *"a câmera deve estar ligada no momento 1000"*, câmera e momento são variáveis, e o CSP é composto como segue:

Dados os domínios

$D_{câmera}$ = {ligada, desligada}

D_t = {0, ...}

E as variáveis

câmera $\in D_{câmera}$

momento $\in D_t$

A restrição que estabelece o objetivo como uma relação entre estado e tempo é

$R_{câmera,\ momento}$ = (ligada, 1000)

A Figura 5.3 ilustra esse objetivo representado como um CSP binário na GOESA.

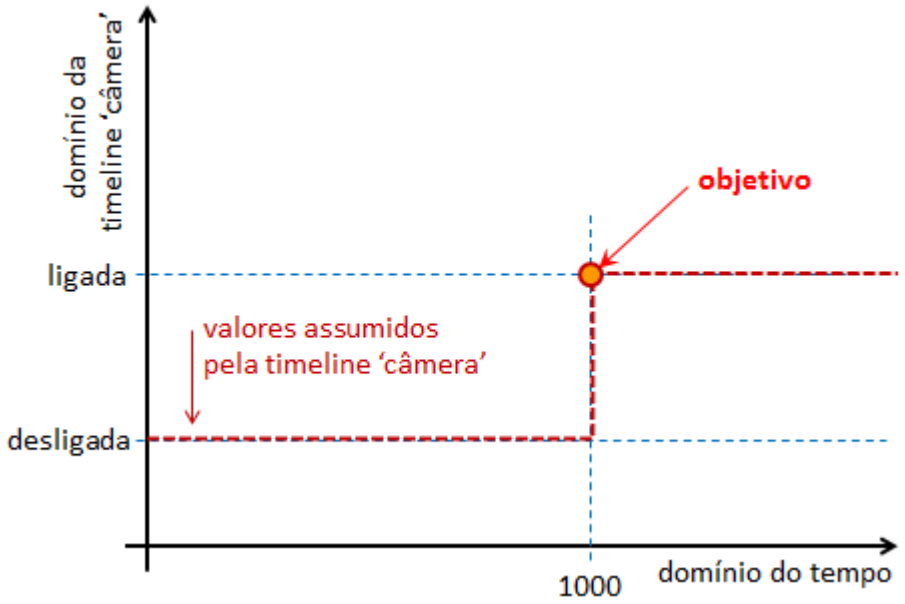


Figura 5.3 - Representação de um objetivo como um CSP binário na GOESA

Séries de valores e períodos de tempo são igualmente representáveis. A partir de um objetivo *"a temperatura deve estar entre 30 e 40 graus durante o período entre 3000 e 3500"*, compõe-se um CSP:

Dados os domínios

$D_{temperatura}$ = {10, ..., 70}

D_t = {0, ...}

E as variáveis

temperatura $\in D_{temperatura}$

período $\in D_t$

A restrição que estabelece o objetivo é

$R_{temperatura,\ período}$ = ({30, ..., 40}, {3000, ..., 3500})

A Figura 5.4 ilustra o objetivo representado como um CSP binário na GOESA.

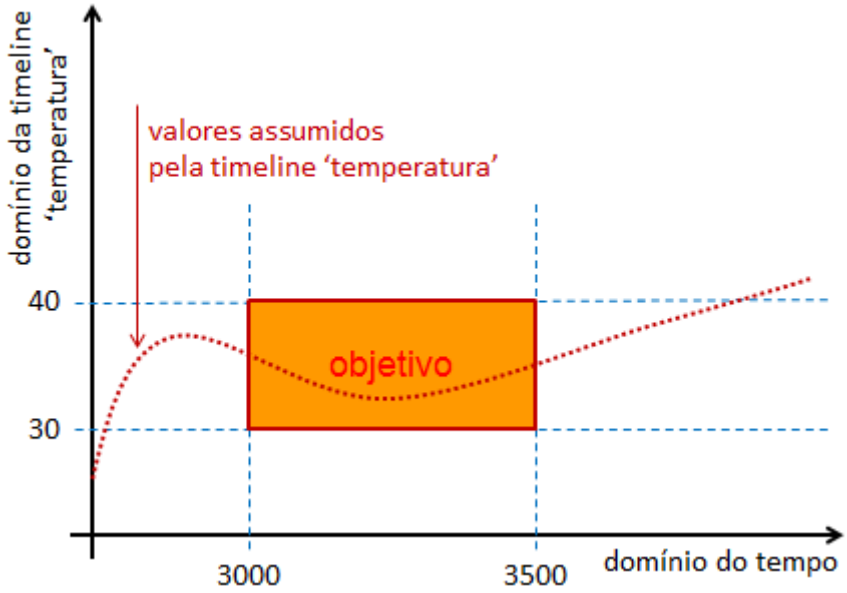


Figura 5.4 - Objetivo estabelecido sobre séries de valores em um período de tempo

Com essa forma de relacionamento entre estados e tempo é possível descrever um comportamento complexo através de conjuntos de restrições binárias que imponham ao CSP estados desejados em períodos específicos, conforme o exemplo da Figura 5.5.

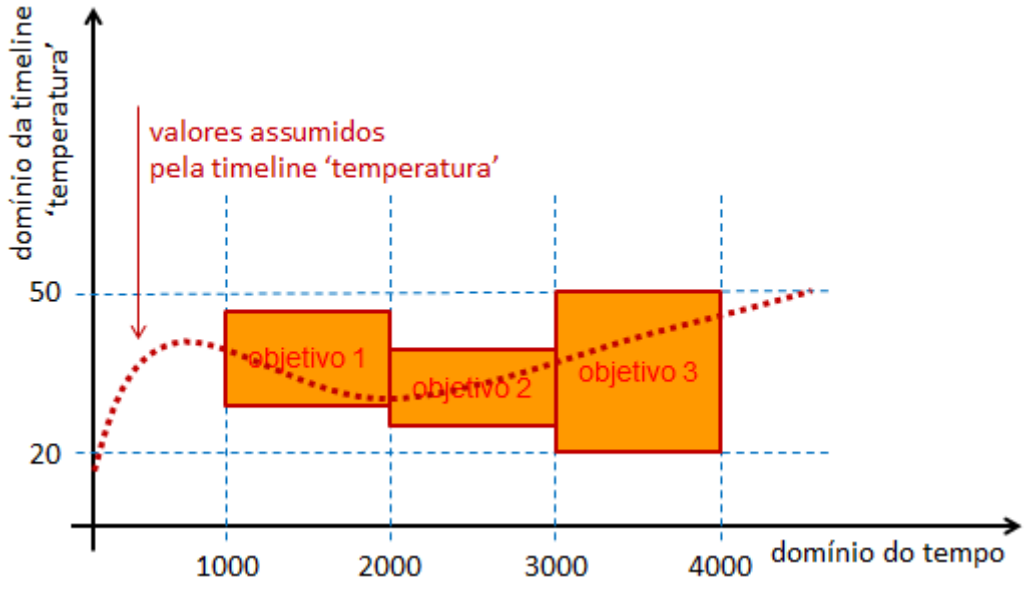


Figura 5.5 - Comportamento complexo definido por múltiplos objetivos

São sobre esses conceitos básicos de modelagem que se descrevem os domínios para a criação de modelos para o componente ISIS da GOESA.

5.3. Descrição de Domínios para a Modelagem no ISIS

Na GOESA, o conhecimento sobre o domínio é mantido pelo modelo que o ISIS contém. O modelo é composto por duas descrições complementares: uma Descrição Estrutural e uma Descrição Comportamental.

A Descrição Estrutural lista os elementos do domínio relevantes a uma dada aplicação, e as *timelines* e recursos que os elementos possuem. A Descrição Comportamental contém as operações que podem modificar o estado das *timelines* e recursos, e quais são seus efeitos sobre eles.

Ambas as descrições são unidas e, através de um analisador (que será tratado nesta Tese pelo termo em inglês, '*parser*'), são transformadas em código-fonte C++, executável no COMAV. Esse código-fonte, quando compilado e *linkado* com uma biblioteca de funções do ISIS, torna-se o modelo a ser embarcado no satélite.

5.3.1. A Descrição Estrutural

A Descrição Estrutural do modelo é mantida em um único arquivo no formato *eXtensible Markup Language* (XML), chamado 'structure.isis'. Esse arquivo especifica:

- Os elementos que compõem o modelo;
- As *timelines* e recursos pertencentes a cada um dos elementos;
- Os consumidores dos recursos;
- Os domínios de valores para *timelines*;
- Os identificadores (IDs) dos dados coletados de equipamentos (chamados pelo ECSS PUS de 'parâmetros de *housekeeping*') associados a cada *timeline* e recurso, para a Interface com o *Software* de Supervisão de Bordo do ISIS.

Os elementos são componentes estruturais utilizados como forma de organizar as

informações sobre o domínio a ser modelado. Exemplos de elementos seriam equipamentos e subsistemas do satélite. Cabe ressaltar que os elementos não são necessários para o CSP, uma vez que a busca pela solução trabalha diretamente sobre as *timelines* e recursos – mas eles são importantes para a inteligibilidade do modelo. O ISIS impõe que quase todos os demais componentes do modelo estejam associados a um dado elemento.

As *timelines* são as unidades de dados básicas do modelo. Elas representam a evolução dos valores de variáveis (estados) no tempo. Cada *timeline* é associada a um parâmetro de *housekeeping* disponibilizado pelo *software* de voo do COMAV.

Os recursos constituem um tipo especial de *timeline* e representam os componentes consumíveis do domínio a ser modelado, como energia, memória de massa e combustível para os propulsores, por exemplo.

A Descrição Estrutural estabelece a quantidade disponível do recurso e quais são os elementos que o consomem. Também deve ser estabelecida a forma de consumo: se é consumido em quantidades absolutas – como por exemplo potência elétrica, em Watts – ou por taxas – como memória, consumida em volume de dados por tempo (e.g., Megabytes por segundo).

Os tipos de dados aceitos pelo ISIS para *timelines* e recursos são *booleanos*, inteiros com e sem sinal de 8, 16 e 32 bits, e enumerações. A especificação das enumerações também deve constar no arquivo structure.isis.

Há a possibilidade de estabelecer restrições condicionais a um dado recurso, e essa informação também consta da Descrição Estrutural. Por exemplo, a quantidade de energia disponível a equipamentos pode ser menor que o total enquanto o satélite estiver em eclipse, período no qual os painéis solares não fornecem energia.

Finalmente, um conjunto de IDs de parâmetros de *housekeeping* deve ser fornecido na Descrição Estrutural para que o ISIS possa obter do *software* de voo do COMAV o

consumo de cada recurso por cada consumidor. Segue um exemplo de Descrição Estrutural.

```
<structural_description>
<domains>
    <domain name = "CameraModes">
        <value>camera_power_off</value>
        <value>camera_stand_by</value>
        <value>camera_imaging</value>
    </domain>
    <domain name = "RecorderModes">
        <value>recorder_power_off</value>
        <value>recorder_starting_up</value>
        <value>recorder_stand_by</value>
        <value>recorder_record</value>
        <value>recorder_playback</value>
        <value>recorder_erase</value>
    </domain>
    <domain name = "SwitchState">
        <value>real_time</value>
        <value>playback</value>
    </domain>
</domains>
<elements>
    <element name = "Camera">
        <timeline name = "Mode" type="CameraModes" hkparameterid="9"/>
    </element>
    <element name = "Recorder">
        <timeline name="Mode" type="RecorderModes" hkparameterid="3"/>
        <timeline name="Ready" type = "bool" kparameterid = "41"/>
        <timeline name="Switch" type="SwitchState" hkparameterid="7"/>
        <resource name="Memory" type = "uint32" quantity = "160000"
                  consumptionperrate = "true">
            <consumer name = "Recorder"
                      initialtotalconsumptionhkparameterid = "22"
                      initialrateconsumptionhkparameterid = "23"/>
        </resource>
    </element>
    <element name = "PowerSubsystem">
        <resource name = "Power" type = "uint16" quantity = "350"
                  consumptionperrate = "false">
            <consumer name = "Camera"
                      initialtotalconsumptionhkparameterid = "24"/>
            <consumer name = "Recorder"
                      initialtotalconsumptionhkparameterid = "25"/>
        </resource>
    </element>
</elements>
</structural_description>
```

5.3.2. A Descrição Comportamental

5.3.2.1. Tipos de Operações

Há dois tipos de operações no ISIS que modificam os estados de *timelines* e recursos: as ações e os eventos.

Uma ação é a descrição dos efeitos sobre *timelines* e recursos de um comando executado no segmento espacial. Cada ação é descrita em três blocos: precondições,

efeitos e identificação do comando correspondente.

As precondições são opcionais e informam quais estados devem ser verdadeiros para que a ação possa ser executada. Não deve ser possível, por exemplo, comandar a gravação de dados se o gravador não estiver ligado. Operadores lógicos '*and*' e '*or*' são aceitos ao estabelecer precondições. Uma dada ação pode conter múltiplas precondições.

Os efeitos descrevem de que forma a execução da ação afeta as *timelines* e recursos.

A identificação do comando serve para a Interface com o *Software* de Supervisão de Bordo do ISIS, e é utilizada com dois fins: extrair do plano de operações corrente quais comandos afetam o domínio modelado, e enviar ao *Command Schedule* do COMAV, após o término do planejamento, os comandos correspondentes às ações do novo plano.

Foi criada como parte da GOESA uma linguagem de modelagem para a Descrição Comportamental, chamada *ISIS modeling language* (ISISml). A ISISml possui sintaxe similar à da linguagem de programação C++, como pode-se verificar no exemplo de descrição de ação que segue.

```
Action StartRecording
{
    Preconditions
    {
        Recorder.Mode = recorder_stand_by;
        Recorder.Ready = true;
        Recorder.Switch = real_time;
    }

    Effects
    {
        Recorder.Mode = recorder_record;
        Recorder.Consume (Recorder.Memory, 128 per_second);
        Recorder.Consume (PowerSubsystem.Power, 28);
    }

    ServiceRequest // identificacao do comando correspondente
    {
        ServiceType = 134;
        ServiceSubtype = 8;
    }
}
```

Note-se que o exemplo se resume a uma descrição declarativa da ação disponível,

exatamente como colocado por Smith et al. (2000) ao relatar o que se busca da união entre planejamento e escalonamento.

O segundo tipo de operação do ISIS, os eventos, descrevem os efeitos de ocorrências que afetam os estados do modelo, mas não são associadas diretamente a comandos. Os eventos podem ser de dois tipos: exógenos e endógenos.

Os eventos exógenos são ocorrências fora do controle do *software* de voo, como a entrada e saída de eclipse e o início e fim de passagens sobre estações terrenas.

Já os eventos endógenos descrevem efeitos indiretos de comandos. Ao comandar que se ligue um equipamento que possua certo tempo de inicialização, por exemplo, o momento em que tal equipamento se torna disponível para uso pode ser marcado por um evento endógeno. Outro exemplo é o estouro no consumo de memória: se um plano possui um comando para gravar, mas nenhum para interromper a gravação, um evento endógeno pode marcar o momento em que não houver mais memória disponível, o que automaticamente interrompe a gravação.

Segue um exemplo de descrição de um evento endógeno em ISISml.

```
// apos ligado, o Recorder fica 15 seg em auto-teste e formatacao
EndogenousEvent RecorderReady
{
    Raises15secondsAfter
    {
        Recorder.Mode = recorder_starting_up;
    }

    Effects
    {
        Recorder.Mode = recorder_stand_by;
        Recorder.Ready = true;
    }
}
```

Não se especificam informações temporais para as operações na modelagem; elas serão estabelecidas pelo ISIS em função dos objetivos recebidos, do estado e do plano de operações corrente do satélite.

Por serem traduzíveis em código C++, tanto as ações como os eventos aceitam estruturas condicionais, de seleção de caso, laços de repetição e chamadas a funções

da biblioteca matemática do C/C++.

Todas as ações devem ser associadas a algum elemento especificado na Descrição Estrutural. Para cada elemento há um arquivo 'elemento_actions.isis' com a descrição das ações que lhe pertencem. Já a descrição de todos os eventos modelados fica em um único arquivo, 'events.isis'.

5.3.2.2. Consumo de Recursos

Uma das características que mais diferencia a modelagem no ISIS daquela de outros sistemas de planejamento e escalonamento é a forma de consumo de recursos. Ao contrário de outros sistemas, nos quais o consumo de recursos é descrito em quantidades fixas e por períodos também fixos e vinculados a tarefas, o ISIS registra modificações no perfil de consumo de recursos por um dado consumidor.

Isso significa que uma operação pode indicar o início ou o término do consumo de um recurso, ou uma mudança na taxa em que esse recurso é consumido ou liberado. É possível informar, por exemplo, que a ação '*Record*' faça com que o gravador de dados passe a consumir 2 Megabytes de memória por segundo. Esse consumo seguirá até que uma ação '*Stop Recording*' subsequente seja executada, ou mesmo uma nova ação '*Record*' que especifique outra taxa de consumo.

Tal representação do consumo de recursos é mais próxima da operação real de satélites do que a de sistemas de planejamento e escalonamento baseados em HTN.

5.3.3. Compilação do Modelo Embarcado

Para a criação do modelo embarcado, os arquivos que contêm as Descrições Estrutural e Comportamental precisam passar por um processo de análise ('*parsing*') e compilação, ilustrado na Figura 5.6.

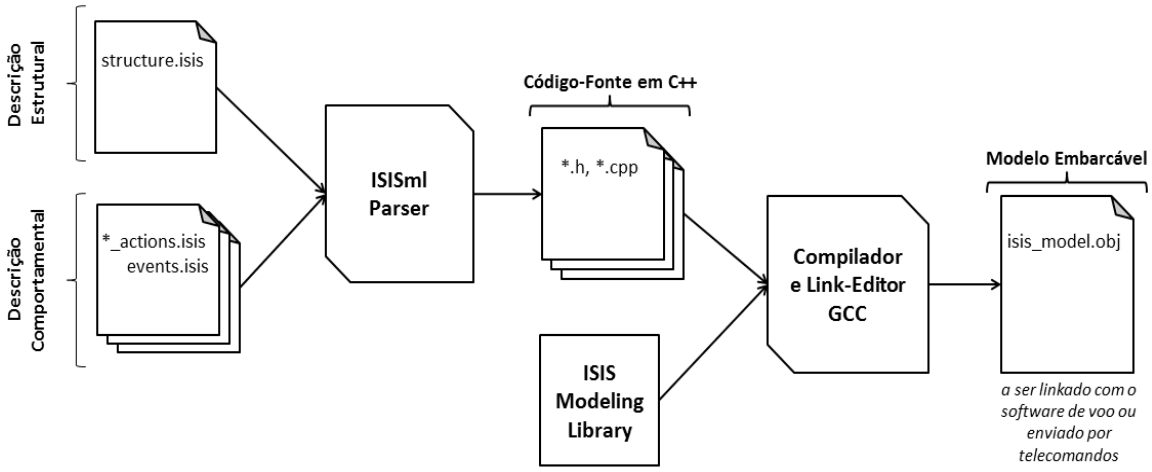


Figura 5.6 - O processo de compilação de modelos para o ISIS

Todos os arquivos que descrevem o domínio devem ser submetidos a um *parser* da ISISml. Cabe a esse *parser* transformar as descrições em código-fonte em C++, compatível com o COMAV.

O código-fonte é então compilado por um compilador cruzado GCC (o mesmo usado pelo COMAV) e *linkado* com uma biblioteca de modelagem do ISIS, que contém todas as interfaces, estruturas de dados e afins, necessárias para embarcar o modelo no satélite.

O resultado da compilação é um arquivo de extensão '.obj', que pode ser *linkado* ao *software* de voo ou enviado por telecomandos ao ISIS para a substituição de um modelo anterior.

O *parser* da ISISml foi totalmente especificado, mas não foi desenvolvido como parte deste trabalho. Suas transformações foram aplicadas manualmente sobre as descrições dos modelos para a geração do código-fonte em C++ e posterior compilação do modelo embarcado. O resultado desse processo de *parsing* aplicado a um estudo de caso pode ser conferido no Apêndice A.

O próximo Capítulo descreve como o modelo é utilizado pelo ISIS para a inferência de estados a bordo de satélites.

6. O INTERNAL STATE INFERENCE SERVICE

O ISIS é o núcleo da arquitetura GOESA. Ele contém o modelo embarcado e possui mecanismos que operam sobre esse modelo para inferir os estados futuros. Os mecanismos provêm métodos que são usados por um planejador ao tentar alcançar os objetivos que são enviados ao segmento espacial.

Este Capítulo descreve o funcionamento interno do ISIS e seus principais componentes.

6.1. O Metamodelo para o Motor de Inferência de Estados

O Motor de Inferência de Estados é o principal componente do ISIS, sendo o responsável pelo gerenciamento da mudança de estados de *timelines* e recursos em função da ocorrência de ações e eventos. Ele possui um metamodelo no qual são armazenadas todas as informações sobre o domínio descritas no modelo embarcado.

O metamodelo é composto por classes e estruturas de dados para a manipulação dos estados de *timelines* e consumo de recursos, as restrições impostas a eles, e outros dados usados na busca pela solução do CSP.

Os subitens seguintes descrevem os componentes internos do metamodelo e como eles são utilizados para a inferência de estados.

6.1.1. As Conversões Realizadas pelo Parser da ISISml

É o *Parser* da ISISml o responsável por converter os componentes descritos no modelo em código-fonte C++. Tal código-fonte preenche o metamodelo do ISIS para a manipulação das *timelines* e recursos pelo Motor de Inferência de Estados.

A Tabela 6.1 mostra a correspondência entre os componentes do modelo e os

componentes do metamodelo no qual os primeiros são manipulados.

Tabela 6.1 - Conversões realizadas pelo Parser da ISISml para o metamodelo

Componente do modelo	Componente do metamodelo
Elemento	Classe 'estática' (contém apenas métodos e atributos estáticos), não instanciada.
Timeline	Atributo estático da classe do elemento a que pertence. É uma instância da classe '*Timeline*'.
Recurso	Atributo estático da classe do elemento a que pertence. É uma instância da classe '*Resource*'.
Ação	Método estático da classe do elemento a que pertence.
Evento	Função global.
Domínio de valores	Restrições impostas sobre *timelines* e recursos.

Para reduzir o acoplamento entre o metamodelo do ISIS e o modelo que ele contém, o metamodelo trata os valores das *timelines* e recursos, especificados no modelo em uma série de tipos de dados, como inteiros de 32 bits. Os valores de domínios estabelecidos no modelo são colocados no metamodelo como restrições.

6.1.2. Timelines e Recursos no Motor de Inferência de Estados

Tanto as *timelines* como os recursos são transformados em atributos estáticos das classes que representam, no metamodelo, os elementos aos quais pertencem.

Tais atributos são instâncias das classes '*Timeline*' e '*Resource*' respectivamente, que contêm dados e métodos para o gerenciamento e a manipulação desses componentes do modelo.

Todo o histórico de modificações dos valores é armazenado nessas classes, associados a informações sobre relações causais (qual operação modificou o valor, e em que momento). As classes contêm métodos de acesso a qualquer dos valores atribuídos no histórico de uma *timeline* ou recurso.

Essas classes mantêm ainda as restrições impostas pelos objetivos ou pelo próprio modelo (como os domínios de valores associados a cada variável). Há métodos para adicionar e remover restrições, e métodos usados pelo Motor de Inferência de Estados

para consultar se as restrições estão sendo violadas em algum momento.

Finalmente, as classes *Timeline* e *Resource* armazenam os IDs necessários ao ISIS para a leitura dos parâmetros de *housekeeping* do *software* de voo.

6.1.3. A Inicialização Estática do Metamodelo e os Catálogos

Entre os arquivos de código-fonte gerados pelo *Parser* da ISISml, encontra-se um chamado 'ModelInitialization.cpp'. Esse arquivo preenche o metamodelo do ISIS de forma estática, em tempo de compilação, com os dados do modelo de domínio.

É nesse arquivo que todos os atributos estáticos (*timelines* e recursos) são inicializados e todos os IDs usados na Interface com o *Software* de Supervisão de Bordo são registrados. Mas a principal função do arquivo é o preenchimento de catálogos internos dos componentes do modelo do ISIS. Há quatro catálogos: *timelines*, recursos, ações e eventos.

Para cada componente do modelo, o *Parser* da ISISml gera um ID correspondente (IDs do modelo, diferentes daqueles necessários à Interface com o *Software* de Supervisão de Bordo). Esses IDs são utilizados nas consultas e operações realizadas pelos usuários do serviço: o Segmento Solo, Gerenciador de Objetivos e Planejador Embarcado.

Os catálogos mantêm a relação entre os IDs do modelo e os ponteiros internos aos respectivos objetos do metamodelo. Essa relação é utilizada tanto para a interface com os usuários do ISIS, como para o registro interno dos efeitos de ações e eventos.

Ao se preencher os catálogos, são informados inclusive quais ações afetam quais *timelines* e recursos. O *Parser* da ISISml extrai essa informação da Descrição Comportamental, quando da criação do arquivo 'ModelInitialization.cpp'.

6.2. As Ocorrências e Restrições

O principal conceito para a realização da inferência de estados é o de ocorrência. Uma

ocorrência é o registro de um acontecimento no tempo que afeta o estado de *timelines* ou recursos, ou que impõe restrições a eles. Os tipos de ocorrências existentes no ISIS são:

- Início ou fim da sessão de inferência;
- Ação;
- Evento exógeno;
- Evento endógeno;
- Início ou fim de restrição imposta a *timeline* ou recurso.

A ação e os eventos fazem parte do modelo de domínio e já foram descritos anteriormente. O início e fim de restrições dizem respeito aos objetivos estabelecidos pelo usuário.

Já as ocorrências de início e fim de sessão marcam os limites temporais ao Motor de Inferência de Estados. Elas também determinam os limites do domínio de valores possíveis para a representação do tempo como uma variável no CSP.

As ocorrências existem apenas dentro de uma Lista de Ocorrências mantida pelo Motor de Inferência de Estados.

6.2.1. A Lista de Ocorrências e as Relações Causais

Se o ISIS é o núcleo da arquitetura GOESA, e o Motor de Inferência de Estados é o núcleo do ISIS, a Lista de Ocorrências é o núcleo do Motor de Inferência. É nessa lista que são consolidadas as informações do modelo embarcado, os dados obtidos da Interface com o *Software* de Supervisão de Bordo e os objetivos recebidos do usuário.

Cada registro de ocorrência na lista possui uma identificação única dentro de uma sessão de inferência (descrita a seguir neste Capítulo), o momento, e informações

adicionais sobre o quê ocorre. Isso permite ao ISIS trabalhar com relações causais durante a inferência.

As informações adicionais variam de acordo com o tipo de ocorrência, conforme descrito abaixo:

- A ocorrência de início de sessão é associada aos estados iniciais da sessão de inferência, que correspondem aos valores de parâmetros de *housekeeping* obtidos do *software* de voo pela Interface com o *Software* de Supervisão de Bordo;
- Ocorrências do tipo ação ou eventos são associadas aos seus efeitos sobre *timelines* e recursos;
- A ocorrência de fim de sessão é associada aos estados finais inferidos;
- As ocorrências de início e fim de restrições são referências temporais para os objetivos do usuário.

Não há limitações para a existência de mais de uma ocorrência em um mesmo momento.

Uma mesma ação pode se repetir em ocorrências distintas, representando um mesmo comando executado duas ou mais vezes. O mesmo ocorre com eventos. Nesse sentido, ocorrências podem ser vistas como instâncias de ações e eventos.

Toda a informação temporal para a inferência de estados encontra-se na Lista de Ocorrências. Já os efeitos das ocorrências são registrados nas respectivas *timelines* e recursos.

6.2.2. As Restrições e os Registros de suas Violações

O ISIS permite a adição e remoção de restrições impostas sobre *timelines* e recursos, utilizadas para informar ao Motor de Inferência de Estados quais são os objetivos do

usuário.

As restrições são binárias, impondo limitações de valores sobre *timelines* e recursos em um dado intervalo de tempo. Elas são especificadas pelos atributos: valor mínimo, valor máximo, momento inicial e momento final.

Cada restrição adicionada é registrada dentro do metamodelo na *timeline* ou recurso correspondente, e um par de ocorrências marca o início e fim de sua vigência na Lista de Ocorrências.

Quando o Motor de Inferência de Estados detecta que uma restrição está sendo violada, essa informação é registrada na Lista de Ocorrências. Precondições falsas de ações são tratadas da mesma forma.

Se a restrição violada está relacionada a objetivos do usuário, essa informação também é registrada para que o planejador possa tratá-las com maior prioridade.

Além dos registros de violações nas ocorrências, o Motor de Inferência possui contadores totais de violações que podem ser consultados externamente.

6.3. O Motor de Inferência de Estados e as Sessões de Inferência

O Motor de Inferência de Estados percorre a Lista de Ocorrências, registrando seus efeitos e as violações de restrições. Ele é executado dentro de uma sessão de inferência.

A sessão de inferência é o processo iniciado pelo usuário do ISIS no qual se submetem os objetivos a alcançar, e se obtém do *software* de voo o estado atual e o plano de operações corrente.

Ela possui um momento de início e um de término, ambos futuros com relação ao tempo de bordo (*on-board time*, OBT), que representam o período de tempo para o qual devem ser inferidos os estados.

A unidade de tempo do ISIS é o segundo, e o tempo é representado por uma contagem absoluta de 32 bits. O ISIS não impõe época alguma, apenas recebe o OBT e informações temporais do *software* de voo e os manipula sem conversão de formato. Isso o torna potencialmente compatível com qualquer tipo de OBT adotado em uma dada missão.

Na abertura da sessão, ao ler o plano de operações corrente, o ISIS extrai dele apenas os comandos que possuírem ações correspondentes no modelo embarcado. Assume-se que comandos que não estejam descritos no modelo não afetem o domínio modelado.

Com o estado atual e o plano de operações corrente, o Motor de Inferência processa as ocorrências, uma a uma e na ordem em que constam da Lista de Ocorrências, registrando os seus efeitos sobre *timelines* e recursos, as precondições falsas de ações, e as restrições violadas.

Uma vez realizada a inferência, os resultados são disponibilizados ao usuário. Após as consultas necessárias, o usuário pode modificar a Lista de Ocorrências através de uma operação de inferência, o que dispara novamente a execução do Motor de Inferência.

6.3.1. As Operações de Inferência

O ISIS provê três operações de inferência aos seus usuários:

- Adicionar uma ação à Lista de Ocorrências;
- Remover uma ação existente na Lista;
- Mover uma ação no tempo.

O ISIS também permite aos usuários desfazer a última operação, retornando a Lista de Ocorrências ao seu estado imediatamente anterior, através de um método '*Rollback*'.

É possível ainda fazer um backup completo dos estados inferidos no metamodelo e

restaurá-lo posteriormente. Isso dá aos usuários a possibilidade de criar 'pontos de restauração' durante a busca por soluções.

Não é possível operar sobre eventos exógenos; eles apenas ocorrem e seus efeitos devem ser respeitados. Pode-se operar sobre eventos endógenos indiretamente, ao mover ou remover as ações relacionadas a eles.

Cabe destacar que, após a submissão de uma operação de inferência, os estados não são reavaliados a partir do momento de início da sessão, mas sim a partir do momento indicado pela operação de inferência – afinal, o comportamento previsto pelo modelo será modificado somente a partir desse momento.

6.3.2. Consultas e Métodos Disponíveis aos Usuários do ISIS

O ISIS disponibiliza uma série de consultas e métodos aos seus usuários, que podem ser utilizadas para a conversão de objetivos em restrições ou para o processo de planejamento.

Os principais métodos disponíveis seguem listados abaixo.

- Abrir e fechar sessões de inferência de estados;
- Adicionar e remover restrições a *timelines* e recursos;
- Adicionar, remover e mover uma ação no tempo;
- Desfazer a última operação de inferência;
- Fazer um backup completo do metamodelo, com os estados inferidos, Lista de Ocorrências e informações auxiliares;
- Restaurar o backup do metamodelo.

As principais consultas que podem ser realizadas por usuários do ISIS são:

- Obter uma lista das ações catalogadas que afetam uma dada *timeline* ou recurso;
- Obter uma lista das ocorrências que afetam uma dada *timeline* ou recurso, anteriores a um dado momento;
- Obter informações sobre restrições violadas e precondições falsas de uma dada ocorrência na Lista de Ocorrências;
- Obter a primeira ocorrência na Lista de Ocorrências que viole restrições, ou possua precondições falsas;
- Obter os contadores totais de violações de restrições e precondições falsas;
- Obter uma lista com as *timelines* e recursos que possuam alguma restrição violada, ou que estejam envolvidos em precondições avaliadas como falsas;
- Obter uma versão sumária da Lista de Ocorrências.

Os estados das *timelines* e recursos não são consultáveis externamente ao ISIS. Isso em princípio não é necessário, uma vez que é possível consultar quais *timelines* ou recursos possuem violações, quais ocorrências na Lista as afetam, e quais ações podem ser utilizadas para modificar seus valores.

A consulta aos estados poderia ser adicionada ao serviço, caso seja desenvolvido para a GOESA um planejador que faça uso de heurísticas de busca. Esse, entretanto, não é o caso do planejador de referência criado como parte desta Tese, descrito no próximo Capítulo.

7. O PLANEJADOR EMBARCADO LETMEDO

O principal usuário do ISIS é o Planejador Embarcado. Como parte deste trabalho foi criado um planejador de uso geral, não voltado a nenhuma missão específica. A esse planejador foi dado o nome '*LetMeDo*'.

O *LetMeDo* implementa um algoritmo de planejamento simples, mas adequado para execução a bordo de satélites. Além disso, seu comportamento na busca pela solução de um CSP é configurável para melhor se adaptar a uma dada aplicação. A partir dos objetivos recebidos, o *LetMeDo* é capaz tanto de criar planos de operação novos, como de modificar um plano existente.

Esse planejador deve ser visto como uma referência para a arquitetura GOESA. O *LetMeDo* não se propõe a ser um planejador de desempenho otimizado – embora haja a possibilidade de otimização por revisões na lógica e melhorias em seu código, e eventualmente pela adição de heurísticas de busca.

O presente Capítulo descreve o planejador embarcado LetMeDo.

7.1. Técnicas de Busca Local

O algoritmo implementado no *LetMeDo* é da classe de busca local 'gulosa' (*greedy local search*), mais especificamente, do tipo 'subida de encosta' (*hill-climbing*). Ele pode, entretanto, ser configurado para comportar-se como uma busca do tipo aleatória (*randomized local search*). Heurísticas de busca não são utilizadas.

A escolha pela busca local foi baseada na pesquisa realizada por Kucinskis (2007) sobre algoritmos de planejamento adequados para execução a bordo de satélites. Essa é uma técnica geral de planejamento, algo que, de acordo com Smith et al. (2000), é desejável para a unificação do planejamento com o escalonamento.

A busca local começa com todas as variáveis do CSP inicializadas. Algumas das atribuições feitas às variáveis violam restrições impostas ao problema. O algoritmo tenta aproximar iterativamente os valores das variáveis de um objetivo, onde todas as restrições são satisfeitas.

Para isso, algoritmos de busca local tipicamente modificam o valor de uma única variável a cada passo, avaliando se a nova atribuição está mais próxima do objetivo no espaço de busca.

No caso do *LetMeDo*, o algoritmo aplica uma modificação ao plano – representado na Lista de Ocorrências – por vez, avaliando se os efeitos nas *timelines* e recursos levaram o estado geral do domínio modelado mais próximo dos objetivos recebidos do usuário. Essa técnica é similar ao 'reparo iterativo' adotado pelo planejador CASPER no satélite EO-1 da NASA.

A busca local é um método incompleto para encontrar a solução de problemas, ou seja, ela não garante que uma solução seja encontrada. Também não é um método ótimo, uma vez que qualquer solução válida, mesmo que não otimizada, finaliza a busca. Ela é, entretanto, menos custosa em termos computacionais do que métodos completos como a busca construtiva, e frequentemente apresenta melhores resultados (DECHTER, 2003).

7.2. O Algoritmo de Planejamento do LetMeDo

O planejamento no *LetMeDo* pode partir tanto de um plano de operações preexistente como de um plano vazio. Em ambos os casos, todas as *timelines* e recursos são inicializados com os estados obtidos pela Interface com o *Software* de Supervisão de Bordo, na abertura de uma sessão de inferência.

Os objetivos do usuário são aplicados como restrições às *timelines* e recursos, gerando as violações a serem resolvidas.

O loop principal do algoritmo de planejamento consiste da escolha de uma única violação por vez, e da busca por sua solução.

A busca pela solução é feita pela aplicação tentativa de um conjunto de operações de planejamento. Grosso modo, uma operação de planejamento, descrita a seguir neste Capítulo, consiste de uma modificação ao plano.

A cada operação tentada, o *LetMeDo* analisa o estado resultante e atribui uma pontuação que indica o quão mais próximo o estado geral ficou do objetivo. Pontuações negativas são atribuídas a operações que distanciem o estado do objetivo.

Após cada operação de planejamento tentada, o *LetMeDo* devolve os estados inferidos à situação anterior (usando o método '*Rollback*' do ISIS) e tenta a próxima. Uma vez que todas as operações disponíveis tenham sido tentadas, seus resultados são comparados e se escolhe aquela de maior pontuação, aplicando-a definitivamente ao plano contido no ISIS.

O *LetMeDo* retorna então ao seu loop principal, selecionando a próxima violação a resolver. Esse processo se repete até que não haja mais violações, o que encerra o planejamento.

Qualquer operação aplicada pode adicionar novas violações de restrições ou tornar falsas precondições (a sua própria ou a de ações seguintes no plano), que serão tratadas posteriormente pelo loop principal de planejamento.

O *LetMeDo* prioriza a solução das violações das restrições diretamente relacionadas aos objetivos do usuário. Assim que todas essas são resolvidas, ele passa a buscar a solução para as violações do próprio modelo e aquelas adicionadas durante a solução das primeiras.

7.3. As Operações de Planejamento

Uma operação de planejamento é uma tentativa de modificação do plano para levar o

estado inferido do sistema modelado mais próximo ao estado desejado, no qual todos os objetivos são atendidos.

As operações de planejamento são criadas sobre as três operações de inferência disponibilizadas pelo ISIS: adicionar, remover ou mover ações no tempo.

Há 30 operações de planejamento para serem tentadas pelo *LetMeDo*, todas elas desabilitáveis individualmente. É possível assim identificar de forma empírica as operações mais efetivas para uma dada aplicação, para um dado modelo, e desabilitar as demais.

Seguem alguns exemplos das operações de planejamento disponíveis:

- Adicionar ao plano uma ação que opera sobre a *timeline* ou recurso violado, 'n' segundos antes de uma ocorrência que indique uma violação de restrição ou precondição falsa (que serão tratadas como 'ocorrência com violação');
- Adicionar ao plano uma ação que opera sobre a *timeline* ou recurso violado, 'n' segundos depois de uma ocorrência com violação;
- Adicionar ao plano uma ação que opera sobre a *timeline* ou recurso violado, em um momento aleatório entre a ocorrência com violação e aquela que a precede, ou o início da sessão;
- Remover do plano a ocorrência com violação, caso ela seja uma ação;
- Remover do plano uma ação que opere sobre a *timeline* ou recurso violados e ocorra em um momento anterior à ocorrência com violação. Isso é tentado para todas as ações que se enquadrem nessas condições, para cada violação detectada;
- Mover temporalmente no plano a ocorrência com violação, adiantando-a ou atrasando-a 'n' segundos;

- Mover temporalmente no plano a ocorrência com violação, para um momento aleatório entre ela e a ocorrência que a precede, ou o início da sessão.

Testes empíricos realizados durante o desenvolvimento do *LetMeDo* sugerem que para uma dada aplicação, seria suficiente para a solução dos problemas um conjunto não maior que 10 operações de planejamento habilitadas.

7.4. Adição de Perturbações ao Plano

O maior problema dos algoritmos de busca local é a possível presença de 'platôs' e mínimos locais. Platôs são regiões do espaço de busca onde nenhum movimento local (ou nenhuma modificação local no plano) leve a um estado geral mais próximo do objetivo. Mínimos locais são regiões onde todos os movimentos locais distanciam o estado do objetivo.

No *LetMeDo*, tais situações são representadas por pontuação menor ou igual a zero para todas as operações tentadas em um laço do loop de planejamento principal.

Para resolver esse tipo de problema foram criados os algoritmos de busca local aleatória: eles tentam escapar de platôs e mínimos locais realizando movimentos aleatórios.

O *LetMeDo* tenta sair desse tipo de situação adicionando perturbações ao plano. Quando as perturbações são utilizadas, seu algoritmo de planejamento comporta-se de forma similar a um de busca local aleatória.

A perturbação ao plano pode ser:

- A adição de uma ação aleatória, selecionada entre as que operam sobre a *timeline* ou recurso violado, em um momento também aleatório, anterior ao da ocorrência que indica a violação;

- A substituição de uma ação que indica uma violação por outra ação, aleatória.

7.5. Busca pelo Melhor Caminho para Desempate

Caso, ao selecionar a operação de planejamento a aplicar ao plano, o *LetMeDo* detecte que há mais de uma operação com a maior pontuação (situação de 'empate'), ele seleciona aquela que temporalmente está mais próxima da ocorrência que indica a violação que se tenta resolver. Essa é a que potencialmente tem menor impacto negativo sobre o plano.

Entretanto, há a possibilidade de iniciar uma busca sistemática pelo melhor caminho a seguir no espaço de busca a partir desse empate, explorando a aplicação de 'n' outras operações de planejamento a partir de cada uma das ocorrências 'empatadas'.

Após o término da exploração, o *LetMeDo* escolhe o melhor caminho, o aplica, e segue a busca. Esse processo se repetirá a cada empate encontrado durante o planejamento.

A busca pelo melhor caminho para desempate é uma característica configurável do *LetMeDo*. Se por um lado os planos encontrados passem a ser mais próximos do ótimo, por outro isso faz com que os tempos de planejamento se elevem a outras ordens de grandeza.

7.6. Características Configuráveis do LetMeDo

Apesar de não ser um planejador de desempenho otimizado, o *LetMeDo* pode ter seu comportamento na busca por uma solução configurado. Isso tem por objetivo permitir a customização do planejador à solução de problemas diferentes, pertencentes a classes de aplicações ou missões diferentes.

Os principais parâmetros configuráveis são listados a seguir.

- Número máximo de operações a tentar a cada laço do loop principal de planejamento;

- Número máximo de laços de planejamento a executar. Ao ser atingido, o *LetMeDo* indica ao usuário falha na criação ou modificação do plano para atender aos objetivos recebidos;

- Tempo máximo de planejamento. Ao ser atingido, o *LetMeDo* também indica falha ao usuário na busca por uma solução;

- Buscar ou não pelo melhor caminho em situações de empate;

- Número de caminhos alternativos e nível de profundidade a explorar na busca para desempate;

- Adicionar ou não perturbações ao plano quando necessário;

- Pontuações a aplicar ao avaliar os estados resultantes das operações de planejamento tentadas;

- Priorizar ou não a solução de violações 'imediatas', ou seja, violações da ocorrência selecionada para resolução no loop principal;

- Habilitar e desabilitar individualmente as operações de planejamento.

O próximo Capítulo apresenta os estudos de caso aplicados à GOESA nos quais o *LetMeDo* foi utilizado, incluindo dados de desempenho do planejador e do ISIS.

8. ESTUDOS DE CASO PARA A GOESA

Dois estudos de caso de missões do INPE foram selecionados para demonstrar as capacidades da GOESA como ferramenta de viabilização da operação de missões espaciais baseada em objetivos.

O primeiro estudo de caso, para o satélite Amazonia-1, teve como finalidade exercitar (i) os recursos de modelagem providos pela ISISml, (ii) a recepção e tratamento de objetivos a bordo, e (iii) o planejamento realizado sob diferentes cenários.

O segundo consistiu de uma reprodução do estudo de caso originalmente criado para o protótipo RASSO, voltado à missão EQUARS, e teve como meta demonstrar que a GOESA é aplicável a diferentes missões, além de permitir uma comparação de desempenho entre os planejadores.

Este Capítulo detalha o estudo de caso para o satélite Amazonia-1, e sumariza os resultados do estudo de caso para o EQUARS.

8.1. O Ambiente de Operação Utilizado para os Estudos de Caso

Os estudos de caso apresentados neste Capítulo foram executados em um simulador do processador ERC32 utilizado no COMAV, o *SPARC Instruction Simulator* (SIS).

Para verificar a adequação da GOESA ao *hardware* de voo, o estudo de caso para o Amazonia-1 também foi executado no protótipo do COMAV existente no laboratório do Grupo de Supervisão de Bordo (SUBORD) da Divisão de Eletrônica Aeroespacial (DEA) da Engenharia do INPE.

Foi criada uma interface do tipo menu, com opções numéricas fornecidas por console a um operador. A interface permite tanto a execução do planejador, como de sessões de inferência sem planejamento, modo no qual o operador pode exercitar a adição de

restrições e as operações de inferência, analisando os resultados providos pelo ISIS. A Figura 8.1 apresenta uma das telas da interface com o operador.

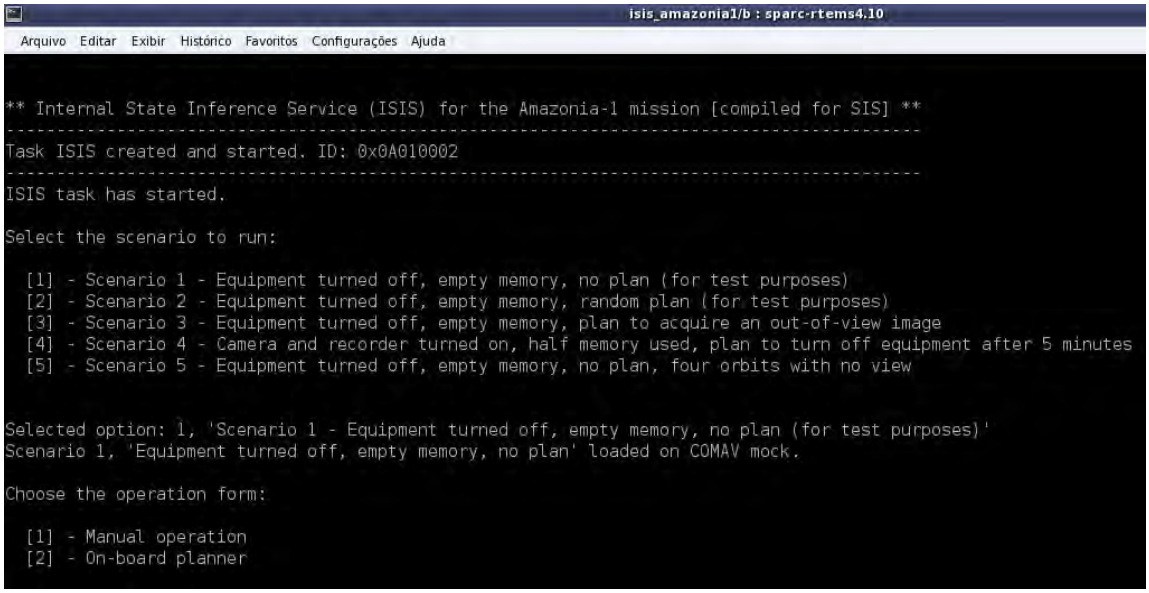


Figura 8.1 - Interface de operação via console

Em ambos os casos (execução com ou sem o planejador) é possível selecionar um cenário inicial entre um conjunto predefinido, e adicionar restrições/objetivos predefinidos ou criados pelo operador.

Ao optar pelo planejador, é possível configurá-lo para pausar a execução após ‘n’ laços de planejamento, para que o usuário verifique as decisões tomadas pelo planejador e o estado intermediário inferido pelo ISIS para o sistema modelado.

8.2. O Estudo de Caso para a Missão Amazonia-1

8.2.1. Descrição do Domínio a Ser Modelado e dos Objetivos

A missão Amazonia-1 é composta por um satélite de sensoriamento remoto baseado na PMM. Ele será posicionado em uma órbita polar heliossíncrona a cerca de 750 km de altitude. O período da órbita é de 100 minutos, com aproximadamente 35 minutos em eclipse (INPE, 2010).

O satélite é equipado com uma câmera do tipo CCD chamada *Advanced Wide-Field Imager* (AWFI), um gravador digital (*Digital Data Recorder*, DDR) e um transmissor de dados (*Data Transmitter*, DT). Esses três equipamentos constituem a carga útil da missão.

Para o estudo de caso foi assumida a existência de apenas uma estação terrena, com tempo de visada de 13 minutos por passagem.

A obtenção de imagens fora de visada é realizada através de sequências de comandos temporizados. É preciso ligar a câmera AWFI e o gravador de dados DDR e mantê-los ligados por todo o período da imagem a ser obtida. Quando o satélite termina sua passagem sobre a área a imagear, a AWFI e o DDR devem ser desligados.

Na próxima passagem sobre a estação terrena, é necessário comandar a reprodução dos dados gravados no DDR e ligar o transmissor DT para que a imagem obtida seja enviada para solo. Após transmitir todos os dados, a memória do DDR deve ser apagada para permitir novas gravações.

Uma chave eletrônica (*Digital Signal Switch*, DSS) pertencente ao DDR determina se os dados enviados ao DT provêm de sua memória ou diretamente da AWFI. A posição da DSS é controlada por comandos.

Todos os equipamentos são supridos eletricamente pelo subsistema de potência (*Power Supply Subsystem*, PSS). Para utilizá-los, é preciso ligar suas linhas de suprimento, e em seguida enviar comandos do tipo *on/off* para cada equipamento. Uma vez ligados, os equipamentos devem ser colocados em seus devidos modos de operação.

O PSS também é responsável pelo controle dos painéis solares, que são móveis. Isso ocorre através do *Solar Array Drive Assembly* (SADA). Como a rotação dos painéis gera uma vibração no corpo do satélite, eles devem estar parados durante a captura de imagens pela câmera.

Os objetivos da operação do Amazonia-1 são sempre *"adquirir e gravar uma imagem entre os momentos A e B, e transmiti-la para solo"*, com algumas variações quanto ao número de imagens a adquirir, e se a transmissão faz parte ou não do objetivo.

A maioria das informações usadas para a modelagem foi obtida da documentação técnica da missão Amazonia-1 e do programa PMM. Algumas informações que ainda não haviam sido definidas foram assumidas com base em dados preliminares e na experiência prévia de engenheiros do Grupo SUBORD do INPE.

Os itens seguintes relatam o processo de modelagem do domínio aqui descrito. O modelo completo encontra-se transcrito no Apêndice A desta Tese.

8.2.2. A Criação da Descrição Estrutural

Os elementos identificados para a modelagem foram os equipamentos que compõem a carga útil (AWFI, DDR e DT) e o subsistema PSS, devido ao fornecimento de energia aos demais e ao controle da rotação dos painéis solares.

A única *timeline* criada para a AWFI representa seu modo de operação. A AWFI possui três modos: '*power off*', quando não há suprimento de energia, '*stand-by*', quando há suprimento mas a câmera não está em operação, e '*imaging*', quando a câmera está em operação. Foram criadas enumerações na Descrição Estrutural para representar esses e outros modos dentro do modelo.

Também foi criada uma *timeline* para representar o modo de operação do DDR. Os valores possíveis para essa *timeline* são: '*power off*', quando não há suprimento de energia, '*starting up*', um modo transitório no qual o DDR encontra-se em processo de inicialização e auto-teste, '*stand-by*', quando o equipamento está ligado mas sem realizar nenhuma operação, '*record*', quando está gravando dados, '*playback*', quando está reproduzindo os dados gravados, e '*erase*', modo no qual se mantém enquanto seus dados estão sendo apagados.

Além do modo, o DDR possui mais três *timelines*: '*Ready To Operate*', que indica se a inicialização e auto-teste já foram executados, um booleano '*Memory Full*', e '*DSS State*', que informa a posição da chave eletrônica.

O DDR fornece um recurso consumível, que é sua memória de massa. São 160.000 Megabits disponíveis. De forma a gerenciar a transmissão dos dados para solo, um segundo recurso foi criado, '*Data To Send*', com o mesmo volume da memória. Os recursos memória e dados a enviar são mutuamente exclusivos: quanto mais dados a enviar, menos memória disponível. Ambos os recursos são consumíveis por taxas, e o único consumidor é o próprio DDR.

O DT possui apenas duas *timelines*: uma que indica seu modo ('*power off*', '*stand-by*' ou '*nominal*') e outra que indica se o satélite encontra-se em visada e, portanto, em comunicação com a estação terrena.

O PSS possui um recurso consumido por todos os demais elementos do modelo: energia elétrica. Da quantidade total de energia, 350 Watts são dedicados à carga útil. Quando o satélite encontra-se em eclipse e os painéis solares não captam luz solar, essa quantidade cai para 300 Watts.

O PSS possui também *timelines* para indicar o status da linha de suprimento de energia de cada um dos demais equipamentos, uma que indica se os painéis solares (SADA) estão se movendo ou não, e uma que informa a fase da órbita: se o satélite encontra-se ou não em uma zona de eclipse.

A Tabela 8.1 lista as *timelines*, recursos e domínios de valores existentes no modelo criado para o estudo de caso da missão Amazonia-1.

Tabela 8.1 - Componentes da Descrição Estrutural do modelo Amazonia-1

Elemento	Timeline	Recurso	Domínio de Valores
AWFI	Mode		Power Off, Stand-By, Imaging
DDR	Mode		Power Off, Starting Up, Stand-By, Record, Playback, Erase
	Ready to Operate		True, False
	Memory Full		True, False
	DSS State		Real-Time, Playback
		Memory	0 a 160000
		Data to Download	0 a 160000
DT	Mode		Power Off, Stand-By, Nominal
	Communicating with Ground		True, False
PSS	Orbit Period		Sunlight, Eclipse
	SADA State		Moving, Stopped
	AWFI Line Status		On, Off
	DDR Line Status		On, Off
	DT Line Status		On, Off
		Power	0 a 350 (0 a 300 quando em eclipse)

8.2.3. A Criação da Descrição Comportamental

Uma vez definidas as *timelines* e recursos, a próxima etapa é a criação da Descrição Comportamental. Para isso foi necessário identificar na documentação técnica, entre os comandos aos equipamentos, aqueles que modificam os estados das *timelines* e recursos modelados.

A Tabela 8.2 lista todas as operações identificadas no estudo de caso da missão Amazonia-1, com suas precondições e sobre quais *timelines* e recursos elas operam.

Tabela 8.2 - Operações identificadas para o modelo Amazonia-1

Elem.	Ação ou Evento	Descrição	Precondições ou condição para ocorrência	Opera sobre ou consome
AWFI	Turn On DC/DC Converters	Liga a câmera, colocando-a em modo imaging.	AWFI.Mode em stand-by, SADA parado, satélite fora de eclipse, (DDR.Mode em record ou DDR.DssState em real-time)	AWFI.Mode, PSS.Power
	Turn Off DC/DC Converters	Desliga a câmera, colocando-a em modo stand-by.	AWFI.Mode em imaging	AWFI.Mode, PSS.Power
DDR	Turn On DC/DC Converter	Liga o gravador, deixando-o por 15 segundos em modo starting up e posteriormente em stand-by (evento endógeno).	Linha de suprimento do DDR ligada, DDR.Mode em power off	DDR.Mode, PSS.Power
	Turn Off DC/DC Converter	Desliga o gravador, colocando-o em modo power off.	Linha de suprimento do DDR ligada	DDR.Mode, DDR.ReadyToOperate, PSS.Power
	Playback	Inicia a transmissão para o DT dos dados gravados.	DT.Mode em nominal, DDR.Mode em stand-by, DDR.DssState em playback	DDR.Mode, PSS.Power, DDR.DataToDownload
	Erase File	Apaga os dados gravados, liberando memória.	DDR.Mode em stand-by	DDR.Mode, DDR.Memory
	Stop Playback	Interrompe a transmissão de dados para o DT.	DDR.Mode em playback	DDR.Mode, PSS.Power, DDR.DataToDownload
	Start Recording	Inicia a gravação de dados.	DDR.Mode em stand-by, DDR.MemoryFull em false, DDR.DssState em real-time	DDR.Mode, PSS.Power, DDR.Memory, DDR.DataToDownload

continua

Tabela 8.2 - Continuação

Elem.	Ação ou Evento	Descrição	Precondições ou condição para ocorrência	Opera sobre ou consome
DDR	Stop Recording	Interrompe a gravação de dados.	DDR.Mode em record	DDR.Mode, PSS.Power, DDR.Memory
	AWFI Channel Playback	Muda o estado do DSS para playback.	DDR.DssState em real-time	DDR.DssState
	AWFI Channel Real-Time	Muda o estado do DSS para real-time.	DDR.DssState em playback	DDR.DssState
DT	DT-AWFI Channel Turn On	Coloca o DT em modo nominal, para a transmissão de dados a solo.	DT.Mode em stand-by, comunicação com solo estabelecida	DT.Mode, PSS.Power
	DT-AWFI Channel Turn Off	Desliga o DT.	DT.Mode em nominal	DT.Mode, PSS.Power
PSS	Switch On AWFI Line	Liga a linha de suprimento de energia da AWFI.	Linha da AWFI desligada	AWFI.Mode, PSS.Power, PSS.AwfiLineStatus
	Switch Off AWFI Line	Desliga a linha de suprimento de energia da AWFI.	Linha da AWFI ligada	AWFI.Mode, PSS.Power, PSS.AwfiLineStatus
	Switch On DDR Line	Liga a linha de suprimento de energia do DDR.	Linha do DDR desligada	PSS.DdrLineStatus
	Switch Off DDR Line	Desliga a linha de suprimento de energia do DDR.	Linha do DDR ligada	PSS.DdrLineStatus
	Switch On DT Line	Liga a linha de suprimento de energia do DT.	Linha do DT desligada	DT.Mode, Pss.DtLineStatus, PSS.Power
	Switch Off DT Line	Desliga a linha de suprimento de energia do DT.	Linha do DT ligada	DT.Mode, PSS.DtLineStatus, PSS.Power
	Stop SADA	Interrompe a movimentação dos painéis solares.	Painéis solares em movimento	PSS.SadaState
	Resume SADA	Retoma a movimentação dos painéis solares.	Painéis solares parados	PSS.SadaState

continua

Tabela 8.2 - Conclusão

Elem.	Ação ou Evento	Descrição	Precondições ou condição para ocorrência	Opera sobre ou consome
Eventos Exógenos	Enter in Eclipse	Indica a entrada do satélite em zona de eclipse.	N/A	PSS.OrbitPeriod
	Exit Eclipse	Indica a saída do satélite da zona de eclipse.	N/A	PSS.OrbitPeriod
	Start Communication	Indica o início da comunicação com a estação terrena.	N/A	DT.Communicating WithGround
	Finish Communication	Indica o término da comunicação com a estação terrena.	N/A	DT.Communicating WithGround
Eventos Endógenos	Memory Full	Indica que não há mais memória disponível.	Consumo de memória atingir o máximo	DDR.Memory, DDR.MemoryFull, DDR.Mode, PSS.Power
	Memory Empty	Indica que não há mais dados a transmitir para solo.	Memória totalmente apagada	DDR.Memory, DDR.Mode, PSS.Power
	DDR Ready	Indica que o DDR está pronto para receber comandos.	Start-up do DDR finalizado (15 segundos após ligar)	DDR.Mode, DDR.ReadyToOperate

Seguem algumas considerações sobre as principais decisões de modelagem tomadas na definição dessas operações:

- Os efeitos da maioria das ações resumem-se a modificar o estado de uma ou duas *timelines* e a quantidade de energia consumida – que varia de acordo com o modo de operação de cada elemento;

- O modelo sofreu apenas duas simplificações com relação à operação dos equipamentos reais:

    1. O sistema de arquivos do DDR foi ignorado, o que simplificou os comandos '*Playback*' e '*Erase File*';

2. Os cinco modos de operação existentes no SADA foram reduzidos a apenas dois: painéis solares parados ou em movimento; além disso, a capacidade de movimentação independente dos painéis foi ignorada. Sua movimentação pode ser comandada de forma conjunta, e esse tipo de comando é o que se adotou no modelo.

- O DDR possui um período de inicialização, após ligado, no qual não pode receber comandos. Isso foi modelado como um modo transitório '*starting up*', que é definido ao ligar o equipamento, e do qual se sai através do evento endógeno '*DDR Ready*', que ocorre 15 segundos após o DDR ser ligado;

- Os eventos endógenos '*Memory Full*' e '*Memory Empty*' indicam os momentos em que o DDR, após os comandos '*Start Recording*' e '*Erase File*', tem sua memória totalmente preenchida ou totalmente esvaziada. Nessas condições o DDR automaticamente finaliza a operação em andamento e volta ao modo '*stand-by*'. Tais eventos não ocorrem se comandos subsequentes interrompem a gravação ou limpeza da memória antes que essas se completem;

- Operacionalmente não faz sentido ligar a câmera quando o satélite encontra-se em uma zona de eclipse, período em que não há iluminação suficiente para as imagens. Assim, uma *timeline* foi criada para informar ao Motor de Inferência as fases da órbita. Essa *timeline* é precondição para ligar a câmera, e seus valores são modificados apenas pelos eventos exógenos '*Enter in Eclipse*' e '*Exit Eclipse*';

- A ação para ligar o transmissor de dados só deve ocorrer quando houver visibilidade entre o satélite e a estação terrena. Uma *timeline* foi criada e é utilizada de forma análoga àquela que informa a fase da órbita. Essa *timeline* só é operada pelos eventos exógenos '*Start Communication*' e '*Finish Communication*'.

8.2.4. O Gerenciador de Objetivos para a Missão Amazonia-1

Conforme colocado anteriormente na Tabela 4.1, o Gerenciador de Objetivos da GOESA é específico para cada aplicação, devendo ser customizado. Isso se deve à natureza dos objetivos em cada missão, e à sua conversão em restrições, que é fortemente vinculada ao modelo embarcado no ISIS.

Para o estudo de caso do satélite Amazonia-1 foi criado um protótipo de Gerenciador de Objetivos. Esse protótipo recebe os objetivos, os converte em restrições, abre uma sessão de inferência e aplica as restrições ao ISIS.

Múltiplos objetivos podem ser estabelecidos, a qualquer momento em uma sessão de inferência. Entretanto, o protótipo não verifica a consistência dos objetivos recebidos, ou se há conflito entre diferentes objetivos.

Há as seguintes variações de objetivos, que o operador pode informar:

- Adquirir uma única imagem entre os momentos A e B e gravar;
- Adquirir uma única imagem entre os momentos A e B, gravar e transmitir;
- Adquirir múltiplas imagens em diferentes períodos especificados e gravá-las;
- Adquirir múltiplas imagens em diferentes períodos especificados, gravá-las e transmiti-las.

Cenários iniciais distintos podem ser definidos através da interface de operação. A composição entre diferentes cenários e objetivos permite verificar a adaptabilidade dos planos gerados à situação na qual o satélite se encontra.

8.2.5. Operação Baseada em Objetivos para o Amazonia-1 com a GOESA

Segue um exemplo de uma simulação da operação baseada em objetivos com a GOESA para o estudo de caso da missão Amazonia-1. As referências temporais estão no

formato do ISIS, como uma contagem absoluta de inteiros compatível com o tempo de bordo.

8.2.5.1. O Cenário Inicial e o Objetivo Recebido

O cenário inicial para a simulação aqui descrita é o seguinte:

- Sessão de inferência com início no momento 1200 segundos e término em 14400 segundos;
- Todos os equipamentos da carga útil desligados, inclusive suas linhas de suprimento elétrico, no momento inicial;
- Plano original preexistente, que consiste em adquirir uma imagem entre os momentos 7200 segundos e 7800 segundos;
- Tabela de Predição da Ocorrência de Eventos Exógenos com a previsão de duas órbitas, e duas passagens sobre a estação terrena.

O objetivo recebido para a simulação foi: *"obter uma imagem entre os momentos 1300s e 1500s, gravá-la e transmitir todos os dados armazenados para solo"*.

8.2.5.2. A Transformação do Objetivo em Restrições

O Gerenciador de Objetivos converteu o objetivo recebido num conjunto de restrições enviadas ao ISIS e listadas abaixo:

- Manter a câmera AWFI em modo '*imaging*', o gravador DDR em modo '*record*' e o SADA em modo '*stopped*' entre os momentos 1300 e 1500;
- No momento 1510, a AWFI deve estar em modo '*stand-by*', o DDR em modo '*stand-by*', e o SADA em modo '*moving*';
- Durante cada passagem prevista sobre a estação terrena, manter o DDR em modo '*playback*' e o DT em modo '*nominal*';

- Após cada passagem prevista sobre a estação terrena, o DDR e o DT devem ser colocados em seus respectivos modos '*stand-by*'.

O objetivo apenas é alcançado pelo planejador quando todas essas restrições do usuário são respeitadas, sem violar nenhuma outra restrição imposta pelo modelo, e sem conflitos com o plano preexistente.

8.2.5.3. O Plano de Operações Gerado pelo LetMeDo

A Tabela 8.3 traz o plano resultante do processo de planejamento. As modificações realizadas pelo planejador para atingir o objetivo que lhe foi passado estão destacadas na cor laranja.

Dados de desempenho deste e outros cenários são apresentados na seção 8.2.6.2 deste Capítulo.

Tabela 8.3 - Plano gerado pelo LetMeDo

Objetivos	Momento	Tipo de Ocorrência	Descrição
	1200	Início de Sessão	-
Objetivo: obter e gravar uma imagem entre os momentos 1300 e 1500.	1268	Ação	PSS: Switch On DDR Line
	1269	Ação	DDR: Turn On
	1284	Evento Endógeno	DDR Ready (causado por 'DDR: Turn On' em 1269)
	1290	Ação	PSS: Stop SADA
	1298	Ação	PSS: Switch On AWFI Line
	1299	Ação	AWFI: Turn On
	1299	Ação	DDR: Start Recording
	1509	Ação	AWFI: Turn Off
	1509	Ação	DDR: Stop Recording
	1509	Ação	PSS: Resume (move) SADA
	3000	Evento Exógeno	Enter in Eclipse
Objetivo: transmitir imagem gravada para solo.	4199	Ação	PSS: Switch On DT Line
	4200	Evento Exógeno	Start Communication
	4200	Ação	DT: DT-AWFI Channel Turn On
	4208	Ação	DDR: AWFI Channel Playback
	4209	Ação	DDR: Playback
	4979	Ação	DDR: Stop Playback
	4979	Ação	DT: DT-AWFI Channel Turn Off
	4980	Evento Exógeno	Finish Communication
	5100	Evento Exógeno	Exit Eclipse
Aquisição e gravação de imagem prevista pelo plano original.	7181	Ação	PSS: Stop SADA
	7181	Ação	PSS: Switch Off AWFI Line
	7182	Ação	PSS: Switch On AWFI Line
	7182	Ação	PSS: Switch Off DDR Line
	7183	Ação	PSS: Switch On DDR Line
	7184	Ação	DDR: Turn On
	7199	Evento Endógeno	DDR Ready (causado por 'DDR: Turn On' em 7184)
	7199	Ação	DDR: AWFI Channel Record
	7200	Ação	DDR: Start Recording
	7200	Ação	AWFI: Turn On
	7800	Ação	DDR: Stop Recording
	7801	Ação	PSS: Resume (move) SADA
	7802	Ação	AWFI: Turn Off
	9000	Evento Exógeno	Enter in Eclipse
Objetivo: transmitir imagem gravada para solo.	10200	Evento Exógeno	Start Communication
	10200	Ação	DT: DT-AWFI Channel Turn On
	10208	Ação	DDR: AWFI Channel Playback
	10209	Ação	DDR: Playback
	10979	Ação	DDR: Stop Playback
	10979	Ação	DT: DT-AWFI Channel Turn Off
	10980	Evento Exógeno	Finish Communication
	11100	Evento Exógeno	Exit Eclipse
	14400	Fim de Sessão	-

8.2.5.4. Considerações sobre o Plano Resultante

O *LetMeDo* inseriu no plano os comandos necessários para atender aos objetivos, respeitando o comportamento modelado para as cargas úteis e o plano de operações original.

Ao verificar o plano resultante, destaca-se a inserção de comandos para desligar as linhas de suprimento elétrico da AWFI e do DDR nos momentos 7181 e 7182, respectivamente. Isso merece alguma análise.

O plano original previa que as linhas fossem ligadas apenas nesses momentos. A adição de comandos para a aquisição de imagens no início do plano fez com que as linhas de suprimento passassem a ser ligadas antes, e se mantivessem assim.

Dessa forma, as precondições das ações no plano original para ligar as linhas – que estabeleciam que as linhas deveriam estar desligadas – passaram a ser falsas. Para atender ao plano original, o *LetMeDo* optou por desligar as linhas imediatamente antes dos comandos que as ligavam.

Uma atitude mais 'inteligente' do planejador seria simplesmente remover os comandos '*Switch On AWFI/DDR Line*' entre os momentos 7182 e 7183, uma vez que as linhas já estavam ligadas.

A opção do *LetMeDo*, entretanto, atende às condições e restrições estabelecidas no modelo. Uma melhoria possível consistiria de ajustar o modelo para induzir o planejador a tomar uma decisão mais 'inteligente'.

8.2.6. Dados de Desempenho da GOESA para o Estudo de Caso Amazonia-1

8.2.6.1. Tamanho e Alocação de Memória dos Componentes

A Tabela 8.4 apresenta o tamanho dos componentes da GOESA compilados, e a necessidade de memória RAM para a execução de cada um deles.

Tabela 8.4 - Tamanho e alocação de memória dos componentes da GOESA

Componente da GOESA	Tamanho compilado (sem otimização)	Tamanho compilado (com otimização)	Consumo de RAM (seções .data e .bss)
ISIS	65,14 kbytes	33,03 kbytes	46,75 kbytes
LetMeDo	27,17 kbytes	13,42 kbytes	46,22 kbytes
Modelo Embarcado para o Amazonia-1	15,22 kbytes	9,00 kbytes	61,42 kbytes
Gerenciador de Objetivos	1,96 kbytes	0,84 kbytes	0
Totais	**109,49 kbytes**	**56,29 kbytes**	**154,39 kbytes**

Tais dados foram obtidos com o compilador GCC para o processador ERC32, o mesmo utilizado para o *software* de voo do COMAV. Os dados foram obtidos com os parâmetros de compilação '-o0 -g3' (sem otimização) e '-oS' (com otimização).

Esses valores demonstram que a GOESA é compatível com os satélites da PMM em termos de requisitos de memória de programa e de dados.

8.2.6.2. Tempo de Planejamento para o Estudo de Caso Amazonia-1

A Tabela 8.5 traz os tempos de planejamento para cenários aplicados ao estudo de caso Amazonia-1. A configuração do planejador na qual foram obtidos esses tempos foi a seguinte:

- Priorizar a resolução de violações imediatas em detrimento de outras violações na Lista de Ocorrências;
- Não buscar pelo melhor caminho em situações de empate;
- Não adicionar perturbações ao plano;
- Seis operações de planejamento habilitadas, escolhidas empiricamente como as mais efetivas para o modelo do Amazonia-1.

Os tempos foram registrados no SIS, operando à taxa de 15 MHz (5.8 MIPS). A alocação do tempo de processador foi de 100%.

Tabela 8.5 - Desempenho da GOESA em cenários para o modelo Amazonia-1

Cenário	Status Inicial	Duração da sessão	Objetivos	Tempo de planejamento
1	Todos os equipamentos desligados, sem plano preexistente.	Entre 0s e 12000s	Obter e gravar uma imagem entre 5000s e 5500s	55,54 segundos
2	Equipamentos mantidos ligados por 5 minutos a partir do início da sessão (obtendo uma imagem), plano original para obter e gravar uma segunda imagem entre 7200s e 7800s.	Entre 1200s e 13200s	Obter e gravar uma nova imagem entre 8000s e 8500s	62,79 segundos
3	Todos os equipamentos desligados, plano original para obter e gravar uma imagem entre 7200s e 7800s, duas órbitas e duas passagens sobre a estação terrena previstas na tabela de eventos exógenos.	Entre 1200s e 14400s	Obter e gravar uma nova imagem entre 1300s e 1500s, enviar todos os dados para solo.	24,50 minutos

Seguem algumas considerações sobre os cenários:

- O cenário 1 constitui a criação de um novo plano a partir de um objetivo recebido, sem um plano preexistente;
- O cenário 2 constitui a modificação de um plano preexistente para comportar um novo objetivo, com pouco impacto sobre o plano original;
- O cenário 3, cujo plano resultante é apresentado na Tabela 8.3, é o mais realista. Ele considera a modificação de um plano existente para obter uma nova imagem e enviar todos os dados armazenados para solo. Como a nova imagem deve ser obtida antes daquela prevista no plano original, há um impacto maior sobre o plano original do que no cenário 2.

Em diversos cenários de criação de um novo plano de operações, ou de modificações com pouco impacto em planos preexistentes, o tempo de planejamento do *LetMeDo* esteve entre 1 e 3 minutos.

O tempo de planejamento para o cenário 3, considerado o pior caso, permitiria ao *LetMeDo* gerar uma resposta ao objetivo recebido num período de 100 minutos, fazendo uso de 25% do tempo do processador. Ou seja, se um plano fosse solicitado em uma órbita, ele estaria pronto na seguinte.

Prevê-se que as missões futuras da PMM utilizem processadores LEON2, que trabalham ao dobro da frequência do ERC32 presente no COMAV e também no Amazonia-1. Estima-se que, com o uso de tais processadores e uma eventual otimização do planejador e do ISIS, o tempo de planejamento para os piores cenários possa ficar entre 10 e 20 minutos, mantendo a alocação de 25% do processador.

8.3. Estudo de Caso para a Missão EQUARS

Conforme colocado no início deste Capítulo, o segundo estudo de caso criado para a GOESA teve por objetivo comprovar a aplicabilidade da arquitetura para diferentes missões, e também propiciar uma comparação de desempenho entre o *LetMeDo* e o RASSO.

Esse estudo de caso envolve a alocação adicional de recursos para experimentos científicos pertencentes à missão EQUARS, quando estes detectam a ocorrência de fenômenos físicos de curta duração em órbita.

O cenário utilizado foi o mesmo descrito por Kucinskis (2007), assim como o objetivo: alocar mais energia e memória para um dos experimentos por um dado período de tempo.

Enquanto o RASSO encontrou um plano em 114,28 segundos (KUCINSKIS, 2007), o *LetMeDo*, sendo executado no mesmo computador e sob as mesmas condições, o fez em 119,21 segundos. Isso demonstra um desempenho comparável – e com o diferencial que, ao contrário do RASSO, o *LetMeDo* não foi desenvolvido visando uma missão específica.

Como não se considera o modelo do RASSO para a missão EQUARS representativo de uma operação real, não foram exercitados novos cenários com ele.

9. CONCLUSÃO

Esta Tese apresenta a operação de missões espaciais baseadas em objetivos como uma evolução do paradigma vigente de operação por sequências de comandos.

Foi proposta uma arquitetura de *software* embarcado no segmento espacial para habilitar a adoção do novo paradigma em missões futuras do INPE. Tal arquitetura é adequada à execução a bordo de satélites, é reutilizável entre missões, e unifica os conceitos de planejamento e escalonamento.

Os principais componentes da arquitetura foram desenvolvidos, e suas capacidades foram exercitadas com dois estudos de caso, um deles baseado na documentação técnica de uma missão atual do Instituto. Dados de desempenho demonstram o potencial da aplicação desta tecnologia em missões reais.

Neste Capítulo final são listadas as principais contribuições deste trabalho, o status do desenvolvimento dos componentes da arquitetura, e os trabalhos futuros identificados para a continuidade desta linha de pesquisa.

9.1. Principais Contribuições

Foi concebida e desenvolvida uma arquitetura de *software* capaz de habilitar a operação de missões espaciais baseada em objetivos, e consequentemente aumentar a autonomia do segmento espacial, através de planejamento embarcado.

Tal arquitetura apresenta as seguintes características que constituem contribuições ao estado da arte em operações baseadas em objetivos para missões espaciais:

- foi integrada a um típico *software* de voo e executada em *hardware* de voo de satélites, algo que a literatura registra como havendo ocorrido em apenas duas missões até o momento, ambas da NASA;

- diferentemente desses dois casos, a arquitetura se propõe a ser reutilizável entre missões com características diversas e unifica os módulos de planejamento e escalonamento em um único sistema. Isso implica na existência de um único modelo embarcado, outra característica que a distingue de propostas similares;

- para atender a esse modelo único, uma nova linguagem de modelagem para o planejamento foi criada, provendo uma descrição declarativa das ações disponíveis. Assim, técnicas gerais de planejamento podem ser utilizadas para que os objetivos recebidos sejam alcançados. Essa linguagem minimiza a abstração do sistema modelado, o que permite uma descrição do domínio mais próxima da operação real;

- embora todo o *software* tenha sido desenvolvido com foco nas missões do INPE que fazem uso da PMM, ele é compatível com qualquer missão que adote o *Packet Utilization Standard*, padrão da ECSS usado atualmente por diversas agências espaciais.

Com a finalidade de promover a continuidade do desenvolvimento, foi gerado um relatório técnico para a Divisão de Engenharia Aeroespacial do INPE, descrevendo em maiores detalhes a GOESA e seus componentes (INPE, 2012).

O produto desta Tese está em condições de ser considerado para um experimento de validação tecnológica em uma missão futura do Instituto.

9.2. Status do Desenvolvimento da GOESA

O presente trabalho propôs a arquitetura GOESA e desenvolveu seus principais componentes. A Tabela 9.1 apresenta o status do desenvolvimento dos componentes da arquitetura.

Tabela 9.1 - Status do desenvolvimento dos componentes da GOESA

Componente	Status atual e considerações
ISIS	Implementado completamente.
Modelos Embarcados	Dois modelos foram implementados para missões diferentes.
Determinador Embarcado de Objetivos	Não implementado. Sua criação dependerá de uma missão que possua requisitos específicos para a resposta autônoma a eventos detectados em órbita.
Gerenciador de Objetivos	Dois protótipos foram desenvolvidos, que convertem objetivos em restrições. A consistência dos objetivos recebidos ou potenciais conflitos entre múltiplos objetivos não são verificados.
Planejador Embarcado	Um planejador de referência, não otimizado, foi desenvolvido.
Linguagem de descrição de modelos	Implementada completamente.
Parser da linguagem de descrição de modelos	Não desenvolvido. As transformações necessárias para tornar a descrição do domínio em código executável já estão definidas.

9.3. Trabalhos Futuros

9.3.1. Exercício da GOESA com Novos Estudos de Caso

É necessário exercitar a GOESA com novos modelos para diferentes missões espaciais, preferencialmente aquelas pertencentes ao programa PMM do INPE – mas não apenas essas.

O ideal é que futuros trabalhos, que se proponham a contribuir com o desenvolvimento da arquitetura e seus componentes, adotem estudos de caso para novas missões, sempre buscando fidelidade com relação à operação real dos satélites.

Isso propiciará uma melhor exploração dos recursos da linguagem ISISml e eventualmente a identificação de melhorias e correções a aplicar sobre o serviço ISIS.

Identificam-se as missões Lattes-1 e principalmente a GTEO como candidatas a futuros estudos de caso. Em uma apresentação sobre a GTEO para a *National Academy of Sciences*, os responsáveis pela missão (então chamada de 'Flora') apontaram que "a operação autônoma como a demonstrada na missão EO-1 seria uma capacidade valiosa, e apropriada para uma missão pequena [como esta]" (ASNER et al., 2005, p. 10).

9.3.2. Gerenciamento de Objetivos

A criação de um Gerenciador de Objetivos completo é provavelmente o trabalho futuro de maior prioridade para a GOESA. Os protótipos criados para os estudos de caso não verificam a consistência dos objetivos recebidos, ou possíveis conflitos entre múltiplos objetivos.

Para o estudo de caso da missão Amazonia-1, por exemplo, seria aceito um pedido de aquisição de imagens para um período em que o satélite se encontre em eclipse. Nesse caso, como o objetivo vai contra as restrições do modelo, não há uma solução possível e o processo de planejamento seria encerrado com falha. A verificação desse tipo de situação deve ser adicionada, permitindo à GOESA identificar e rejeitar objetivos inconsistentes.

A flexibilização de objetivos, como a possibilidade de atendimento parcial ao que for pedido, ou a adição de objetivos categorizados como 'desejáveis' – além dos mandatórios –, também pode ser estudada.

O estudo da aplicabilidade de técnicas de consistência local para CSPs e propagação de restrições pode apontar caminhos para responder a essas questões.

Finalmente, apesar de um Gerenciador de Objetivos ser específico para uma dada aplicação, parte dele poderia ser de uso geral. Seria interessante tentar determinar o quão independente do modelo esse Gerenciador pode ser.

Apontam-se as seguintes referências a serem estudadas como ponto de partida para um gerenciamento mais completo dos objetivos: Dvorak et al. (2007), Dvorak et al. (2000), Ingham et al. (2005), Dvorak et al. (2008) e Bennett et al. (2008).

9.3.3. Planejadores Embarcados com Desempenho Otimizado

O planejador desenvolvido como parte desta Tese não é otimizado. Portanto, um trabalho relevante seria a criação de planejadores embarcados que operem sobre o

ISIS com algoritmos de melhor desempenho. Deve ser dada maior atenção às técnicas da área de Pesquisa Operacional.

Estudos para a melhoria do desempenho do planejador *LetMeDo* e do próprio ISIS também são de interesse.

9.3.4. Determinação de Objetivos a Bordo

O Determinador Embarcado de Objetivos previsto na GOESA para a operação autônoma de missões não foi implementado. Para que o seja, é preciso identificar missões que tenham como requisitos a detecção de eventos em órbita e a geração de respostas autônomas a eles.

Como exemplo cita-se a missão MIRAX que visa "detectar, localizar e identificar fenômenos de curta duração, raros e/ou imprevisíveis, incluindo transientes de raios X e novas 'rápidas' de raios X" (ROTHSCHILD, 2003, p. 1).

Em situações como essa, os tempos de resposta são muito pequenos para a comunicação com o segmento solo, e a GOESA, completada por um Determinador Embarcado de Objetivos, torna-se uma ferramenta importante.

Até o momento, apenas os sistemas ASE e AEGIS da NASA implementaram a determinação de objetivos no segmento espacial.

9.3.5. Validação dos Modelos Embarcados e Aprendizado

A validação dos modelos embarcados pode ser realizada através da comparação dos estados inferidos a partir da descrição do domínio que o modelo contém, com os valores aferidos pelo *software* de voo.

Deve ser levado em conta o fato de que um modelo válido não o será indefinidamente; o satélite envelhece, seus equipamentos irão falhar, e o comportamento previsto pelo modelo não estará mais de acordo com o comportamento real. Algumas considerações

iniciais a esse respeito, relacionadas ao presente trabalho, encontram-se descritas em Kucinskis (2011) e Kucinskis e Ferreira (2011).

Podem ainda ser exploradas técnicas de aprendizado computacional para permitir que as modificações no comportamento do satélite sejam detectadas, e que o modelo se adapte a isso. O principal desafio, nesse caso, será identificar e adaptar técnicas com desempenho e confiabilidade compatíveis com missões espaciais.

9.3.6. Execução e Monitoramento de Planos em Tempo Real

A GOESA deixa a execução dos planos que cria ou modifica a cargo do *software* de voo convencional. As arquiteturas de alguns dos sistemas descritos no Capítulo 3 adotam para esse fim um monitor e executor de planos para o replanejamento dito 'reativo' ou 'de tempo real'. Tais componentes não existem na GOESA, mas poderiam ser adicionados caso sejam identificados requisitos de missões do INPE que possam ser atendidos por esse tipo de replanejamento.

9.3.7. Ferramentas de Suporte ao Desenvolvimento de Modelos

O presente trabalho não contemplou a criação de ferramentas de suporte ao desenvolvimento dos modelos a serem embarcados. São três as ferramentas necessárias: um *parser* para a linguagem ISISml, um editor de modelos, e um visualizador de *timelines*.

Dentre essas, o *parser* da ISISml é claramente a mais importante. As transformações a serem realizadas por ele podem ser verificadas no Apêndice A. Maiores detalhes estão registrados em um relatório técnico relacionado a esta Tese (INPE, 2012). Essa tarefa envolve conceitos de Linguagens Formais, Autômatos e Compiladores, além de um bom conhecimento da linguagem de programação C++.

O editor de modelos deve prover recursos gráficos para a composição e depuração de modelos embarcados na GOESA, e deve ser similar aos ambientes de desenvolvimento

de *software* (*Integrated Development Environment*, IDE). De fato, ele poderia ser criado sobre um IDE existente: o *Visual Studio 2008 Shell* e o *Eclipse RCP* são exemplos de IDEs que podem ser utilizados como plataforma para a criação de um editor de modelos em ISISml.

Finalmente, um visualizador de *timelines* constitui uma ferramenta de produtividade importante para o desenvolvedor de modelos. Atualmente a única forma de análise dos estados inferidos pelo ISIS é através da interface via console, que exibe informações de forma numérica.

Embora tenha se mostrado um recurso importante, tal forma de análise é cansativa ao desenvolvedor e propensa a erros. Um visualizador gráfico dos estados inferidos pelo ISIS propiciaria uma melhora significativa na análise do comportamento previsto pelos modelos.

Um protótipo para esse visualizador foi desenvolvido em *Java* sobre a plataforma *Eclipse RCP*. Esse protótipo, ilustrado na Figura 9.1, lê um arquivo em formato XML e exibe seu conteúdo em tela, permitindo a navegação pelo usuário e a extração de mais dados a partir do que se visualiza.

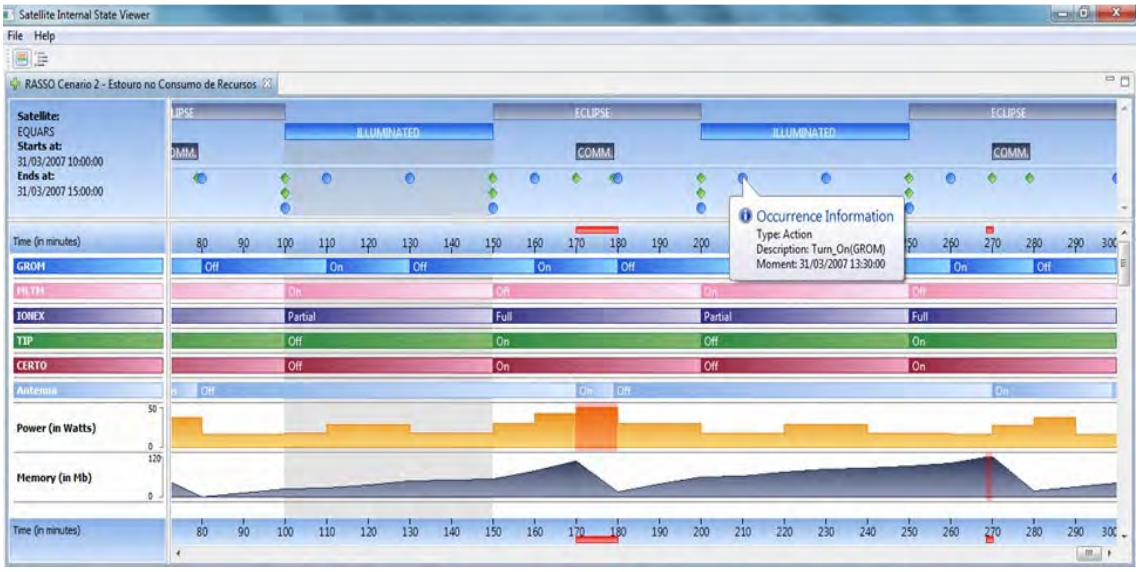


Figura 9.1 - Protótipo para um visualizador de timelines

9.4.Trabalhos Publicados e Submetidos para Publicação

As ideias iniciais para esta Tese, como uma evolução do protótipo RASSO, foram publicadas e apresentadas no *Symposium on Applied Computing da Association for Computing Machinery* (ACM – KUCINSKIS e FERREIRA, 2008a).

A proposta para um serviço de inferência como o núcleo de um sistema de planejamento embarcado foi tema de um artigo para a edição de 2008 da *International Conference on Space Operations* (SpaceOps) do *American Institute of Aeronautics and Astronautics* (AIAA – KUCINSKIS e FERREIRA, 2008b).

Outro artigo apresentado na edição seguinte do SpaceOps tratou de conceitos de autonomia aplicados aos padrões de engenharia adotados pelo INPE, e da integração do serviço de inferência com tais padrões (KUCINSKIS e FERREIRA, 2010).

Esse artigo foi selecionado, entre mais de 300 trabalhos de agências, instituições e empresas da área espacial, como um dos melhores da Conferência, sendo então publicado na forma de um capítulo de livro (KUCINSKIS e FERREIRA, 2011). Cabe registrar que o mesmo já havia ocorrido com o trabalho que precedeu a presente tese, na edição de 2006 da Conferência (KUCINSKIS e FERREIRA, 2007).

Considerações iniciais sobre a validação de sistemas autônomos, tendo o ISIS e modelos embarcados como foco, foram publicadas e apresentadas no quinto *Latin-American Symposium on Dependable Computing*, organizado e realizado pelo INPE (KUCINSKIS, 2011).

Um artigo descrevendo a arquitetura completa e o estudo de caso para a missão Amazonia-1 foi apresentado e publicado na edição deste ano do SpaceOps (KUCINSKIS e FERREIRA, 2012).

Um artigo em inglês foi submetido para o *IEEE Aerospace and Electronic Systems Magazine*. Dois artigos em português, com focos em diferentes aspectos do trabalho,

foram submetidos para a Revista IEEE América Latina e para a Revista de Informática Teórica e Aplicada da Universidade Federal do Rio Grande do Sul.

REFERÊNCIAS BIBLIOGRÁFICAS

ALAMI, R.; CHATILA, R.; FLEURY, S.; GHALLAB, M.; INGRAND, F. An architecture for autonomy. **The International Journal of Robotics Research**, v. 17, n. 4, p. 315-337, 1998.

ALLEN, J. Maintaining knowledge about temporal intervals. **Communications of the ACM**, v. 26, p. 832-843, 1983.

ARIAS, R.; KUCINSKIS, F. N.; ALONSO, J. D. D. Lessons learned from an onboard ECSS PUS object-oriented implementation. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 10., 2008, Heidelberg, Alemanha. **Proceedings...** Heidelberg: AIAA, 2008.

ASCHWANDEN, P.; BASKARAN, V.; BERNARDINI, S.; FRY, C.; MORENO, M.; MUSCETTOLA, N.; PLAUNT, C.; RIJSMAN, D.; TOMPKINS, P. Model-unified planning and execution for distributed autonomous system control. In: WORKSHOP ON SPACECRAFT AUTONOMY: USING AI TO EXPAND HUMAN SPACE EXPLORATION, AAAI FALL SYMPOSIUM, 2006, Washington DC, USA. **Proceedings...** Washington DC: AAAI, 2006.

ASNER, G. P.; KNOX, R. G.; GREEN, R. O.; UNGAR, S. G. **The Flora mission for ecosystem composition, disturbance and productivity.** 2005. Disponível em http://pages.csam.montclair.edu/~chopping/rs/FLORA_NRCDecadalSurvey_2005.pdf. Acesso em 15 dez 2010.

BEAUMET, G.; VERFAILLIE, G.; CHARMEAU, M. C. Feasibility of autonomous decision making on-board an agile Earth-observing satellite. **Computational Intelligence**, v. 27, n. 1, p. 123-139, 2011.

BENNETT, M.; DVORAK, D.; HUTCHERSON, J.; INGHAM, M.; RASMUSSEN, R.; WAGNER, D. An architectural pattern for goal-based control. In: IEEE AEROSPACE CONFERENCE, 2008, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2008.

BERNARD, D.; DORAIS, G.; GAMBLE, E.; KANEFSKY, B.; KURIEN, J.; MAN, G. K.; MILLAR, W.; MUSCETTOLA, N.; NAYAK, P.; RAJAN, K.; ROUQUETTE, N.; SMITH, B.; TAYLOR, W.; TUNG, Y. Spacecraft autonomy flight experience: the DS1 Remote Agent Experiment. In: SPACE TECHNOLOGY CONFERENCE AND EXPOSITION, 1999, Albuquerque, NM, USA. **Proceedings...** Albuquerque: AIAA, 1999.

BORNSTEIN, B.; ESTLIN, T.; GAINES, D.; THOMPSON, D. R.; GRANVILLE, C.; BURL, M.; CASTAÑO, R.; ANDERSON, R. C.; JUDD, M.; CHIEN, S. **Engineering AEGIS autonomous science targeting for the Mars Exploration Rovers.** 10 dez 2010. WORKSHOP ON SPACECRAFT FLIGHT SOFTWARE. Pasadena, CA, USA, 2010.

BRAMAN, J. B.; WAGNER, D. Energy management of the Multi-Mission Space Exploration Vehicle using a goal-oriented control system. In: IEEE AEROSPACE

CONFERENCE, 2011, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2011.

BRAT, G.; DENNEY, E.; FARELL, K.; GIANNAKOPOULOU, D.; JONSSON, A.; FRANK, J.; BODDY, M.; CARPENTER, T.; ESTLIN, T.; PIVTORAIKO, M. A robust compositional architecture for autonomous systems. In: IEEE AEROSPACE CONFERENCE, 2006, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2006.

CASTAÑO, A.; FUKUNAGA, A.; BIESIADECKI, J.; NEAKRASE, L.; WHELLEY, P.; GREEDELY, R.; LEMMON, M.; CASTAÑO, R.; CHIEN, S. Autonomous detection of dust devils and clouds on Mars. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 2006, Atlanta, GA, USA. **Proceedings...** Atlanta: AIAA, 2006.

CASTRO, A. P. A.; SILVA, J. D. S.; MEDEIROS, F. L. L.; SHIGUEMORI, E. H. Restauração de imagens e detecção automática de características aplicados à navegação aérea autônoma. In: SIMPÓSIO BRASILEIRO DE SENSORIAMENTO REMOTO, 14., 2009, Natal, Brasil. **Proceedings...** Natal, Brasil: INPE, 2009. P. 6813-6819.

CEBALLOS, A.; BENSALEM, S.; CESTA, A.; SILVA, L.; FRATINI, S.; INGRAND, F.; OCÓN, J.; ORLANDINI, A.; RAJAN, K.; RASCONI, R.; VAN WINNENDAEL, M. A goal-oriented autonomous controller for space exploration. In: SYMPOSIUM ON ADVANCED SPACE TECHNOLOGIES IN ROBOTICS AND AUTOMATION, 11., 2011, Noordwijk, Holanda. **Proceedings...** Noordwijk: ECSS, 2011.

CHANTHERY, E.; BARBIER, M.; FARGES, J. L. Integration of mission planning and flight scheduling for unmanned aerial vehicles. In: CASTILLO, L.; BORRAJO, D.; SALIDO, A.; ODDI, A. **Planning, Scheduling and Constraint Satisfaction:** from theory to practice. IOS Press, 2005, p. 109-118. Frontiers in Artificial Intelligence and Applications Series.

CHARMEAU, M. C.; BENSANA, E. AGATA, a lab bench project for spacecraft autonomy. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE ROBOTICS AND AUTOMATION IN SPACE (I-SAIRAS), 8., 2005, Hollywood, USA. **Proceedings...** Hollywood: European Space Agency, 2005.

CHIEN, S.; SHERWOOD, R.; TRAN, D.; CASTANO, R.; CICHY, B.; DAVIES, A.; RABIDEAU, G.; TANG, N.; BURL, M.; MANDL, D.; FRYE, S.; HENGEMIHLE, J.; D'AGOSTINO, J.; BOTE, R.; TROUT, B.; SHULMAN, S.; UNGAR, S.; VAN GAASBECK, J.; BOYER, D.; GRIFFIN, M.; BURKE, H.; GREELEY, R.; DOGGETT, T.; WILLIAMS, K.; BAKER, V.; DOHM, J. Autonomous science on the EO-1 mission. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE ROBOTICS AND AUTOMATION IN SPACE (I-SAIRAS), 7., 2003, Nara, Japão. **Proceedings...** Nara: JAXA, 2003.

CHIEN, S.; TRAN, D.; RABIDEAU, G.; SCHAFFER, S.; MANDL, D.; FRYE, S. Improving the operations of the Earth Observing One mission via automated mission planning. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 11., 2010, Huntsville, AL, USA. **Proceedings...** Huntsville: AIAA, 2010.

CHOUINARD, C. M.; FISHER, F.; GAINES, D. M.; ESTLIN, T. A.; SCHAFFER, S. R. An approach to autonomous operations for remote mobile robotic exploration. In: IEEE

AEROSPACE CONFERENCE, 2003, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2003.

DAS, J.; PY, F.; MAUGHAN, T.; O'REILLY, T.; MESSIÉ, M.; RYAN, J.; SUKHATME, G. S.; RAJAN, K. Coordinated sampling of dynamic oceanographic features with underwater vehicles and drifters. **International Journal of Robotics Research**, v. 5, n. 31, p. 626–646, 2012.

DECHTER, R. **Constraint processing**. 1. ed. Waltham: Morgan Kaufmann Publishers, 2003. 481 p. ISBN-13 (978-1558608900). The Morgan Kaufmann Series in Artificial Intelligence.

DVORAK, D. L.; RASMUSSEN, R.; REEVES, G.; SACKS, A. Software architecture themes in JPL's Mission Data System. In: IEEE AEROSPACE CONFERENCE, 2000, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2000.

DVORAK. D. L.; INGHAM, M. D.; MORRIS, R. M.; GERSH, J. Goal-based operations: an overview. In: AIAA INFOTECH@AEROSPACE 2007 CONFERENCE AND EXHIBIT. 2007, Rohnert Park, CA, USA. **Proceedings...** Rohnert: AIAA, 2007.

DVORAK, D. L.; AMADOR A. V.; STARBIRD, T. W. Comparison of goal-based operations and command sequencing. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 10., 2008, Heidelberg, Alemanha. **Proceedings...** Heidelberg: AIAA, 2008.

ESTLIN, T.; CASTAÑO, R.; GAINES, D.; BORNSTEIN, B.; JUDD, M.; ANDERSON, R. C.; NESNAS, I. Autonomous science technologies for a Mars rover. In: NASA SCIENCE TECHNOLOGY CONFERENCE, 2007, Adelphi, MD, USA. **Proceedings...** Adelphi: UMUC, 2007.

ESTLIN, T.; CASTAÑO, R.; GAINES, D.; BORNSTEIN, B.; JUDD, M.; ANDERSON, R. C. Enabling autonomous science for a Mars rover. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 10., 2008, Heidelberg, Alemanha. **Proceedings...** Heidelberg: AIAA, 2008.

ESTLIN, T.; BORNSTEIN, B.; GAINES, D.; THOMPSON, D. R.; CASTAÑO, R.; ANDERSON, R. C.; GRANVILLE, C.; BURL, M.; JUDD, M.; CHIEN, S. AEGIS automated targeting for the MER Opportunity rover. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE ROBOTICS AND AUTOMATION IN SPACE (I-SAIRAS), 10., 2010, Sapporo, Japão. **Proceedings...** Sapporo: JAXA, 2010.

EUROPEAN COMMITTEE FOR SPACE STANDARDIZATION (ECSS). **Ground systems and operations** – telemetry and telecommand packet utilization. Noordwijk, Holanda: ECSS, 2003. (ECSS-E-70-41A).

EUROPEAN COMMITTEE FOR SPACE STANDARDIZATION (ECSS). **Space segment operability.** Noordwijk, Holanda: ECSS, 2008. (ECSS-E-ST-70-11C).

EUROPEAN COMMITTEE FOR SPACE STANDARDIZATION (ECSS). **On-board control procedures**. Noordwijk, Holanda: ECSS, 2010. (ECSS-E-ST-70-01C).

FOX, M.; LONG, D. PDDL2.1: An extension of PDDL for expressing temporal planning

domains. **Journal of AI Research**, v.20, p. 61–124, 2003.

FRANK, J.; JÓNSSON, A. Constraint-based attribute and interval planning. **Journal of Constraints, Special Issue on Constraints and Planning,** v. 8, n. 4, 2003.

FRATINI, S.; CESTA, A.; BENEDICTIS, R.; ORLANDINI, A.; RASCONI, R. APSI-based deliberation in goal oriented autonomous controllers. In: SYMPOSIUM ON ADVANCED SPACE TECHNOLOGIES IN ROBOTICS AND AUTOMATION, 11., 2011, Noordwijk, Holanda. **Proceedings...** Noordwijk: ECSS, 2011.

FUKUNAGA, A.; RABIDEAU, G.; CHIEN, S.; YAN, D. Towards an application framework for automated planning and scheduling. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE ROBOTICS AND AUTOMATION IN SPACE (I-SAIRAS), 4., 1997, Tókio, Japão. **Proceedings...** Tókio: NASDA, 1997.

GAINES, D. M.; ESTLIN, T.; SCHAFFER, S.; CASTAÑO, R.; ELFES, A. Robust and opportunistic autonomous science for a potential Titan aerobot. In: AAS GEORGE H. BORN SYMPOSIUM, 2010, Boulder, CO, USA. **Proceedings...** Boulder: AAS, 2010.

INGHAM M. D.; RASMUSSEN, R. D.; BENNETT, M. B.; MONCADA, A. C. Engineering complex embedded systems with state analysis and the Mission Data System. **AIAA Journal of Aerospace Computing, Information and Communication (JACIC)**. Vol. 2, No. 12, p. 507-536, 2005.

INGRAND, F.; LACROIX, S.; LEMAI-CHENEVIER, S.; PY, F. Decisional autonomy of planetary rovers. **Journal of Field Robotics**, v. 24, n. 7, 2007.

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **Amazonia 1 mission requirements specification**. São José dos Campos, Brasil, 2010. Relatório técnico (A800000-SPC-001/01).

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **Recomendações de projeto e codificação em C++ para o software de voo do COMAV.** São José dos Campos, Brasil, 2011b. Relatório técnico (DEA-SB-001-2011-v1.1).

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **Goal-based Operations Enabling Software Architecture:** manual do desenvolvedor**.** São José dos Campos, Brasil, 2012. Relatório técnico (DEA-SB-001-2012-v1).

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE). **Plano diretor 2011-2015.** Disponível em http://www.inpe.br/noticias/arquivos/pdf/Plano_diretor_miolo.pdf. São José dos Campos, 2011a. Acesso em 14 jan 2012.

JET PROPULSION LABORATORY (JPL). **Under their own AEGIS.** 2009. Disponível em: http://scienceandtechnology.jpl.nasa.gov/newsandevents/newsdetails/?NewsID=677. Acesso em 27 fev 2011.

JÓNSSON, A.; MORRIS, P.; MUSCETTOLA, N.; RAJAN, K. Planning in interplanetary space: theory and practice. In: 2000 INTERNATIONAL CONFERENCE ON AI PLANNING AND SCHEDULING (AIPS), 2000, Breckenridge, CO, USA. **Proceedings...** [s.n.], 2000. p.

177-186.

JÓNSSON, A.; BRAT, G. Challenges in verification and validation of autonomous systems for space exploration. In: PERFORMANCE OF NEURO-ADAPTIVE AND LEARNING SYSTEMS: ASSESSMENT, MONITORING, AND VALIDATION (IJCNN), 2005, Montréal, Canadá. **Proceedings...** Montréal: 2005.

KNIGHT, R.; RABIDEAU, G.; CHIEN, S.; ENGELHARDT, B.; SHERWOOD, R. CASPER: space exploration through continuous planning. **IEEE Intelligent Systems**, v. 16, n. 4 , p. 70-75, 2001.

KUCINSKIS, F. N. **Alocação dinâmica de recursos computacionais para experimentos científicos com replanejamento automatizado a bordo de satélites.** 2007. 165 p. (INPE-14798-TDI/1241). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2007. Disponível em: <http://urlib.net/sid.inpe.br/mtc-m17@80/2007/04.23.11.42>. Acesso em 05 mar 2012.

KUCINSKIS, F. N.; FERREIRA, M. G. V., Dynamic allocation of resources to improve scientific return with onboard automated replanning. In: BRUCA, L.; DOUGLAS, J.; SORENSEN, T. **Space Operations:** mission management, technologies, and current applications. Reston, VA, USA: AIAA Press, 2007, v. 220, p. 345-359. Progress in Astronautics and Aeronautics Series.

KUCINSKIS, F. N.; FERREIRA, M. G. V. An onboard knowledge representation tool for satellite autonomous applications. In: ACM SYMPOSIUM ON APPLIED COMPUTING, 2008a, Fortaleza, Brasil. **Proceedings...** New York: ACM, 2008a.

KUCINSKIS, F. N.; FERREIRA, M. G. V. An Internal State Inference Service for onboard diagnosis, prognosis and contingency planning applications. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 10., 2008b, Heidelberg, Alemanha. **Proceedings...** Heidelberg: AIAA, 2008b.

KUCINSKIS, F. N.; FERREIRA, M. G. V Taking the ECSS autonomy concepts one step further. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 11., 2010, Huntsville, AL, USA. **Proceedings...** Huntsville: AIAA, 2010.

KUCINSKIS, F. N. Validation of reasoning and decision-making of a satellite autonomous on-board software. In: LATIN-AMERICAN SYMPOSYUM ON DEPENDABLE COMPUTING, (LADC), 5., 2011, São José dos Campos, Brasil. **Proceedings…** São José dos Campos, Brasil: IEEE, 2011. 1 CD.

KUCINSKIS, F. N.; FERREIRA, M. G. V. Taking the European Committee for Space Standardization autonomy concepts one step further. In: CRUZEN, C. A.; GUNN, J. M.; AMADIEU, P. J. **Space Operations:** exploration, scientific utilization, and technology development. Reston, VA, USA: AIAA Press, 2011, v. 236, p. 187-199. Progress in Astronautics and Aeronautics Series.

KUCINSKIS, F. N.; FERREIRA, M. G. V. Modeling the operation of satellite payloads for

on-board, goal-based planning. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 12., 2012, Stockholm, Suécia. **Proceedings...** Stockholm: AIAA, 2012.

LEMAI, S.; OLIVE, X.; CHARMEAU, M. C. Decisional architecture for autonomous space systems. In: ESA WORKSHOP ON ADVANCED SPACE TECHNOLOGIES FOR ROBOTICS AND AUTOMATION (ASTRA), 2006, Noordwijk, Holanda. **Proceedings...** Noordwijk: European Space Agency, 2006.

MARTINS, M. P.; MEDEIROS, F. L. L.; MONTEIRO, M. V.; SHIGUEMORI, E. H.; FERREIRA, L. C. A.; DOMICIANO, M. A. P. Navegação aérea autônoma por imagens. In: WORKSHOP ANUAL DE PESQUISA E DESENVOLVIMENTO DO INSTITUTO DE ESTUDOS AVANÇADOS, 6., 2006, São José dos Campos, Brasil. **Proceedings...** São José dos Campos: IEAv, 2006.

MUSCETTOLA, N.; PELL, B.; HANSSON, O.; MOHAN, S. Automating mission scheduling for space-based observatories. In: HENRY, G. W.; EATON, J.A. (eds.). **Robotic telescopes:** current capabilities, present developments, and future prospects for automated astronomy. Provo, UT, USA: Astronomical Society of the Pacific, 1995. ISBN: 0-937707-98-8; ISBN: 978-1-58381-415-4.

MUSCETTOLA, N.; DORAIS, G. A.; FRY, C.; LEVINSON, R.; PLAUNT, C. IDEA: planning at the core of autonomous reactive agents. In: 2002 INTERNATIONAL CONFERENCE ON AI PLANNING AND SCHEDULING (AIPS), 6., 2002, Toulouse, França. **Proceedings...** AIAA Press, 2002. p. 58-60.

NESNAS, I. A. D.; WRIGHT, A.; BAJRACHARYA, M.; SIMMONS, R.; ESTLIN, T.; KIM, W. S. CLARATy: an architecture for reusable robotic software. In: SPIE AEROSENSE CONFERENCE, 2003, Orlando, FL, USA. **Proceedings...** Orlando: SPIE, 2003.

PADUANO, J. **Onboard Planning System for UAVs in Support of Expeditionary Reconnaissance and Surveillance (OPS-USERS)**. 2006. Disponível em: http://www.virtualacquisitionshowcase.com/document/1654/briefing. Acesso em 10 mai 2012.

PELL, B.; BERNARD, D. E.; CHIEN, S. A.; GAT, E.; MUSCETTOLA, N.; NAYAK, P. P.; WAGNER, M. D.; WILLIAMS, B. C. A Remote Agent prototype for spacecraft autonomy. In: SPIE CONFERENCE ON OPTICAL SCIENCE, ENGINEERING, AND INSTRUMENTATION, 1996, San Diego, CA, USA. **Proceedings...** San Diego: SPIE, 1996.

POLLE, B. Autonomy requirements & technologies for future constellation. **Astrium Summary Report**, p. 1-30, 2002. (EAA.NT.BP.3682899.02).

RAJAN, K.; PY, F.; MCGANN, C.; RYAN, J.; O'REILLY, T.; MAUGHAN, T.; ROMAN, B. Onboard adaptive control of AUVs using automated planning and execution. In: INTERNATIONAL SYMPOSIUM ON UNMANNED UNTETHERED SUBMERSIBLE TECHNOLOGY (UUST), 2009. **Proceedings…** Durham, NH, USA.

RAJAN, K.; PY, F.; MCGANN, C. Adaptive control of AUVs using onboard planning and

execution. **Sea Technology Magazine**, April-May 2010. Disponível em http://sea-technology.com/features/2010/0410/adaptive_control_auvs.html. Acesso em 16 mai 2012.

ROTHSCHILD, R. ; HEINDL, W. ; MATTESON, J. ; REMILLARD, R. ; BRAGA, J. ; STAUBERT, R. ; KENDZIORRA, E. ; WILMS, J. ; HEISE, J. ; KUULKERS, E. . MIRAX - The Explosive X-Ray Transient Explorer. In: **Seventh Meeting of the High Energy Astrophysics Division of the American Astronomical Society**, 2003, Mt. Tremblant, Quebec, Canadá. Bulletin of the American Astronomical Society, 2003.

RUSSELL, S.; NORVIG, P. **Inteligência artificial**. Tradução da 2. ed., São Paulo: Editora Campus, 2004. 1040 p. ISBN (8535211772).

SHERWOOD, R.; CHIEN, S.; TRAN, D.; CICHY, B.; CASTAÑO, R.; DAVIES, A.; RABIDEAU, G. Operating the Autonomous Sciencecraft Experiment. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 8., 2004, Montréal, Canadá. **Proceedings...** Montreal: AIAA, 2004.

SIMMONS, R.; APFELBAUM, D. A task description language for robot control. In: CONFERENCE ON INTELLIGENT ROBOTICS AND SYSTEMS, 1998, Vancouver, Canada. **Proceedings...** [s.n.], 1998.

SMITH, D. E.; FRANK, J.; JÓNSSON, A. K. Bridging the gap between planning and scheduling. **Knowledge Engineering Review**, v. 1, n. 15, p. 47-83, 2000.

TRAN, D.; CHIEN, S.; RABIDEAU, G.; CICHY, B. Flight software issues in onboard automated planning - lessons learned on EO-1. In: INTERNATIONAL WORKSHOP ON PLANNING AND SCHEDULING FOR SPACE (IWPSS). 2004, Darmstadt, Alemanha. **Proceedings...** Darmstadt: STCI, 2004.

TRUSZKOWSKI, W., HALLOCK, H., ROUFF, C., KARLIN, J., RASH, J., HINCHEY, M., STERRITT, R. **Autonomous and autonomic systems:** with applications to NASA intelligent spacecraft operations and exploration systems. 1. ed. New York, NY, USA: Springer-Verlag New York, Inc. 2009. 306 p. ISBN-13 (978-1846282324).

VERFAILLIE, G.; PRALET, C.; LEMAÎTRE, M. Constraint-based modeling of discrete event dynamic systems. **Journal of Intelligent Manufacturing, Special Issue on Planning, Scheduling, and Constraint Satisfaction**. Springer Science+Business Media, 2008. Publicado on-line. DOI 10.1007/s10845-008-0176-3.

VERMA, V.; JÓNSSON, A.; ESTLIN, T.; LEVINSON, R. Survey of command execution systems for NASA spacecraft and robots. In: INTERNATIONAL CONFERENCE ON AUTOMATED PLANNING AND SCHEDULING (ICAPS), 2005, Monterey, CA, USA. **Proceedings...** Monterey: AIAA, 2005a. p. 92-99.

VERMA, V.; ESTLIN, T.; JÓNSSON, A.; PASSAREANU, C.; SIMMONS, R.; TSO, K. Plan Execution Interchange Language (PLEXIL) for executable plans and command sequences. In: INTERNATIONAL SYMPOSIUM ON ARTIFICIAL INTELLIGENCE ROBOTICS AND AUTOMATION IN SPACE (I-SAIRAS), 8., 2005, Hollywood, USA. **Proceedings...**

Hollywood: European Space Agency, 2005b.

VILAIN, M. B.; KAUTZ, H. Constraint propagation algorithms for temporal reasoning. In: AIAA CONFERENCE, 5., 1986, Philadelphia, PA. **Proceedings…** Philadelphia: AIAA, 1986. P. 377-382.

VOLPE, R.; NESNAS, I.; ESTLIN, T.; MUTZ, D.; PETRAS, R.; DAS, H. The CLARAty architecture for robotic autonomy. In: IEEE AEROSPACE CONFERENCE, 2001, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2001.

WAGNER, D.; DVORAK, D.; BAROFF, L.; MISHKIN, A.; INGHAM, M.; BENNETT, M.; MITTMAN, D. A control architecture for safe human-robotic interactions during lunar surface operations. In: AIAA INFOTECH@AEROSPACE, 2009, Seatle, WA, USA. **Proceedings...** Seatle: AIAA, 2009.

WAGNER, D.; DVORAK, D.; MISHKIN, A.; HORVATH, G.; JOHNSON, G.; JONES, G. Architectural concepts for human-rated automation. In: AIAA INFOTECH@AEROSPACE, 2010, Atlanta, GA, USA. **Proceedings...** Atlanta: AIAA, 2010.

WEGENER, S. S.; SHOENUNG, S. M.; TOTAH, J.; SULLIVAN, D.; FRANK, J.; ENOMOTO, F.; FROST, C.; THEODORE, C. UAV autonomous operations for airborne science missions. In: UNMANNED... UNLIMITED TECHNICAL CONFERENCE, WORKSHOP, AND EXHIBIT, 2004, Chicago, IL, USA. **Proceedings...** Chicago: AIAA, 2004.

WERTZ, J. R.; REINERT, R. P. Mission Characterization. In: LARSON, W. J.; WERTZ, J. R. (eds.). **Space mission analysis and design**. 3. ed. Torrance, CA, USA: Microcosm, Inc. and Kluwer Academic Publishers, 1999. 969 p. ISBN-13 (978-1881883104).

WILLIAMS, B. C.; NAYAK, P. P.; NAYAK, U. A model-based approach to reactive self-configuring systems. In: AAAI NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 13., 1996, Portland, Oergon. **Proceedings...** Portland: AAAI, 1996.

WOODS, M. J.; BALDWIN, L.; WILSON, G.; HALL, S.; PIDGEON, A.; LONG, D.; FOX, M.; AYLETT, R.; VITULI, R. MMOPS: assessing the impact of on-board autonomy for planetary exploration missions. In: INTERNATIONAL CONFERENCE ON SPACE OPERATIONS (SPACEOPS), 9., 2006, Roma, Itália. **Proceedings...** Roma: AIAA, 2006.

WOODS, M.; SHAW, A.; BARNES, D.; PRICE, D.; LONG, D.; PULLAN, D. Autonomous science for an ExoMars rover-like mission. **Journal of Field Robotics**, v. 26, n. 4, p. 358-390, 2009.

WU, P. P. Y.; CAMPBELL, D.; MERZ, T. On-board multi-objective mission planning for unmanned aerial vehicles. In: IEEE AEROSPACE CONFERENCE, 2009, Big Sky, MT, USA. **Proceedings...** Big Sky: IEEE, 2009.

APÊNDICE A – O MODELO DESENVOLVIDO PARA O ESTUDO DE CASO DA MISSÃO AMAZONIA-1

A.1 A Descrição Estrutural

Conteúdo do arquivo ‘structure.isis’:

```
<!-- ISIS Structural Description for the Amazonia-1 domain, created by ISIS
modeling framework on [data aqui] -->
<structural_description>
<domains>
    <domain name = "AwfiModes">
        <value>awfi_power_off</value>
        <value>awfi_stand_by</value>
        <value>awfi_imaging</value>
    </domain>
    <domain name = "DdrModes">
        <value>ddr_power_off</value>
        <value>ddr_starting_up</value>
        <value>ddr_stand_by</value>
        <value>ddr_record</value>
        <value>ddr_playback</value>
        <value>ddr_erase</value>
    </domain>
    <domain name = "DssStatus">
        <value>real_time</value>
        <value>playback</value>
    </domain>
    <domain name = "DtModes">
        <value>dt_power_off</value>
        <value>dt_stand_by</value>
        <value>dt_nominal</value>
    </domain>
    <domain name = "OrbitPeriods">
        <value>sunlight</value>
        <value>eclipse</value>
    </domain>
    <domain name = "SadaStatus">
        <value>moving</value>
        <value>stopped</value>
    </domain>
    <domain name = "OnOff">
        <value>on</value>
        <value>off</value>
    </domain>
</domains>
<elements>
    <element name = "AWFI">
        <timeline name = "Mode" type = "AwfiModes" hkparameterid = "9" />
    </element>
    <element name = "DDR">
        <timeline name = "Mode" type = "DdrModes" hkparameterid = "10" />
        <timeline name = "ReadyToOperate" type = "bool"
                  hkparameterid = "41" />
        <timeline name = "MemoryFull" type = "bool" hkparameterid = "11" />
        <timeline name = "DssState" type = "DssStatus"
                  hkparameterid = "12" />
        <resource name = "Memory" type = "uint32" quantity = "160000"
                  consumptionperrate = "true">
            <consumer name = "DDR"
                      initialtotalconsumptionhkparameterid = "22"
```

```
                    initialrateconsumptionhkparameterid = "23"/>
        </resource>
        <resource name = "DataToDownload" type = "uint32"
                  quantity = "160000" consumptionperrate = "true">
            <consumer name = "DDR"
                    initialtotalconsumptionhkparameterid = "42"
                    initialrateconsumptionhkparameterid = "43"/>
        </resource>
    </element>
    <element name = "DT">
        <timeline name = "Mode" type = "DtModes" hkparameterid = "15" />
        <timeline name = "CommunicatingWithGround" type = "bool"
                  hkparameterid = "16" />
    </element>
    <element name = "PSS">
        <timeline name = "OrbitPeriod" type = "OrbitPeriods"
                  hkparameterid = "17" />
        <timeline name = "SadaState" type = "SadaStatus"
                  hkparameterid = "18" />
        <timeline name = "AwfiLineStatus" type = "OnOff"
                  hkparameterid = "19" />
        <timeline name = "DdrLineStatus" type = "OnOff"
                  hkparameterid = "20" />
        <timeline name = "DtLineStatus" type = "OnOff"
                  hkparameterid = "21" />
        <resource name = "Power" type = "uint16" quantity = "350"
                  consumptionperrate = "false">
            <conditional_constraint condition = "PSS.OrbitPeriod = eclipse"
                                    maxValue = "300" />
            <consumer name = "AWFI"
                    initialtotalconsumptionhkparameterid = "24"/>
            <consumer name = "DDR"
                    initialtotalconsumptionhkparameterid = "25"/>
            <consumer name = "DT"
                    initialtotalconsumptionhkparameterid = "26"/>
        </resource>
    </element>
</elements>
</structural_description>
```

A.2 A Descrição Comportamental em ISISml

A.2.1 Conteúdo do Arquivo 'awfi_actions.isis'

```
Action TurnOnDcDcConverters
{
    Preconditions
    {
        AWFI.Mode = awfi_stand_by;
        PSS.SadaState = sada_stopped;
        PSS.OrbitPeriod = sunlight;
        DDR.Mode = ddr_record or DDR.DssState = real_time;
    }

    Effects
    {
        AWFI.Mode = awfi_imaging;
        AWFI.Consume (PSS.Power, 170);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 1;
        // se nao tiver data field, basta nao coloca-lo aqui
        DataField = 0A9B8C7D6E5F4;
```

```
    }
}

Action TurnOffDcDcConverters
{
    Preconditions
    {
        // esta precondicao eh redundante, mas necessaria
        // para que o planejador seja mais efetivo
        PSS.AwfiLineStatus = on;
        AWFI.Mode = awfi_imaging;
    }

    Effects
    {
        AWFI.Mode = awfi_stand_by;
        AWFI.Consume (PSS.Power, 56);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 2;
        DataField = 0A9B8C7D6E5F4;
    }
}
```

A.2.2 Conteúdo do Arquivo 'ddr_actions.isis'

```
Action TurnOnDcDcConverter
{
    Preconditions
    {
        PSS.DdrLineStatus = on;
        DDR.Mode = ddr_power_off;
    }

    Effects
    {
        // ficara 15 seg nesse modo; um evento o colocara em stand-by
        DDR.Mode = ddr_starting_up;
        DDR.Consume (PSS.Power, 23);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 3;
    }
}

Action TurnOffDcDcConverter
{
    Preconditions
    {
        PSS.DdrLineStatus = on;
    }

    Effects
    {
        DDR.Mode = ddr_power_off; // modo: de qualquer um para off
        DDR.ReadyToOperate = false;
        DDR.CeaseConsumptionOrRelease (PSS.Power);
```

```
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 4;
    }
}

// unifica playback from beginning e from breakpoint (simplificacao)
Action Playback
{
    Preconditions
    {
        DT.Mode = dt_nominal;
        DDR.Mode = ddr_stand_by;
        DDR.ReadyToOperate = true;
        DDR.DssState = dss_playback;
    }

    Effects
    {
        DDR.Mode = ddr_playback;
        DDR.Consume (PSS.Power, 28);
        DDR.Release (DDR.DataToDownload, 128 per_second)
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 5;
    }
}

Action EraseFile
{
    Preconditions
    {
        DDR.Mode = ddr_stand_by;
        DDR.ReadyToOperate = true;
    }

    Effects
    {
        DDR.Mode = ddr_erase;
        DDR.Release (DDR.Memory, 128 per_second);
        // nao libero DataToDownload para que o planejador nao escolha
        // apagar dados ao inves de transferi-los
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 6;
        DataField = 0A9B8C7D6E5F4;
    }
}

Action StopPlayback
{
    Preconditions
    {
        DDR.Mode = ddr_playback;
        DDR.ReadyToOperate = true;
```

```
    }

    Effects
    {
        DDR.Mode = ddr_stand_by;
        DDR.Consume (PSS.Power, 23);
        DDR.CeaseConsumptionOrRelease (DDR.DataToDownload);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 7;
    }
}

Action StartRecording
{
    Preconditions
    {
        DDR.Mode = ddr_stand_by;
        DDR.ReadyToOperate = true;
        DDR.MemoryFull = false;
        DDR.DssState = dss_real_time;
    }

    Effects
    {
        DDR.Mode = ddr_record;

        // consome memoria a taxa de 128Mbps (TBC), ate que nao haja
        // mais memoria disponivel
        DDR.Consume (DDR.Memory, 128 per_second);
        DDR.Consume (DDR.DataToDownload, 128 per_second);
        DDR.Consume (PSS.Power, 28);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 8;
    }
}

Action StopRecording
{
    Preconditions
    {
        DDR.Mode = ddr_record;
        DDR.ReadyToOperate = true;
    }

    Effects
    {
        DDR.Mode = ddr_stand_by;
        DDR.CeaseConsumptionOrRelease (DDR.Memory);
        DDR.Consume (PSS.Power, 23);
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 9;
        DataField = 0A9B8C7D6E5F4;
```

```
    }
}

Action AwfiChannelPlayback
{
    Preconditions
    {
        DDR.DssState = dss_real_time;
    }

    Effects
    {
        DDR.DssState = dss_playback;
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 10;
    }
}

Action AwfiChannelRealTime
{
    Preconditions
    {
        DDR.DssState = dss_playback;
    }

    Effects
    {
        DDR.DssState = dss_real_time;
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 11;
    }
}
```

A.2.3 Conteúdo do Arquivo ‘dt_actions.isis’

```
Action DtAwfiChannelTurnOn
{
    Preconditions
    {
        DT.Mode = dt_stand_by;
        DT.CommunicatingWithGround = true;
    }

    Effects
    {
        DT.Mode = dt_nominal;
        DT.Consume (PSS.Power, 121);

        // passa a transmitir dados do DDR a taxa de 128 MBPs - isso
        // nao tem efeito na modelagem, porque quem limpa a memoria
        // eh o proprio DDR
    }

    ServiceRequest
    {
```

```
        ServiceType = 134;
        ServiceSubtype = 12;
    }
}

Action DtAwfiChannelTurnOff
{
    Preconditions
    {
        DT.Mode = dt_nominal;
    }

    Effects
    {
        DT.Mode = dt_stand_by;
        DT.Consume (PSS.Power, 107);

        // para de transmitir dados do DDR - isso nao tem efeito na
        // modelagem, porque quem limpa a memoria eh o proprio DDR
    }

    ServiceRequest
    {
        ServiceType = 134;
        ServiceSubtype = 13;
    }
}
```

A.2.4 Conteúdo do Arquivo 'pss_actions.isis'

```
Action SwitchOnAwfiLine
{
    Preconditions
    {
        PSS.AwfiLineStatus = off;
    }

    Effects
    {
        PSS.AwfiLineStatus = on;
        AWFI.Mode = awfi_stand_by;
        AWFI.Consume (PSS.Power, 56);
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 1;
    }
}

Action SwitchOffAwfiLine
{
    Preconditions
    {
        PSS.AwfiLineStatus = on;
    }

    Effects
    {
        PSS.AwfiLineStatus = off;
        AWFI.Mode = awfi_power_off;
        AWFI.CeaseConsumptionOrRelease (PSS.Power);
```

```
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 2;
    }
}

Action SwitchOnDdrLine
{
    Preconditions
    {
        PSS.DdrLineStatus = off;
    }

    Effects
    {
        PSS.DdrLineStatus = on;
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 3;
    }
}

Action SwitchOffDdrLine
{
    Preconditions
    {
        PSS.DdrLineStatus = on;
    }

    Effects
    {
        PSS.DdrLineStatus = off;
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 4;
    }
}

Action SwitchOnDtLine
{
    Preconditions
    {
        PSS.DtLineStatus = off;
    }

    Effects
    {
        PSS.DtLineStatus = on;
        DT.Mode = dt_stand_by;
        DT.Consume (PSS.Power, 107);
    }

    ServiceRequest
    {
        ServiceType = 131;
```

```
        ServiceSubtype = 5;
    }
}

Action SwitchOffDtLine
{
    Preconditions
    {
        PSS.DtLineStatus = on;
    }

    Effects
    {
        PSS.DtLineStatus = off;
        DT.Mode = dt_power_off;
        DT.CeaseConsumptionOrRelease (PSS.Power);
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 6;
    }
}

Action StopSada
{
    Preconditions
    {
        PSS.SadaState = sada_moving;
    }

    Effects
    {
        PSS.SadaState = sada_stopped;
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 7;
    }
}

Action ResumeSada
{
    Preconditions
    {
        PSS.SadaState = sada_stopped;
    }

    Effects
    {
        PSS.SadaState = sada_moving;
    }

    ServiceRequest
    {
        ServiceType = 131;
        ServiceSubtype = 8;
    }
}
```

A.2.5 Conteúdo do Arquivo ‘events.isis’

```
ExogenousEvent EnterInEclipse
{
    PSS.OrbitPeriod = eclipse;
}

ExogenousEvent ExitEclipse
{
    PSS.OrbitPeriod = sunlight;
}

ExogenousEvent StartCommunication
{
    DT.CommunicatingWithGround = true;
}

ExogenousEvent FinishCommunication
{
    DT.CommunicatingWithGround = false;
}

EndogenousEvent MemoryFull
{
    RaisesWhen
    {
        DDR.TotalConsumption >= 160000;
    }

    Effects
    {
        // necessario para interromper o preenchimento de memoria
        DDR.CeaseConsumptionOrRelease(DDR.Memory);

        DDR.MemoryFull = true;

        // com a memoria cheia, o DDR volta a stand_by
        DDR.Mode = ddr_stand_by;
        DDR.Consume (PSS.Power, 23);
    }
}

EndogenousEvent MemoryEmpty
{
    RaisesWhen
    {
        DDR.TotalConsumption <= 0;
    }

    Effects
    {
        // necessario para interromper a liberacao de memoria
        DDR.CeaseConsumptionOrRelease(DDR.Memory);

        // com a memoria vazia, o DDR volta a stand-by
        DDR.Mode = ddr_stand_by;
        DDR.Consume (PSS.Power, 23);
    }
}

EndogenousEvent DdrReady
{
    Raises15secondsAfter
```

```
    {
        // Apos ligado, o DDR fica 15 segundos executando
        // auto-teste e formatacao.
        // Isso nao impede a execucao de comandos ao DSS.
        DDR.Mode = ddr_starting_up;
    }

    Effects
    {
        // o consumo se mantem o mesmo do modo anterior
        DDR.Mode = ddr_stand_by;
        DDR.ReadyToOperate = true;
    }
}
```

A.3 O Modelo Embarcado (Código-Fonte Compilável)

A.3.1 Conteúdo do Arquivo 'domains.h'

```
/**
 * @file   domains.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Enumeracoes utilizadas pelo modelo.
 **/

#ifndef DOMAINS_H_
#define DOMAINS_H_

namespace Isis
{

enum ModelElements
{
    element_awfi = 0,
    element_ddr  = 1,
    element_dt   = 2,
    element_pss  = 3,
    number_of_model_elements
};

enum AwfiModes
{
    awfi_power_off = 0,
    awfi_stand_by  = 1,
    awfi_imaging   = 2
};

enum DdrModes
{
    ddr_power_off   = 0,
    ddr_starting_up = 1,
    ddr_stand_by    = 2,
    ddr_record      = 3,
    ddr_playback    = 4,
    ddr_erase  = 5
};

enum DssStatus
{
    dss_real_time = 0,
    dss_playback  = 1
};

enum DtModes
{
    dt_power_off = 0,
```

```
    dt_stand_by  = 1,
    dt_nominal   = 2
};

enum OrbitPeriods
{
    sunlight = 0,
    eclipse  = 1
};

enum SadaStatus
{
    sada_moving  = 0,
    sada_stopped = 1
};

enum OnOff
{
    off = 0,
    on = 1
};

enum ActionIds
{
    no_action = 0,
    awfi_action_turnondcdcconverters = 1,
    awfi_action_turnoffdcdcconverters = 2,
    ddr_action_turnondcdcconverter = 3,
    ddr_action_turnoffdcdcconverter = 4,
    ddr_action_playback = 5,
    ddr_action_erasefile = 6,
    ddr_action_stopplayback = 7,
    ddr_action_startrecording = 8,
    ddr_action_stoprecording = 9,
    ddr_action_awfichannelplayback = 10,
    ddr_action_awfichannelrealtime = 11,
    dt_action_dtawfichannelturnon = 12,
    dt_action_dtawfichannelturnoff = 13,
    pss_action_switchonawfiline = 14,
    pss_action_switchoffawfiline = 15,
    pss_action_switchonddrline = 16,
    pss_action_switchoffddrline = 17,
    pss_action_switchondtline = 18,
    pss_action_switchoffdtline = 19,
    pss_action_stopsada = 20,
    pss_action_resumesada = 21,
    number_of_action_ids_plus_one // usado pelo sistema
};

enum EventIds
{
    no_event = 0,
    event_entereclipse = 1,
    event_exiteclipse = 2,
    event_startcommunication = 3,
    event_finishcommunication = 4,
    event_memoryfull = 5,
    event_memoryempty = 6,
    event_ddrready = 7,
    number_of_event_ids_plus_one
};

enum TimelineIds
{
    no_timeline = 0,
    awfi_timeline_mode = 1,
    ddr_timeline_mode = 2,
    ddr_timeline_memoryfull = 3,
    ddr_timeline_dssstate = 4,
    ddr_timeline_readytooperate = 5,
    dt_timeline_mode = 6,
```

```
    dt_timeline_communicatingwithground = 7,
    pss_timeline_orbitperiod = 8,
    pss_timeline_sadastate = 9,
    pss_timeline_awfilinestatus = 10,
    pss_timeline_ddrlinestatus = 11,
    pss_timeline_dtlinestatus = 12,
    number_of_timeline_ids_plus_one // usado pelo sistema
};

enum ResourceIds
{
    no_resource = 0,
    ddr_resource_memory = 1,
    ddr_resource_datatodownload = 2,
    pss_resource_power = 3,
    number_of_resource_ids_plus_one // usado pelo sistema
};
} // namespace Isis

#endif /* DOMAINS_H_ */
```

A.3.2 Conteúdo do Arquivo 'AWFI.h'

```
/**
 * @file   AWFI.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Definicao da classe para o elemento AWFI do modelo ISISml.
 **/

#ifndef AWFI_H_
#define AWFI_H_

#include "../isis/metamodel/Types.h"
#include "../isis/metamodel/timelines/Timeline.h"
#include "../isis/metamodel/occurrences/Action.h"

namespace Isis
{

/**
 * @class AWFI
 * @brief Classe que implementa o elemento AWFI do modelo ISISml.
 **/
class AWFI
{
public:
    AWFI();
    ~AWFI();

    static Timeline Mode;

    static Action TurnOnDcDcConverters();
    static Action TurnOffDcDcConverters();
};
} // namespace Isis

#endif /* AWFI_H_ */
```

A.3.3 Conteúdo do Arquivo 'AWFI.cpp'

```
/**
 * @file   AWFI.cpp
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Implementacao da classe para o elemento AWFI do modelo ISISml.
 **/
```

```
#include "AWFI.h"
#include "PSS.h"
#include "DDR.h"
#include "domains.h"

namespace Isis
{

AWFI::AWFI()
{
}

AWFI::~AWFI()
{
}

Action AWFI::TurnOnDcDcConverters()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(AWFI::Mode.GetCurrent() == static_cast<int32_t>(awfi_stand_by)))
    {
        returnStatus.RegisterFalsePrecondition(awfi_timeline_mode);
    }

    if (!(PSS::SadaState.GetCurrent() == static_cast<int32_t>(sada_stopped)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_sadastate);
    }

    if (!(PSS::OrbitPeriod.GetCurrent() == static_cast<int32_t>(sunlight)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_orbitperiod);
    }

    if (!((DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_record)) ||
          (DDR::DssState.GetCurrent()==static_cast<int32_t>(dss_real_time))))
    {
        // TODO: como nao da para saber (devido a estrutura atual) qual dos
        // timelines que participam da verificacao tem seu valor diferente do
        // esperado (podem inclusive ser ambos), devemos registrar todos
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
        returnStatus.RegisterFalsePrecondition(ddr_timeline_dssstate);
    }

    // Effects ******************************************
    AWFI::Mode.SetCurrent(static_cast<int32_t>(awfi_imaging));

    // awfi passa a consumir 170 W (assumindo para a modelagem que os heaters
    // estarao ligados)
    PSS::Power.Consume(static_cast<uint32_t>(element_awfi), 170);

    // o consumo de memoria depende da configuracao do ddr; portanto, isso eh
    // registrado em DDR
    return returnStatus;
}

Action AWFI::TurnOffDcDcConverters()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(PSS::AwfiLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        // esta precondicao eh redundante, mas necessaria para que
        // o planejador seja mais efetivo
        returnStatus.RegisterFalsePrecondition(pss_timeline_awfilinestatus);
    }

    if (!(AWFI::Mode.GetCurrent() == static_cast<int32_t>(awfi_imaging)))
```

```
    {
        returnStatus.RegisterFalsePrecondition(awfi_timeline_mode);
    }

    // Effects ****************************************
    AWFI::Mode.SetCurrent(static_cast<int32_t>(awfi_stand_by));

    // awfi passa a consumir 56W (assumindo para a modelagem que os heaters
    // estarao ligados)
    PSS::Power.Consume(static_cast<uint32_t>(element_awfi), 56);

    // o consumo de memoria depende da configuracao do ddr; portanto, isso eh
    // registrado em DDR

    return returnStatus;
}
} // namespace Isis
```

A.3.4 Conteúdo do Arquivo 'DDR.h'

```
/**
 * @file   DDR.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Definicao da classe para o elemento DDR do modelo ISISml.
 **/

#ifndef DDR_H_
#define DDR_H_

#include "../isis/metamodel/timelines/Timeline.h"
#include "../isis/metamodel/Types.h"
#include "../isis/metamodel/resources/Resource.h"
#include "../isis/metamodel/occurrences/Action.h"

namespace Isis
{

/**
 * @class DDR
 * @brief Classe que implementa o elemento DDR do modelo ISISml.
 **/
class DDR
{
public:
    DDR();
    ~DDR();

    static Timeline Mode;
    static Timeline MemoryFull;
    static Timeline DssState;
    static Timeline ReadyToOperate;

    // capacidade: 160 Gbits de dados (desconsiderando 40 Gbits
    // de edac), consumiveis em pouco mais de 20 minutos
    static Resource Memory;
    static Resource DataToDownload;

    static Action TurnOnDcDcConverter();
    static Action TurnOffDcDcConverter();
    static Action Playback();
    static Action EraseFile();
    static Action StopPlayback();
    static Action StartRecording();
    static Action StopRecording();
    static Action AwfiChannelPlayback();
    static Action AwfiChannelRealTime();

    static void VerifyMemoryResourceEndogenousEventsConditions();
    static void VerifyDdrModeEndogenousEventsConditions();
```

```
};
} // namespace Isis

#endif /* DDR_H_ */
```

A.3.5 Conteúdo do Arquivo ‘DDR.cpp’

```
/**
 * @file   DDR.cpp
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Implementacao da classe para o elemento DDR do modelo ISISml.
 **/

#include "DDR.h"
#include "PSS.h"
#include "DT.h"
#include "domains.h"

namespace Isis
{

DDR::DDR()
{
}

DDR::~DDR()
{
}

Action DDR::TurnOnDcDcConverter()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(PSS::DdrLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_ddrlinestatus);
    }

    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_power_off)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    // Effects ******************************************
    // ficara 15 segundos nesse modo; um evento o colocara em stand-by
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_starting_up));

    // consumo de energia passa a ser de 23W
    PSS::Power.Consume(static_cast<uint32_t>(element_ddr), 23);

    return returnStatus;
}

Action DDR::TurnOffDcDcConverter()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(PSS::DdrLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_ddrlinestatus);
    }

    // Effects ******************************************
    // modo: de qualquer um para off
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_power_off));
    DDR::ReadyToOperate.SetCurrent(false);
```

```
    // para de consumir energia
    PSS::Power.CeaseConsumptionOrRelease(static_cast<int32_t>(element_ddr));

    return returnStatus;
}

// unifica playback from beginning e from breakpoint (simplificacao)
Action DDR::Playback()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DT::Mode.GetCurrent() == static_cast<int32_t>(dt_nominal)))
    {
        returnStatus.RegisterFalsePrecondition(dt_timeline_mode);
    }

    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_stand_by)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    if (!(DDR::DssState.GetCurrent() == static_cast<int32_t>(dss_playback)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_dssstate);
    }

    if (!(DDR::ReadyToOperate.GetCurrent() == static_cast<int32_t>(true)))
    {
        // a timeline readytooperate foi criada para que o planejador
        // seja mais efetivo ao escolher acoes
        returnStatus.RegisterFalsePrecondition(ddr_timeline_readytooperate);
    }

    // Effects ******************************************
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_playback));
    DDR::DataToDownload.Release(static_cast<uint32_t>(element_ddr), 128
                                                          PER_SECOND);

    // consumo de energia passa a ser de 28W
    PSS::Power.Consume(static_cast<uint32_t>(element_ddr), 28);

    return returnStatus;
}

Action DDR::EraseFile()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_stand_by)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    if (!(DDR::ReadyToOperate.GetCurrent() == static_cast<int32_t>(true)))
    {
        // a timeline readytooperate foi criada para que o planejador
        // seja mais efetivo ao escolher acoes
        returnStatus.RegisterFalsePrecondition(ddr_timeline_readytooperate);
    }

    // Effects ******************************************
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_erase));

    DDR::Memory.Release(static_cast<uint32_t>(element_ddr), 128 PER_SECOND);

    // nao libero DataToDownload para que o planejador nao escolha apagar
    // dados ao inves de transferi-los

    // ira voltar ao modo stand_by ao liberar toda a mem.; um evento faz isso
```

```
    // o consumo de energia em stand-by e erase eh o mesmo, nao precisa mudar

    return returnStatus;
}

Action DDR::StopPlayback()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_playback)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    if (!(DDR::ReadyToOperate.GetCurrent() == static_cast<int32_t>(true)))
    {
        // a timeline readytooperate foi criada para que o planejador
        // seja mais efetivo ao escolher acoes
        returnStatus.RegisterFalsePrecondition(ddr_timeline_readytooperate);
    }

    // Effects ******************************************
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_stand_by));
    DDR::DataToDownload.CeaseConsumptionOrRelease(static_cast<uint32_t>
                                                  (element_ddr));

    // consumo de energia passa a ser de 23W
    PSS::Power.Consume(static_cast<uint32_t>(element_ddr), 23);

    return returnStatus;
}

Action DDR::StartRecording()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_stand_by)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    if (!(DDR::MemoryFull.GetCurrent() == static_cast<int32_t>(false)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_memoryfull);
    }

    if (!(DDR::DssState.GetCurrent() == static_cast<int32_t>(dss_real_time)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_dssstate);
    }

    if (!(DDR::ReadyToOperate.GetCurrent() == static_cast<int32_t>(true)))
    {
        // a timeline readytooperate foi criada para que o planejador
        // seja mais efetivo ao escolher acoes
        returnStatus.RegisterFalsePrecondition(ddr_timeline_readytooperate);
    }

    // Effects ******************************************
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_record));

    // consome memoria a taxa de 128Mbps, ate que nao haja mais memoria
    // disponivel. Quando isso ocorrer, volta ao modo stand_by e ao consumo
    // anterior; um evento faz isso
    DDR::Memory.Consume(static_cast<uint32_t>(element_ddr), 128 PER_SECOND);
    DDR::DataToDownload.Consume(static_cast<uint32_t>(element_ddr), 128
                                                      PER_SECOND);

    // consumo de energia passa a ser de 28W
```

```
    PSS::Power.Consume(static_cast<uint32_t>(element_ddr), 28);

    return returnStatus;
}

Action DDR::StopRecording()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::Mode.GetCurrent() == static_cast<int32_t>(ddr_record)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_mode);
    }

    if (!(DDR::ReadyToOperate.GetCurrent() == static_cast<int32_t>(true)))
    {
        // a timeline readytooperate foi criada para que o planejador
        // seja mais efetivo ao escolher acoes
        returnStatus.RegisterFalsePrecondition(ddr_timeline_readytooperate);
    }

    // Effects ******************************************
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_stand_by));

    // interrompe o consumo de memoria
    DDR::Memory.CeaseConsumptionOrRelease(static_cast<uint32_t>
                                                    (element_ddr));
    DDR::DataToDownload.CeaseConsumptionOrRelease(static_cast<uint32_t>
                                                    (element_ddr));

    // consumo de energia passa a ser de 23W
    PSS::Power.Consume(static_cast<uint32_t>(element_ddr), 23);

    return returnStatus;
}

Action DDR::AwfiChannelPlayback()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::DssState.GetCurrent() == static_cast<int32_t>(dss_real_time)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_dssstate);
    }

    // Effects ******************************************
    DDR::DssState.SetCurrent(static_cast<int32_t>(dss_playback));

    return returnStatus;
}

Action DDR::AwfiChannelRealTime()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DDR::DssState.GetCurrent() == static_cast<int32_t>(dss_playback)))
    {
        returnStatus.RegisterFalsePrecondition(ddr_timeline_dssstate);
    }

    // Effects ******************************************
    DDR::DssState.SetCurrent(static_cast<int32_t>(dss_real_time));

    return returnStatus;
}

// os metodos de verificaca de eventos endogenos inseridos pelo
// parser da ISISml nao se encontram transcritos aqui
```

```
} // namespace Isis
```

A.3.6 Conteúdo do Arquivo ‘DT.h’

```
/**
 * @file   DT.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Definicao da classe para o elemento DT do modelo ISISml.
 **/

#ifndef DT_H_
#define DT_H_

#include "../isis/metamodel/timelines/Timeline.h"
#include "../isis/metamodel/Types.h"
#include "../isis/metamodel/occurrences/Action.h"

namespace Isis
{

/**
 * @class DT
 * @brief Classe que implementa o elemento DT do modelo ISISml.
 **/
class DT
{
public:
    DT();
    ~DT();

    static Timeline Mode;
    static Timeline CommunicatingWithGround;

    static Action DtAwfiChannelTurnOn();
    static Action DtAwfiChannelTurnOff();
};
} // namespace Isis

#endif /* DT_H_ */
```

A.3.7 Conteúdo do Arquivo ‘DT.cpp’

```
/**
 * @file   DT.cpp
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Implementacao da classe para o elemento DT do modelo ISISml.
 **/

#include "DT.h"
#include "PSS.h"
#include "domains.h"

namespace Isis
{

DT::DT()
{
}

DT::~DT()
{
}

Action DT::DtAwfiChannelTurnOn()
{
```

```
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DT::Mode.GetCurrent() == static_cast<int32_t>(dt_stand_by)))
    {
        returnStatus.RegisterFalsePrecondition(dt_timeline_mode);
    }

    if (!(DT::CommunicatingWithGround.GetCurrent() ==
                                                  static_cast<int32_t>(true)))
    {
        returnStatus.RegisterFalsePrecondition
                                      (dt_timeline_communicatingwithground);
    }

    // Effects ******************************************
    DT::Mode.SetCurrent(static_cast<int32_t>(dt_nominal));

    // dt passa a consumir 121 Watts
    PSS::Power.Consume(static_cast<uint32_t>(element_dt), 121);

    // passa a transmitir dados do DDR a taxa de 128 MBPs - isso nao tem
    // efeito na modelagem, porque quem limpa a memoria eh o proprio DDR

    return returnStatus;
}

Action DT::DtAwfiChannelTurnOff()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(DT::Mode.GetCurrent() == static_cast<int32_t>(dt_nominal)))
    {
        returnStatus.RegisterFalsePrecondition(dt_timeline_mode);
    }

    // Effects ******************************************
    DT::Mode.SetCurrent(static_cast<int32_t>(dt_stand_by));

    // dt passa a consumir 107 Watts
    PSS::Power.Consume(static_cast<uint32_t>(element_dt), 107);

    // para de transmitir dados do DDR - isso nao tem efeito na modelagem,
    // porque quem limpa a memoria eh o proprio DDR

    return returnStatus;
}
} // namespace Isis
```

A.3.8 Conteúdo do Arquivo 'PSS.h'

```
/**
 * @file   PSS.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Definicao da classe para o elemento PSS do modelo ISISml.
 **/

#ifndef PSS_H_
#define PSS_H_

#include "../isis/metamodel/timelines/Timeline.h"
#include "../isis/metamodel/Types.h"
#include "../isis/metamodel/resources/Resource.h"
#include "../isis/metamodel/occurrences/Action.h"

namespace Isis
{
```

```
/**
 * @class PSS
 * @brief Classe que implementa o elemento PSS do modelo ISISml.
 **/
class PSS
{
public:
    PSS();
    ~PSS();

    static Timeline OrbitPeriod;
    static Timeline SadaState;
    static Timeline AwfiLineStatus;
    static Timeline DdrLineStatus;
    static Timeline DtLineStatus;

    static Resource Power;

    static Action SwitchOnAwfiLine();
    static Action SwitchOffAwfiLine();
    static Action SwitchOnDdrLine();
    static Action SwitchOffDdrLine();
    static Action SwitchOnDtLine();
    static Action SwitchOffDtLine();
    static Action StopSada();
    static Action ResumeSada();

    static bool VerifyPowerConditionalConstraints (const uint32_t
                                                   &currentTotalConsumption);
};
} // namespace Isis

#endif /* PSS_H_ */
```

A.3.9 Conteúdo do Arquivo 'PSS.cpp'

```
/**
 * @file   PSS.cpp
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Implementacao da classe para o elemento PSS do modelo ISISml.
 **/

#include "PSS.h"
#include "AWFI.h"
#include "DT.h"
#include "DDR.h"
#include "domains.h"

namespace Isis
{

PSS::PSS()
{
}

PSS::~PSS()
{
}

Action PSS::SwitchOnAwfiLine()
{
    Action returnStatus;

    // Pre-conditions ***********************************
    if (!(PSS::AwfiLineStatus.GetCurrent() == static_cast<int32_t>(off)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_awfilinestatus);
    }
```

```
    // Effects ****************************************
    PSS::AwfiLineStatus.SetCurrent(static_cast<int32_t>(on));
    AWFI::Mode.SetCurrent(static_cast<int32_t>(awfi_stand_by));

    // awfi passa a consumir 56 Watts de energia
    PSS::Power.Consume(static_cast<uint32_t>(element_awfi), 56);

    return returnStatus;
}

Action PSS::SwitchOffAwfiLine()
{
    Action returnStatus;

    // Pre-conditions *********************************
    if (!(PSS::AwfiLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_awfilinestatus);
    }

    // Effects ****************************************
    PSS::AwfiLineStatus.SetCurrent(static_cast<int32_t>(off));
    AWFI::Mode.SetCurrent(static_cast<int32_t>(awfi_power_off));

    // awfi para de consumir energia
    PSS::Power.CeaseConsumptionOrRelease(static_cast<uint32_t>(element_awfi));

    return returnStatus;
}

Action PSS::SwitchOnDdrLine()
{
    Action returnStatus;

    // Pre-conditions *********************************
    if (!(PSS::DdrLineStatus.GetCurrent() == static_cast<int32_t>(off)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_ddrlinestatus);
    }

    // Effects ****************************************
    PSS::DdrLineStatus.SetCurrent(static_cast<int32_t>(on));

    return returnStatus;
}

Action PSS::SwitchOffDdrLine()
{
    Action returnStatus;

    // Pre-conditions *********************************
    if (!(PSS::DdrLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_ddrlinestatus);
    }

    // Effects ****************************************
    PSS::DdrLineStatus.SetCurrent(static_cast<int32_t>(off));
    DDR::Mode.SetCurrent(static_cast<int32_t>(ddr_power_off));

    return returnStatus;
}

Action PSS::SwitchOnDtLine()
{
    Action returnStatus;

    // Pre-conditions *********************************
    if (!(PSS::DtLineStatus.GetCurrent() == static_cast<int32_t>(off)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_dtlinestatus);
```

```
    }

    // Effects ***************************************
    PSS::DtLineStatus.SetCurrent(static_cast<int32_t>(on));
    DT::Mode.SetCurrent(static_cast<int32_t>(dt_stand_by));

    // DT passa a consumir 107 Watts
    PSS::Power.Consume(static_cast<uint32_t>(element_dt), 107);

    return returnStatus;
}

Action PSS::SwitchOffDtLine()
{
    Action returnStatus;

    // Pre-conditions ********************************
    if (!(PSS::DtLineStatus.GetCurrent() == static_cast<int32_t>(on)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_dtlinestatus);
    }

    // Effects ***************************************
    PSS::DtLineStatus.SetCurrent(static_cast<int32_t>(off));
    DT::Mode.SetCurrent(static_cast<int32_t>(dt_power_off));

    // DT deixa de consumir power
    PSS::Power.CeaseConsumptionOrRelease(static_cast<uint32_t>(element_dt));

    return returnStatus;
}

Action PSS::StopSada()
{
    Action returnStatus;

    // Pre-conditions ********************************
    if (!(PSS::SadaState.GetCurrent() == static_cast<int32_t>(sada_moving)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_sadastate);
    }

    // Effects ***************************************
    PSS::SadaState.SetCurrent(sada_stopped);

    return returnStatus;
}

Action PSS::ResumeSada()
{
    Action returnStatus;

    // Pre-conditions ********************************
    if (!(PSS::SadaState.GetCurrent() == static_cast<int32_t>(sada_stopped)))
    {
        returnStatus.RegisterFalsePrecondition(pss_timeline_sadastate);
    }

    // Effects ***************************************
    PSS::SadaState.SetCurrent(static_cast<int32_t>(sada_moving));

    return returnStatus;
}


// os metodos de verificaca de restricoes condicionais impostas a recursos,
// inseridos pelo parser da ISISml, nao se encontram transcritos aqui

} // namespace Isis
```

A.3.10 Conteúdo do Arquivo ‘events.h’

```
/**
 * @file   events.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Externs para os eventos do modelo.
 **/

#ifndef EVENTS_H_
#define EVENTS_H_

#include "../isis/metamodel/Types.h"

namespace Isis
{
extern ExogenousEvent EnterEclipse();
extern ExogenousEvent ExitEclipse();
extern ExogenousEvent StartCommunication();
extern ExogenousEvent FinishCommunication();

extern EndogenousEvent MemoryFull();
extern EndogenousEvent MemoryEmpty();
extern EndogenousEvent DdrReady();
} // namespace Isis

#endif /* EVENTS_H_ */
```

A.3.11 Conteúdo do Arquivo ‘events.cpp’

```
/**
 * @file   events.h
 * @author ISISml Parser
 * @date   01/07/2011
 * @brief  Implementacao dos eventos do modelo.
 **/

#include "../isis/metamodel/Types.h"
#include "domains.h"
#include "PSS.h"
#include "DT.h"
#include "DDR.h"

namespace Isis
{
    ExogenousEvent EnterEclipse()
    {
        PSS::OrbitPeriod.SetCurrent(eclipse);
    }

    ExogenousEvent ExitEclipse()
    {
        PSS::OrbitPeriod.SetCurrent(sunlight);
    }

    ExogenousEvent StartCommunication()
    {
        DT::CommunicatingWithGround.SetCurrent(true);
    }

    ExogenousEvent FinishCommunication()
    {
        DT::CommunicatingWithGround.SetCurrent(false);
    }

    EndogenousEvent MemoryFull()
    {
        DDR::MemoryFull.SetCurrent(true);
```

```
        // necessario para interromper o preenchimento de memoria
        DDR::Memory.CeaseConsumptionOrRelease(element_ddr);

        // com a memoria cheia, o DDR volta a stand_by
        DDR::Mode.SetCurrent(ddr_stand_by);
        PSS::Power.Consume(element_ddr, 23);
    }

    EndogenousEvent MemoryEmpty()
    {
        // necessario para interromper a liberacao de memoria
        DDR::Memory.CeaseConsumptionOrRelease(element_ddr);

        // com a memoria vazia, o DDR volta a stand-by
        DDR::Mode.SetCurrent(ddr_stand_by);
        PSS::Power.Consume(element_ddr, 23);
    }

    EndogenousEvent DdrReady()
    {
        DDR::Mode.SetCurrent(ddr_stand_by);
        DDR::ReadyToOperate.SetCurrent(true);
        // o consumo se mantem o mesmo do modo anterior
    }
}
```