Ministério da
Ciência e Tecnologia

INPE-16670-TDI/1621

UM ENFOQUE BASEADO EM CONHECIMENTO E DIRIGIDO A MODELOS DE ENGENHARIA DE REQUISITOS PARA PROJETO CONCEITUAL DE SATÉLITES

Bruno Bustamante Ferreira Leonor

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Stephan Stephany, e Walter Abrahão dos Santos, aprovada em 23 de fevereiro de 2010.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/01.27.12.06>

INPE
São José dos Campos
2010

PUBLICADO POR:

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

Ministério da
Ciência e Tecnologia

INPE-16670-TDI/1621

UM ENFOQUE BASEADO EM CONHECIMENTO E DIRIGIDO A MODELOS DE ENGENHARIA DE REQUISITOS PARA PROJETO CONCEITUAL DE SATÉLITES

Bruno Bustamante Ferreira Leonor

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada, orientada pelos Drs. Stephan Stephany, e Walter Abrahão dos Santos, aprovada em 23 de fevereiro de 2010.

Registro do documento original:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/01.27.12.06>

INPE
São José dos Campos
2010

Dados Internacionais de Catalogação na Publicação (CIP)

L494um

Leonor, Bruno Bustamante Ferreira.
Um enfoque baseado em conhecimento e dirigido a modelos de engenharia de requisitos para projeto conceitual de satélites / Bruno Bustamante Ferreira Leonor. – São José dos Campos : INPE, 2010.
xx + 91 p. ; (INPE-16670-TDI/1621)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2010.
Orientadores : Drs. Stephan Stephany, e Walter Abrahão dos Santos.

1. Engenharia de sistemas. 2. Engenharia de requisitos. 3. Sistemas baseados em conhecimento. 4. Engenharia dirigida por modelos. 5. Linguagem de modelagem de sistemas. I.Título.

CDU 004.414.23

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

Computação Aplicada

Dr. **Nandamudi Lankalapalli Vijaykumar**

Presidente / INPE / SJCampos - SP

Dr. **Stephan Stephany**

Orientador(a) / INPE / SJCampos - SP

Dr. **Walter Abrahão dos Santos**

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

Dr. **Mauricio Gonçalves Vieira Ferreira**

Membro da Banca / INPE / SJCampos - SP

Dra. **Emília Villani**

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

Aluno (a): Bruno Bustamante Ferreira Leonor

São José dos Campos, 23 de fevereiro de 2010

A minha família e a minha namorada.

AGRADECIMENTOS

A meus pais João e Carmen Cinira, a minha namorada Paula, a minha avó Anna Maria e a minha irmã Anna Luiza por todo apoio e compreensão durante este trabalho.

Aos meus orientadores Stephan Stephany e Walter Abrahão dos Santos pelo incentivo e conhecimento compartilhado, pelos quais sou muito grato.

Ao Instituto Nacional de Pesquisas Espaciais (INPE) pela infraestrutura e a oportunidade de realizar meus estudos.

Aos meus amigos do INPE, pelo companheirismo.

A todas as pessoas que, direta ou indiretamente, contribuíram para a realização desse trabalho durante esses anos.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo auxílio financeiro.

RESUMO

Satélites estão tornando-se cada vez mais complexos, tornando questões técnicas de projeto significativamente relevantes. A crescente complexidade destes sistemas torna a atividade de engenharia de requisitos mais importante e mais difícil. Tipicamente, existem especificações e restrições de projeto que impõem um alto nível de exigência para a engenharia de sistemas e, particularmente, para a engenharia de requisitos, que constitui uma fase importante do projeto desses sistemas. Caso não se atinja um nível adequado de conformidade, vários problemas de projeto podem ocorrer, tais como falhas, custos excedidos e atraso. A solução proposta é apoiar a fase de projeto conceitual de satélite com base no reuso e na integração da informação. Isto permite tratar melhor a natureza interdisciplinar do domínio, através da adoção de uma engenharia dirigida por modelo (MDE - *Model Driven Engineering*). Foi escolhida para implantar este enfoque a linguagem SysML (*Systems Modeling Language*). Neste contexto é proposto o desenvolvimento de uma ferramenta de software baseado em conhecimento, denominada SatBudgets, aplicada ao projeto do satélite universitário ITASAT, atualmente em desenvolvimento pelo INPE, ITA e algumas universidades brasileiras.

A KNOWLEDGE-BASED AND MODEL-DRIVEN REQUIREMENTS ENGINEERING APPROACH FOR CONCEPTUAL SATELLITE DESIGN

ABSTRACT

Satellite systems are becoming even more complex, making their technical issues even more significant. The increasing complexity of these systems makes activities related to requirements engineering more important and difficult. Tipically, the design specifications and constraints impose a heavy burden on systems-of-systems engineering and particularly on requirements engineering which is an important phase in a system's life cycle. When there is no conformance to these specifications and constraints, various design problems may occur, such as failures, cost overruns and delays. The proposed solution is to support the preliminary conceptual satellite design with computer-based information reuse and integration in order to deal with the interdisciplinary nature of the domain. This can be attained by taking a model-driven engineering approach (MDE) implemented in this work by means of SysML (Systems Modeling Language). In this context, a novel knowledge-based software tool, named SatBudgets, is proposed to support the conceptual phase of a university satellite called ITASAT, currently being developed by INPE, ITA, and some Brazilian universities.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

Pág.

SUMÁRIO

Pág.

LISTA DE ABREVIATURAS E SIGLAS

ACDH	–	*Attitude Control and On-Board Data Handling*
ADL	–	*Architecture Description Language*
AEB	–	Agência Espacial Brasileira
API	–	*Application Programming Interface*
BRMS	–	*Business Rules Management System*
DOM	–	*Document Object Model*
ESA	–	*European Space Agency*
GPS	–	*Global Positioning System*
IAE	–	Instituto de Atividades Espaciais
IDE	–	*Integrated Development Environment*
INCOSE	–	*International Council on Systems Engineering*
INPE	–	Instituto Nacional de Pesquisas Espaciais
ITA	–	Instituto Tecnológico de Aeronáutica
JDOM	–	*Java Document Object Model*
JPL	–	*Jet Propulsion Laboratory*
JVM	–	*Java Virtual Machine*
LHS	–	*Left Hand Side*
MDE	–	*Model-Driven Engineering*
OMG	–	*Object Management Group*
PCD	–	Plataforma de Coleta de Dados
RHS	–	*Right Hand Side*
SAX	–	*Simple API for XML*
SCD	–	Satélite de Coleta de Dados
SysML	–	*Systems Modeling Language*
SWT	–	*Standard Widget Toolkit*
UML	–	*Unified Modeling Language*
Unicamp	–	Universidade Estadual de Campinas
USP	–	Universidade de São Paulo
XMI	–	*XML Metadata Interchange*
XML	–	*Extensible Markup Language*

1 INTRODUÇÃO

Os sistemas modernos estão tornando-se cada vez mais complexos, por este motivo projetos conceituais estão ganhando cada vez mais espaço no desenvolvimento de um produto, podendo-se assim especificar, analisar, projetar e verificar sistemas sem a necessidade de produção de uma única peça. (LEONOR et al., 2009)

O Projeto Conceitual é a fase inicial do processo de projeto de um sistema e exige a aplicação de conhecimento no domínio do produto. Essa fase deve ser sistematizada, ou seja, o sistema deve ser decomposto em subsistemas e/ou componentes, para que se obtenha uma maior eficiência e velocidade no desenvolvimento. (ALMEIDA, 2000)

Devido à grande concorrência do mercado, a sistematização de sistemas é necessidade premente para os dias de hoje.

Para projeto de satélites isso não é diferente, levando em consideração a grande quantidade de recursos, tanto humanos quanto financeiros, envolvidos em sua produção e também os benefícios trazidos pelos mesmos à população. Um bom planejamento é essencial, pois evita erros que podem comprometer todo o empreendimento. (SANTOS et al., 2009)

A Engenharia de Sistemas Espaciais é o processo iterativo e disciplinado que emprega práticas aceitas de engenharia para planejar, definir, controlar, projetar, analisar, integrar, verificar e então apoiar a produção, testes e operação de produtos de *hardware*, *software*, programas ou sistemas espaciais. Esta atividade requer alta interação entre os engenheiros de sistemas uma vez que suas tarefas são vinculadas entre si e um *workflow* pode depender da saída de outros processos sobre responsabilidade de outros membros da equipe.

A falta de ferramentas adequadas para auxiliar nesta fase de projeto foi a principal motivação para a proposta deste trabalho, que tem como objetivo mostrar a viabilidade do desenvolvimento deste tipo de ferramental.

Um paradigma de projeto é adotado no Team-X do JPL (*Jet Propulsion Laboratory*) (SMITH et al., 2001) empregando metodologias colaborativas e concorrentes para projetar, analisar e avaliar completa e rapidamente os projetos conceituais

de missões espaciais. (MEIJERS et al., 2004) cita algumas práticas focadas em Engenharia de Requisitos de projeto de Satélites Universitários vinculados à ESA (*European Space Agency*), fazendo uso de ferramentas proprietárias.

No INPE (Instituto Nacional de Pesquisas Espaciais), a Engenharia de Sistemas Espaciais utiliza um conjunto de ferramental baseado em planilhas manualmente geradas além de um conjunto de aplicativos específicos para cada subsistema de satélites que não trabalham integradamente. Dados são trocados de maneira ainda manual e sequencial não explorando o caráter paralelo e distribuído de decisões de projeto conceitual.

Uma nova abordagem chamada de MDE (*Model-Driven Engineering*) (SCHMIDT, 2006) pode auxiliar na fase de projeto conceitual de sistemas permitindo o reuso e integração da informação. Metodologia esta não explorada nas abordagens citadas anteriormente. Esta abordagem foca a criação de modelos ou abstrações maximizando a compatibilidade entre sistemas e simplificando o processo de projeto. A utilização de uma metodologia MDE eficaz de trabalho envolve especialistas no domínio, plataformas, teste, verificação, validação, etc. Para isto, MDE requer o uso de uma linguagem de descrição para a arquitetura do sistema.

Para este trabalho em particular, foi adotada a linguagem SysML (*Systems Modeling Language*) (OMG, 2009a), uma linguagem proposta pela OMG (*Object Management Group*) para se tornar uma linguagem padrão na modelagem de sistemas que surgiu como complemento a UML (*Unified Modeling Language*) (OMG, 2009b), devido ao fato da UML ser específica para sistemas de *software* e não apoiar a modelagem de sistemas em geral (*software* e *hardware*). Assim como a MDE, a SysML também não é empregada para descrição de arquiteturas de satélites nas abordagens citadas.

Neste contexto, é apresentado neste trabalho um protótipo de software baseado em conhecimento, denominado SatBudgets. O software será aplicado na fase conceitual do projeto de um sistema de satélite universitário, o ITASAT, atualmente em desenvolvimento pelo INPE, ITA (Instituto Tecnológico de Aeronáutica) e algumas universidades brasileiras.

O conhecimento inserido na ferramenta foi adquirido minimamente através de captura de regras de decisão de projeto. Esta atividade faz parte do escopo de

Engenharia do Conhecimento, que integra o conhecimento do especialista a um ambiente computacional, unindo a aquisição do conhecimento explícito do especialista (coleta, seleção, decomposição, composição e modelagem) ao conhecimento implícito, existente em bases de dados relacionadas ao escopo deste especialista. Dentre as tarefas consideradas inteligentes estão por exemplo: previsões, reconhecimento de padrões, classificação, diagnóstico, capacidade de aprender com novos fatos, realizacão de inferências, realização de análises, tomadas de decisões e etc (SOUZA, 1999).

SatBudgets será aplicado para auxiliar na geração de balanços (mecânico, elétrico, entre outros) de um sistema de satélite. As regras aplicadas para a geração do balanço foram extraídas de especialistas em cada um dos subsistemas de satélite, podendo estas serem alteradas, removidas ou adicionadas sem a necessidade de parar a execução da ferramenta.

SatBudgets, bem como a abordagem MDE, poderá ser aplicado em outros projetos de satélites, assim como a idéia de implementação da ferramenta pode ser utilizada para a criação de ferramentas em outras áreas de pesquisa.

1.1 Objetivo do Trabalho

Este trabalho tem como objetivo desenvolver um protótipo de um software baseado em conhecimento para apoiar a fase de projeto conceitual de satélites. Este software utiliza a abordagem MDE permitindo o reuso e integração da informação e permite ao usuário gerar balanços automáticos, por exemplo de consumo de energia, de massa e telecomandos, para o modelo SysML referente a uma dada configuração de arquitetura de satélite. O estudo de caso escolhido foi o satélite universitário ITASAT.

1.2 Organização do Texto

Os capítulos deste trabalho estão organizados da seguinte forma.

Uma revisão bibliográfica é apresentada no Capítulo 2, onde são apresentados conceitos e definições básicas para o desenvolvimento dos próximos capítulos.

Algumas características do Satélite ITASAT, utilizado neste trabalho como estudo de caso, são apresentadas no Capítulo 3.

No Capítulo 4 é feita uma introdução à SysML, Linguagem de Descrição de Arquiteturas (ADL) escolhida para este trabalho.

O Capítulo 5 apresenta a ferramenta proposta neste trabalho, chamada de SatBudgets bem como as ferramentas utilizadas no seu desenvolvimento, implementação e funcionamento.

Finalmente, no Capítulo 6 são apresentadas as considerações finais e algumas propostas de tarefas futuras para este trabalho de Mestrado.

2 FUNDAMENTAÇÃO

O desenvolvimento de sistemas, como os da área espacial, é uma atividade complexa, pois o sucesso do projeto está relacionado, entre outros fatores, à competência da equipe e à forma de trabalho. Para aumentar a produtividade, bem como maximizar a compatibilidade entre os indivíduos e grupos envolvidos no projeto, existe uma metodologia chamada MDE (Engenharia Dirigida por Modelos) cuja ideia é usar modelos em diferentes níveis de abstração no desenvolvimento do sistema. Os modelos representados na MDE são especificações escritas em uma ADL, que representam o domínio do sistema e que permitem capturar detalhes de um programa/problema. Uma ADL pode ser uma linguagem de programação/modelagem, textual ou gráfica, criada para trabalhar em um determinado domínio (MEDVIDOVIC; TAYLOR, 2000).

2.1 Projeto Conceitual e Engenharia de Requisitos

O Projeto Conceitual é a fase inicial do processo de projeto de um sistema e exige a aplicação de conhecimento de domínio do problema. Essa fase deve ser sistematizada para que seja possibilitada sua integração com as demais fases do projeto do produto, bem como a integração do processo global de projeto com as demais fases de produção de um produto, o planejamento de fabricação e a fabricação propriamente dita (ALMEIDA, 2000).

A fase de projeto exige grande capacidade inventiva e de adaptação do projetista. Adicionalmente, exige do projetista/engenheiro o conhecimento de várias disciplinas diferentes, como apresentado na figura 2.1. Isto envolve várias linhas de raciocínio e uma grande quantidade de informações inter-disciplinares. Dessa forma, é visível a grande complexidade de tal fase do processo de projeto e suas interdependências.

Vale realçar a grande importância da fase inicial de projeto no custo e no sucesso do sistema final. Decisões tomadas nessa fase apresentam grande importância e alto custo para serem alteradas nas fases posteriores do desenvolvimento de um sistema.

Ao automatizar o processo de Projeto Conceitual, obtém-se uma maior eficiência e velocidade de desenvolvimento que é necessidade essencial no mercado competitivo de hoje em dia.

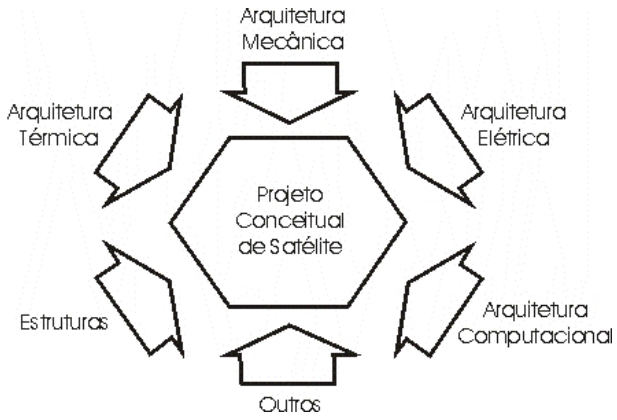


Figura 2.1 - Interdisciplinaridade no Projeto Conceitual de Satélites

Neste contexto, a representação dos requisitos de projeto é um conceito chave para a prática da engenharia de sistemas e visa descrever em modelos conceituais as condições ou capacidades do sistema. O objetivo principal desta representação é propiciar uma melhor compreensão do domínio do problema e uma precisa definição do domínio da solução. A representação dos requisitos é estabelecida em um nível conceitual de modelagem. Os modelos produzidos, textuais ou em notação gráfica, descrevem os aspectos relevantes do domínio do problema e relatam os componentes e comportamento esperado para o sistema. A etapa de definição de requisitos está incluída na fase da engenharia de requisitos e é usada para descrever as atividades relacionadas à produção (levantamento, registro, validação e verificação) e gerência (controle de mudanças, gerência de configuração, rastreabilidade, gerência de qualidade de requisitos) (SPíNOLA; ÁVILA, 2007).

Existem diferentes definições encontradas na literatura técnica para requisitos (SPíNOLA; ÁVILA, 2007):

- um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir os seus objetivos.
- as descrições das funções e restrições são os requisitos do sistema.
- um requisito é uma propriedade que o sistema deve exibir para resolver algum problema no mundo real.
- uma condição ou uma capacidade que deve ser alcançada ou estar pre-

sente em um sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto.

Requisitos são importantes para se: (1) estabelecer uma base de concordância entre o cliente e o fornecedor sobre o que o sistema fará, (2) fornecer uma referência para a validação do produto final e (3) reduzir o custo de desenvolvimento, pois requisitos mal definidos causam retrabalhos.

Os requisitos de um sistema podem ser classificados como (FERNANDES, 2005):

- **Requisitos funcionais**: são requisitos diretamente ligados à funcionalidade do sistema, descrevem as funções que o sistema deve executar (um sistema comercial deve permitir cadastro de clientes).
- **Requisitos não funcionais**: são requisitos que expressam condições que o sistema deve atender ou qualidades específicas que o sistema deve ter. Em vez de informar o que o sistema fará, os requisitos não-funcionais colocam restrições no sistema (um sistema de agendamento de consultas médicas deve garantir que o tempo de resposta não seja maior do que 5 segundos).
- **Requisitos de domínio**: são requisitos derivados do domínio da aplicação e descrevem características do sistema e qualidades que refletem o domínio. Podem ser requisitos funcionais novos, restrições sobre requisitos existentes ou computações específicas (um sistema acadêmico deve calcular a média final de cada aluno por uma regra específica determinando se foi aprovado ou não).

Alguns dos principais objetivos da engenharia de requisitos são (SPíNOLA; ÁVILA, 2007):

- estabelecer uma visão comum entre o cliente e a equipe de projeto em relação aos requisitos que serão atendidos pelo projeto de sistema;
- registrar e acompanhar requisitos ao longo de todo o processo de desenvolvimento;

- documentar e controlar os requisitos alocados para estabelecer uma *baseline* para uso gerencial e da engenharia de sistema, e

- manter planos, artefatos e atividades de sistema consistentes com os requisitos alocados.

A engenharia de requisitos é considerada um dos mais importantes conjuntos de atividades a serem realizados em projetos de desenvolvimento de sistema. Embora não garanta a qualidade dos produtos gerados, é um pré-requisito básico para o sucesso no desenvolvimento do projeto.

2.2 Engenharia Dirigida por Modelos (MDE)

Um esforço proeminente iniciado nos anos 80 focava o desenvolvimento de métodos e ferramentas de software que possibilitava desenvolvedores expressarem seus projetos em representações gráficas, como uma máquina de estado, diagrama de estruturas e diagrama de fluxo de dados (SCHMIDT, 2006).

O OMG (Grupo de Gerenciamento de Objetos, do inglês *Object Management Group*) propôs um sistema padronizado e público de modelos e interfaces que são independentes de linguagem, sistemas ou protocolo. O OMG é formado a partir da associação de empresas preocupadas com a interoperabilidade das aplicações comerciais, constantemente afetadas pela heterogeneidade e pela própria evolução dos produtos relacionados com a tecnologia da informação.

A MDE é uma abordagem para desenvolvimento de sistemas que foca a criação de modelos ou abstrações maximizando a compatibilidade entre sistemas, simplificando o processo de projeto e promovendo comunicação entre indivíduos e grupos trabalhando no sistema e que pode ser mapeado em qualquer plataforma.

De acordo com (SCHMIDT, 2006), a metodologia MDE tem oferecido uma promissora aproximação entre a falta de habilidade das linguagens de terceira geração e a complexidade das plataformas em expressar efetivamente conceitos de domínio.

Como consequência da implantação da MDE, o processo de desenvolvimento de sistemas torna-se iterativo, refinando modelos de abstração para modelos mais concretos (FONDEMENT; SILAGHI, 2004).

Uma das mais importantes vantagens de uso do MDE é a adaptabilidade a mudanças. Quando uma mudança ocorre, seja no mais alto ou baixo nível de abstração, seu impacto é bem localizado e as partes que não são alteradas são imediatamente reutilizáveis.

A utilização eficaz de uma metodologia MDE envolve especialistas no domínio, plataformas, teste, verificação, validação, etc. Se bem utilizada, a MDE pode reduzir significativamente o retrabalho dos desenvolvedores de sistemas.

O modelo do sistema pode ser descrito como uma abstração de um sistema físico distinguindo o que é relevante do que não é com o objetivo de simplificar a realidade.

A MDE representa as visões do sistema desde a sua modelagem até a implementação, tornando-se uma solução viável para resolver problemas de complexidade de desenvolvimento, manutenção e evolução do sistema.

Porém, a complexidade obtida pelo uso de diferentes soluções pelos engenheiros pode acarretar perda de qualidade e de reusabilidade. Visando aumentar a reusabilidade e a qualidade, modelos de sistema podem ser mantidos em diferentes níveis de abstração, a fim de se obter novos modelos em plataformas específicas.

Cada modelo representa um conjunto limitado de informações, por isso um sistema é normalmente obtido através de vários modelos. Um modelo não significa apenas modelos gráficos ou descritos em uma linguagem, mas sim uma descrição de um sistema a partir da especificação de uma perspectiva particular a um nível conceitual que o torne fácil de perceber.

2.3 Linguagem de Descrição de Arquiteturas (ADL)

Uma ADL é uma linguagem formal usada para descrever *software*, *hardware* ou componentes de um sistema e as interfaces entre os componentes (MEDVIDOVIC; TAYLOR, 2000). Em uma visão macro, uma arquitetura especifica como os componentes são combinados em subsistemas, como eles interagem e como eles são atribuidos aos componentes de hardware.

As ADLs são usadas para especificar estruturas genéricas e uma de suas vanta-

gens é possibilitar o uso da especificação na simulação da arquitetura em questão. O resultado da simulação pode ser usado para melhorar e validar a especificação (SOUZA et al., 2004), com o objetivo de encontrar a melhor arquitetura para a aplicação, economizando tempo e reduzindo os custos.

As ADLs concentram-se na representação das partes e são projetadas para descrever sistemas genéricos. Entretanto, um grande número de ADLs para um domínio específico ou de uso geral vem sendo desenvolvidas por grupos acadêmicos ou industriais. Muitas destas linguagens não se destinavam originalmente a serem ADLs, mas revelaram-se adequadas para representar e analisar uma arquitetura.

Uma ADL atende a um mínimo de funcionalidades, que são:

- **Composição**: possibilita a divisão hierárquica de um sistema complexo em partes menores;
- **Abstração**: permite explicitar a estrutura de mais alto nível;
- **Reusabilidade**: suporta o uso de componentes, conectores e padrões de arquitetura;
- **Configuração**: separa a descrição de estruturas compostas da descrição dos elementos dessas composições;
- **Análise**: permite verificar propriedades dos sistemas, especialmente referentes a Arquiteturas Dinâmicas;
- **Heterogeneidade**: possibilita combinar diferentes padrões arquiteturais em um mesmo sistema, e
- **Comunicação**: possui uma arquitetura adequada para integrar todos os grupos envolvidos no projeto.

Genericamente, os blocos que podem ser construídos em uma descrição da arquitetura são:

- Componentes;
- Conectores, e

- Topologia (Configuração de Arquitetura).

A descrição de uma arquitetura pode ser realizada de maneira formal. O produto da descrição arquitetural de um sistema é a sua configuração, ou seja, a estrutura topológica da aplicação. Na configuração, estão definidos os pontos de interação de cada módulo e cada conector bem como a maneira como os componentes irão interagir entre si.

ADLs podem também focar diferentes domínios de aplicação, estilos de arquiteturas, ou aspectos do modelo da arquitetura. Uma série de ADLs tem sido propostas para modelar arquiteturas tanto dentro de um determinado domínio como para fins gerais.

Neste trabalho, foram pesquisadas algumas ADLs mais comuns como: Aesop (GARLAN, 1995), C2 (MEDVIDOVIC et al., 1996), Darwin (MAGEE; KRAMER, 1996), MetaH (VESTAL, 1996), Rapide (LUCKHAM et al., 1995), SADL (RIEMENSCHNEIDER; MORICONI, 1997), Unicon (SHAW et al., 1996), Weaves (GORLICK; RAZOUK, 1991), Wright (ALLEN; GARLAN, 1997). A Tabela 2.1 representa o foco de cada uma das ADLs citadas.

A classificação de uma ADL é dada através da seguinte lista das características de modelagem da arquitetura (MEDVIDOVIC; TAYLOR, 2000):

- **Componentes**: são abstrações que encapsulam funcionalidades em blocos reutilizáveis podendo representar tanto artefatos de software quanto hardware. Um tipo de componente pode ser instanciado várias vezes em uma única arquitetura ou pode ser reutilizado nas arquiteturas. Os componentes podem ser tão pequenos quanto um único procedimento ou tão grandes quanto uma aplicação inteira. Cada componente pode exigir seus próprios dados ou espaço de execução, ou pode compartilhá-los com outros componentes. Uma comparação mais detalhada pode ser vista no Anexo C.
- **Conectores**: são blocos de arquitetura utilizados para modelar interações entre os componentes e as regras que controlam essas interações. Diferentemente de componentes, conectores podem não corresponder a unidades de compilação em um sistema implementado. Uma comparação mais detalhada pode ser vista no Anexo D.

Tabela 2.1 - Aplicabilidade das ADLs

ADL	Foco
ACME	Permuta de arquiteturas, principalmente a nível estrutural.
Aesop	Especificação de arquiteturas em estilos específicos.
C2	Arquitetura de sistemas dinâmicos, evolutivos e altamente distribuídos.
Darwin	Arquiteturas de sistemas altamente distribuídos por pressupostos estritamente formais.
MetaH	Arquiteturas de domínio para aviação, navegação e controle.
Rapide	Modelagem e simulação do comportamento dinâmico descrito por uma arquitetura.
SADL	Refinamento formal de arquiteturas através de diferentes níveis de detalhamento.
UniCon	Geração de código para interconexão de componentes existentes utilizando protocolos comuns de interação.
Weaves	Arquitetura de fluxo de dados, caracterizada por alto volume de dados e de requisitos em tempo real em seu processamento.
Wright	Modelagem e análise (especificamente, análise de *deadlock*) de comportamento dinâmico de sistemas concorrentes.

Fonte: Medvidovic e Taylor (2000)

- **Topologia (Configuração de Arquitetura)**: está ligada graficamente a componentes e conectores que descrevem a estrutura da arquitetura. Esta informação é necessária para determinar se componentes estão conectados adequadamente. As descrições de configurações permitem avaliar a concorrência e distribuição de aspectos da arquitetura. Bem como a análise de arquiteturas para a adesão da concepção heurísticas e restrições da arquitetura. O Anexo E mostra uma comparação mais detalhada entre as ADLs.

- **Ferramentas de Suporte**: a motivação por trás do desenvolvimento de linguagens de descrição de arquiteturas formais é que a sua formalidade torna o raciocínio e a manipulação de ferramentas de software adequadas. Ferramentas de apoio não fazem parte da linguagem. No entanto, a utilidade de uma ADL é diretamente relacionada aos tipos de ferramentas que dispõe para apoiar concepção da arquitetura, análise, evolução, geração de sistema executável, e assim por diante. Embora

ainda preliminares, várias ferramentas tem surgido. O Anexo F mostra uma comparação mais detalhada entre as ferramentas.

2.4 Sistemas Baseados em Conhecimento

Um Sistema Baseado em Conhecimento (RUSSEL; NORVIG, 1995) é um sistema de apoio à decisão que procura representar o modo de raciocínio e o conhecimento utilizado por especialistas na resolução de problemas no seu domínio de especialidade. Estes sistemas são aplicados a diversos domínios de problemas, como diagnóstico médico, exploração de minerais, sistemas de cabeamento telefônico, entre outros. Na computação, estes sistemas são estudados na área de Inteligência Artificial (RICH; KNIGHT, 1991).

De forma mais abrangente, todas as situações, quer envolvam planejamento, diagnóstico, interpretação de dados, otimização ou comportamento social, podem ser expressas mediante um conjunto de regras bem definidas (RUSSEL; NORVIG, 1995).

As componentes básicas de um sistema baseado em conhecimento são (SANTOS, 2006):

- **Base de conhecimento ou base de regras**: constitui uma espécie de base de dados que contém as regras do sistema de um domínio específico abordado.
- **Interface com o usuário**: é uma interface amigável, onde são introduzidos os fatos e se recebem os resultados ou conclusões retiradas pelo sistema.
- **Interpretador de regras ou mecanismo de inferência**: é o algoritmo que a partir de antecedentes estabelece os consequentes com base nas regras.

Este tipo de sistemas tem grande aplicação nos casos em que se consegue representar pequenas peças de conhecimento do domínio de um problema mas não existe um procedimento definido para se encontrar uma solução ótima. Se colocarmos um especialista para resolver um problema, ele irá seguir um determinado caminho para chegar a uma solução; outro especialista poderá seguir

um caminho diferente e, ainda assim, chegar à mesma solução. A resolução do problema depende do caminho adotado, ou seja, das peças de conhecimento que são analisadas por cada perito, sendo extremamente difícil definir qual é a solução global ótima (CARDOSO; MARQUES, 2007).

Uma das grandes vantagens deste tipo de sistema é a possibilidade de substituir a base de conhecimento, mantendo os componentes restantes, para se ter um novo sistema especialista de um diferente domínio do problema.

3 SATÉLITE ARTIFICIAL E O PROJETO ITASAT

Um satélite artificial é qualquer corpo feito pelo homem e colocado em órbita ao redor da Terra ou de qualquer outro planeta. Hoje em dia, ao contrário do que ocorreria no ínicio da história dos satélites artificiais, o termo satélite vem sendo usado praticamente como sinônimo para "satélite artificial". O termo "satélite artificial" tem sido usado quando se quer distinguí-los dos satélites naturais, como a Lua (NASA, 2009).

Atualmente podemos classificar os satélites em:

- satélites do sistema de posicionamento global (GPS - *Global Positioning System*);
- satélites de comunicações;
- satélites científicos;
- satélites militares;
- satélites astronômicos;
- satélites de reconhecimento;
- satélites de observação da Terra, e
- satélites meteorológicos.

Um satélite é conceitualmente dividido em duas partes: a **carga útil** e a **plataforma**. (SOUZA, 2003)

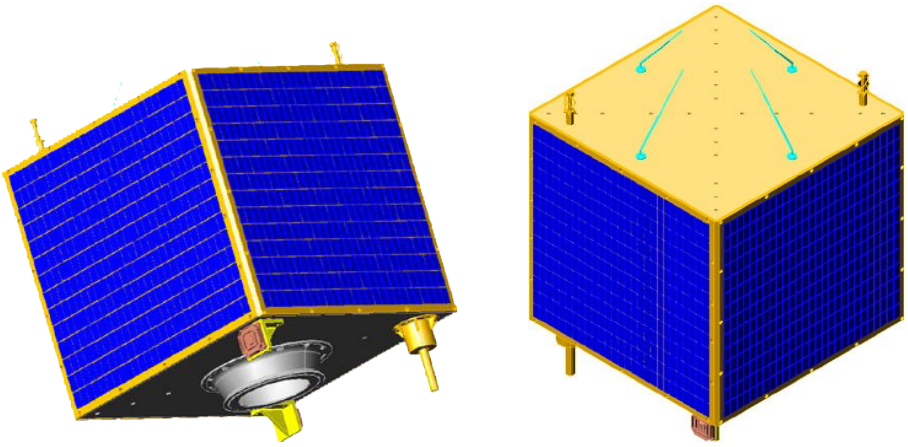
A carga útil é responsável pelo tipo de missão a ser executada pelo satélite. Por conseqüência, atualmente existe uma grande variedade de cargas úteis presentes nos satélites em órbita, reflexo da multiplicidade de missões por eles executadas.

A plataforma é responsável por acomodar a carga útil, sendo dividida em subsistemas. O objetivo da divisão é sistematizar os trabalhos de engenharia requeridos no projeto (divisão por áreas de competência). Usualmente os subsistemas básicos para satélites podem ser (LARSON; WERTZ, 1999):

- Controle de Atitude (*Attitude Determination and Control ou Attitude Control System*)
- Suprimento de Energia (*Electrical Power and Distribution*)
- Telecomunicação de Serviço (*Telemetry, Tracking and Command*)
- Gestão de Bordo (*Command and Data Handling*)
- Estrutura e Mecanismos (*Structures and Mechanisms*)
- Controle Térmico (*Thermal Control*)
- Propulsão (*Propulsion*)

3.1 ITASAT

O satélite abordado neste trabalho é o ITASAT (AGÊNCIA ESPACIAL BRASILEIRA (AEB) et al., 2005), um satélite universitário iniciado no ano de 2005, atualmente sendo desenvolvido pelo INPE (Instituto Nacional de Pesquisas Espaciais), ITA (Instituto Tecnológico de Aeronáutica) e algumas universidades públicas brasileiras. As Figuras 3.1(a) e 3.1(b) ilustram a parte inferior e superior do satélite ITASAT, respectivamente.



(a) Parte Inferior (b) Parte Superior

Figura 3.1 - Satélite ITASAT
Fonte: $http$: $//www.inpe.br/ingles/news/news_dest3.php$

O satélite ITASAT, conta com a participação de acadêmicos do próprio ITA nas áreas de engenharia e computação, da Universidade de São Paulo (USP), campus de São Carlos, nas áreas de engenharia elétrica e telecomunicações, e da Universidade Estadual de Campinas (Unicamp) na área de computação (MILESKI, 2007). O projeto é coordenado pelo INPE e AEB (Agência Espacial Brasileira).

Além da sua funcionalidade experimental, o ITASAT irá contar com um transponder para transmissão dos dados coletados das PCDs (Plataforma de Coleta de Dados) distribuídas por todo o território brasileiro, como mostrado na Figura 3.2. Isso irá representar um importante reforço para o Sistema Nacional de Coleta de Dados. Por este motivo, deseja-se que a órbita do ITASAT seja compatível com as órbitas dos satélites SCD1 e SCD2 (Satélite de Coleta de Dados), ou seja, órbita de baixa altitude e quase equatorial (aproximadamente 750Km com inclinação de 25 graus).

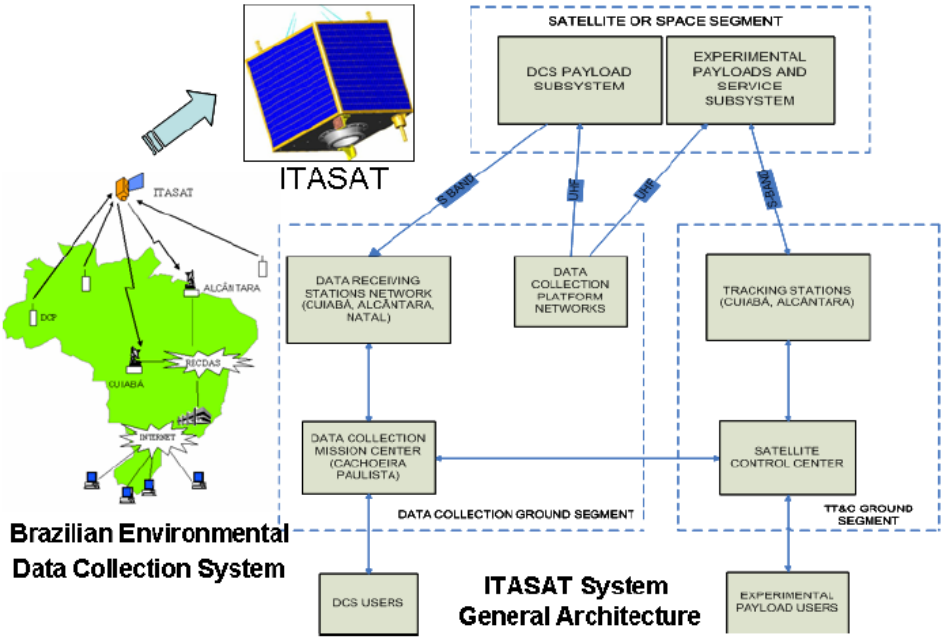


Figura 3.2 - Esquema de Coleta e Transmissão de dados das PCDs

Alguns dos requisitos iniciais para o ITASAT são (AGÊNCIA ESPACIAL BRASILEIRA (AEB) et al., 2008):

- A massa do satélite como um todo não deve exceder 80Kg;

- Os painéis solares devem ser dispostos paralelos ao eixo de spin, para assegurar a iluminação necessária para gerar a potência requerida;
- O satélite, incluindo antenas e demais apêndices, deverá estar contido em um cilindro de aproximadamente 1m de diâmetro por 1m de altura.

A massa inicial estimada para a configuração atual do satélite ITASAT é de 73,6Kg.

A Figura 3.3 representa a estrutura das especificações e documentações do Satélite ITASAT, onde são armazenados os requisitos do sistema.

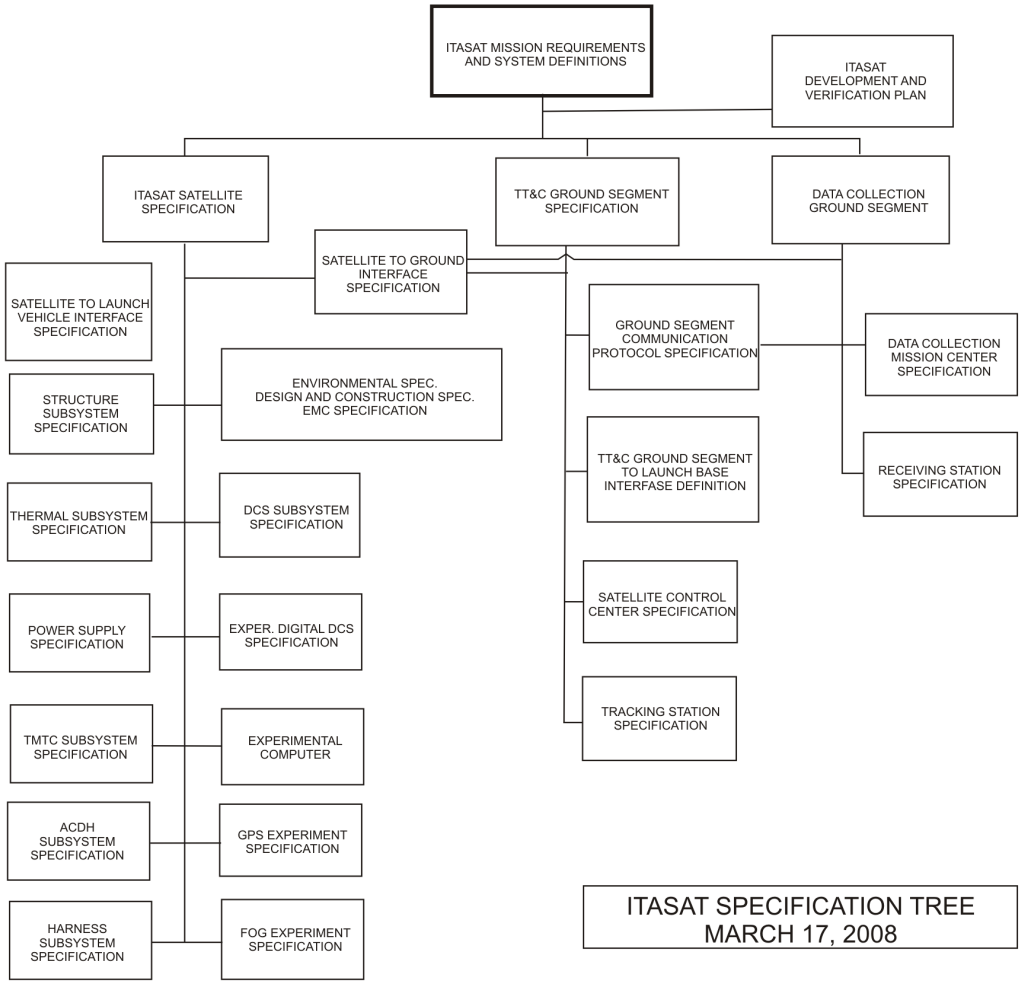


Figura 3.3 - Árvore de Especificação do ITASAT

4 LINGUAGEM SysML

Muitos processos de engenharia de sistemas tendem a ser documentados e empregam uma combinação de várias técnicas de diagramas que são muitas vezes imprecisas e incoerentes. Por este motivo engenheiros de software em geral procuraram uma linguagem de modelagem como a UML (OMG, 2009b) para especificar precisamente sistemas de software. Por outro lado, engenheiros de sistemas buscam uma linguagem de modelagem de domínio específico para especificar sistemas complexos que incluem *hardware*, *software*, informação, pessoal, processos e instalações. Com objetivo de atingir esta meta, o OMG e o INCOSE (do inglês *International Council on Systems Engineering*) (INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING (INCOSE), 2009) sugeriram a criação de uma ADL padrão e de uso geral dentro da Engenharia de Sistemas, chamada SysML (OMG, 2009a), como extensão da UML, que possibilita a modelagem de qualquer tipo de domínio, padroniza a comunicação entre as pessoas e grupos envolvidos no projeto, melhora a gestão do ciclo de vida do produto e captura artefatos da engenharia de sistemas.

SysML é uma linguagem gráfica de modelagem que apoia a análise, especificação, projeto, verificação e validação de sistemas complexos (FRIEDENTHAL et al., 2008).

Como é derivada da UML, SysML possibilita a criação de modelos de sistemas orientado a objetos, permitindo expressar tanto a estrutura como o funcionamento do sistema.

A organização da SysML permite definir as seguintes entidades:

- **Requisitos**: são especificados para resolver as necessidades das partes interessadas;
- **Blocos**: são utilizados para modelar uma ampla variedade de elementos do sistema, como *hardware*, *software*, objetos físicos, entre outros. Isto é, um bloco pode representar qualquer elemento real ou abstrato que pode ser conceituado como uma unidade estrutural com uma ou mais características distintas;
- **Atividades**: é um formalismo para descrever um comportamento que

especifica a transformação de insumos para saídas controladas através de uma seqüência de ações;

- **Blocos Restritos**: é um tipo especial de bloco utilizado para definir as equações para que elas possam ser reutilizadas e interligadas. Blocos restritos tem duas características principais: um conjunto de parâmetros e uma expressão que limita os parâmetros, e

- **Portas e fluxos**: são características estruturais que descrevem os pontos em que um bloco interage com outro e são usados para conectar as partes de um bloco. SysML suporta dois tipos de portas: portas de fluxo, que especificam o que pode fluir para dentro e para fora dos blocos e portas padrão, que especificam os tipos de serviços que um bloco requer ou proporciona.

4.1 SysML como extensão da UML

A UML se estabeleceu como uma linguagem de modelagem no domínio de desenvolvimento de software, sendo hoje um padrão internacional especificado pela OMG.

Apesar de uma série de iniciativas para padronizar os processos de engenharia de sistemas, não se obtinha como resultado uma linguagem de modelagem uniforme. Isto levou a grandes perdas em projetos interdisciplinares. A forma de comunicação entre os modelos é difícil, levando a mal-entendidos e a necessidade do uso de diferentes ferramentas.

Em 2001 o INCOSE decidiu adaptar a UML uma linguagem padrão para engenharia de sistemas. UML reune essencialmente os requisitos, é amplamente utilizada, pode ser adaptada a necessidades específicas, e há um grande número de ferramentas de modelagem existentes no mercado. Graças ao mecanismo de extensão (estereótipos), um novo vocabulário de modelagem pode ser definido, tanto que UML pode ser adaptada para especificar domínios e disciplinas. A adaptação da UML para engenharia de sistemas ganhou o nome de Linguagem de Modelagem de Sistemas ou SysML.

A questão é saber se todas as alterações feitas na UML para apoiar a engenharia de sistemas são justificadas, ou em outras palavras, "Por que outra linguagem

de modelagem?". A resposta é que, embora UML seja uma linguagem poderosa, ela tem algumas lacunas no que diz respeito a engenharia de sistemas. Outra razão para a necessidade de SysML é o fato de que a UML é bastante específica para software e fortemente caracterizada pela orientação a objetos, enquanto a modelagem de sistemas é interdisciplinar. A utilização da UML para modelar sistemas pode levar a problemas e mal-entendidos nos aspectos interdisciplinares envolvidos (WEILKIENS, 2007).

4.2 Diagramas SysML

A SysML reutiliza um subconjunto da UML e acrescenta alguns novos diagramas especificamente concebidos para apoiar a engenharia de sistemas. Uma organização básica dos diagramas SysML e sua relação com a UML são apresentadas na Figura 4.1.

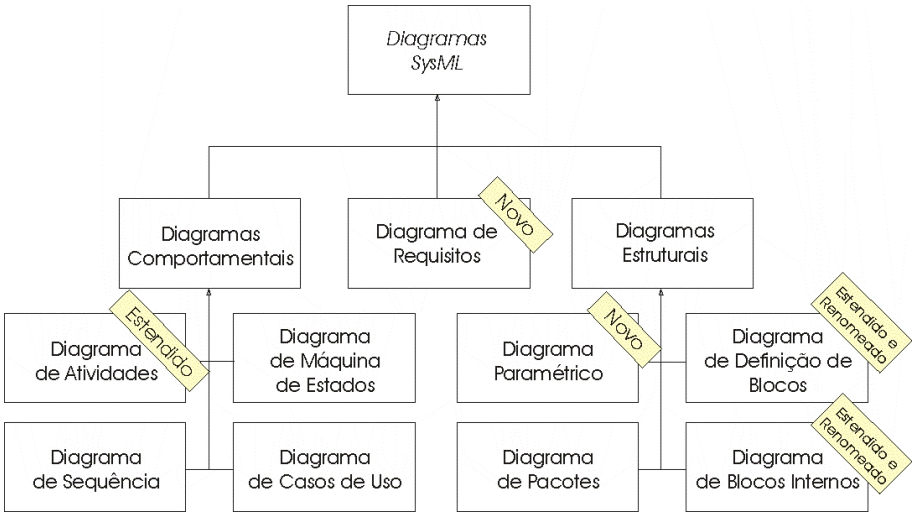


Figura 4.1 - Diagrama de Taxonomia SysML
Fonte: Weilkiens (2007)

A SysML possui nove diagramas, descritos a seguir:

- **Diagrama de Requisitos** é utilizado para representar hierarquias entre requisitos ou mostrar uma exigência individual e sua relação com outros elementos do modelos.

- **Diagrama de Atividades** é usado para modelar o comportamento em termos de fluxo de entradas, saídas, e de controle.

- **Diagrama de Sequência** é usado para representar a interação entre elementos estruturais de um bloco, através de mensagens. Entende-se por mensagem o serviço solicitado de um objeto a outro, e a resposta desenvolvida para as solicitações.

- **Diagrama de Máquina de Estado** são utilizados para descrever o estado dependente do comportamento de um bloco ao longo do seu ciclo de vida em termos de seus estados e as transições entre eles. Transições são representadas por flechas, etiquetadas com seu evento, e estados são representados por retângulos com as bordas arredondadas.

- **Diagrama de Casos de Uso** descreve a funcionalidade proposta para o sistema. Um caso de uso representa uma unidade discreta da interação entre um ator (humano ou máquina que interage com o sistema para executar um trabalho) e o sistema. Casos de uso são tipicamente relacionados a atores.

- **Diagrama de Definição de Blocos** é utilizado para definir as características dos blocos em termos de características estruturais e comportamentais, e as relações entre os blocos, tais como a sua relação hierárquica. Extensões do diagrama são utilizadas para definir restrições paramétricas e também para mostrar uma hieraquia das atividades.

- **Diagrama de Bloco Interno** é usado para descrever a estrutura interna de um bloco em termos de como as suas partes estão interligadas.

- **Diagrama Paramétrico** são utilizados para criar sistemas de equações que pode ser utilizadas para restringir as propriedades dos blocos.

- **Diagrama de Pacotes** descreve os pacotes ou pedaços do sistema divididos em agrupamentos lógicos mostrando as dependências entre eles, ou seja, pacotes podem depender de outros pacotes. São muitas vezes utilizados para ilustrar a arquitetura de um sistema - as camadas, subsistemas, pacotes, etc. São também usados para definir extensões da linguagem SysML, chamados perfis.

Os diagramas SysML são portanto classificados em três grupos:

- **Diagramas Estruturais**: Diagrama de Blocos, Diagrama de Blocos Internos, Diagrama de Pacotes e Diagrama de Parâmetros;
- **Diagramas Comportamentais**: Diagrama de Atividades, Diagrama de Sequência, Diagrama de Máquina de Estados e Diagrama de Casos de Uso, e
- **Diagramas Transversais (*Cross-cutting Constructs*)**: Diagrama de Requisitos.

4.3 Uso proposto da SysML

Um estudo de caso aplicado ao satélite ITASAT será descrito nesta seção, onde serão apresentados alguns exemplos dos diagramas SysML utilizados para modelagem do sistema. A linguagem adotada nestes diagramas foi o Inglês devido à documentação do projeto ITASAT assim requerer.

A Figura 4.2 representa um exemplo de diagrama de blocos de uma das versões arquiteturais do ITASAT. Nesta Figura são mostrados os blocos definidos para o ITASAT e os relacionamentos entre eles, além de representar a utilização de pacotes, tal como o pacote do subsistema ACDH (do inglês, *Attitude Control and On-Board Data Handling*), que inclui outros blocos utilizados no sistema.

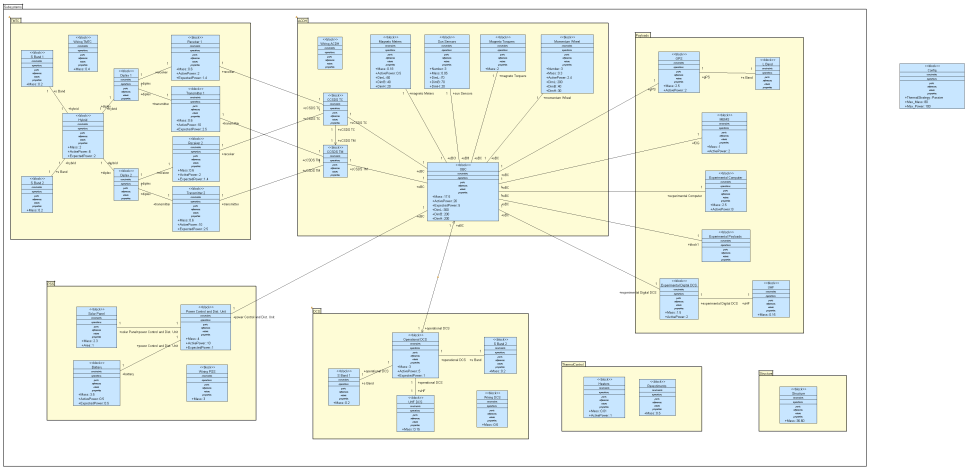


Figura 4.2 - Diagrama de Blocos do ITASAT

A título de exemplo, a Figura 4.3 apresenta um diagrama de pacotes de estruturação de requisitos. Um pacote é um recipiente para outros elementos do modelo e auxiliam a identificar a hierarquia entre elementos, assim podemos dizer que se um pacote é excluído ou copiado, os elementos contidos neste pacote serão excluídos ou copiados juntamente com ele. O exemplo apresentado na Figura 4.3 representa uma hierarquia entre os pacotes *Requirements*, *Operational*, *Operability* e *Telemetry Design*, sendo este último um pacote de requisitos.

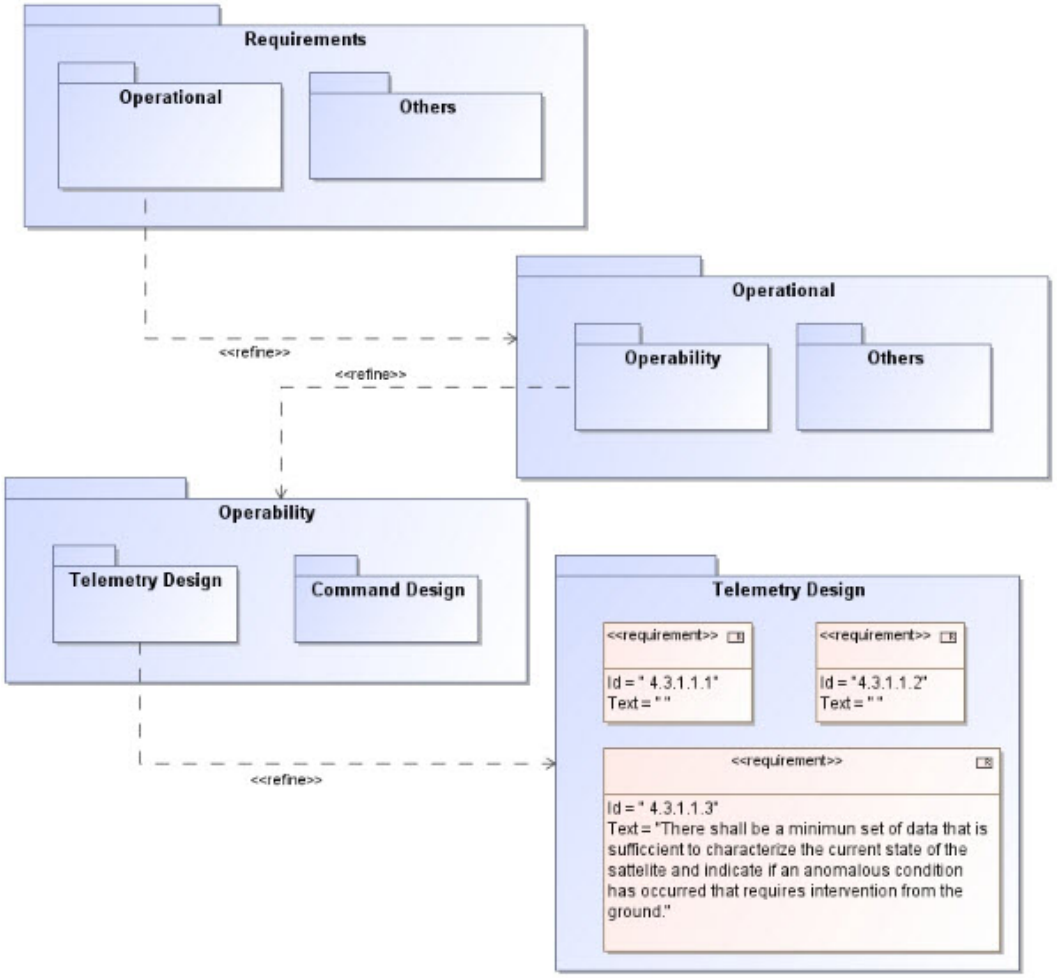


Figura 4.3 - Diagrama de Pacotes Simplificado de Requisitos do ITASAT

A Figura 4.4 exemplifica um diagrama de requisitos e mostra alguns relacionamentos possíveis dentro do diagrama. O diagrama apresentado é para o subsistema de potência do ITASAT. A Figura 4.4, apresenta o relacionamento de

requisitos com casos de uso, casos de teste, blocos e outros requisitos, além de apresentar uma maneira de como se obtém a rastreabilidade de requisitos.

Para este exemplo, pode ser descrito o relacionamento entre o requisito *Power Supply Requirements* e os casos de uso *Power Supply Functions* e *Test Power Supply Functions*. O caso de uso *Power Supply Functions* está relacionado com o requisito através de um relacionamento ≪satisfy≫, o que indica que este caso de uso deve satisfazer a implementação do requisito e o caso de uso *Test Power Supply Functions* está relacionado através de um relacionamento ≪verify≫, indicando que este caso de uso irá verificar se o requisito é satisfeito.

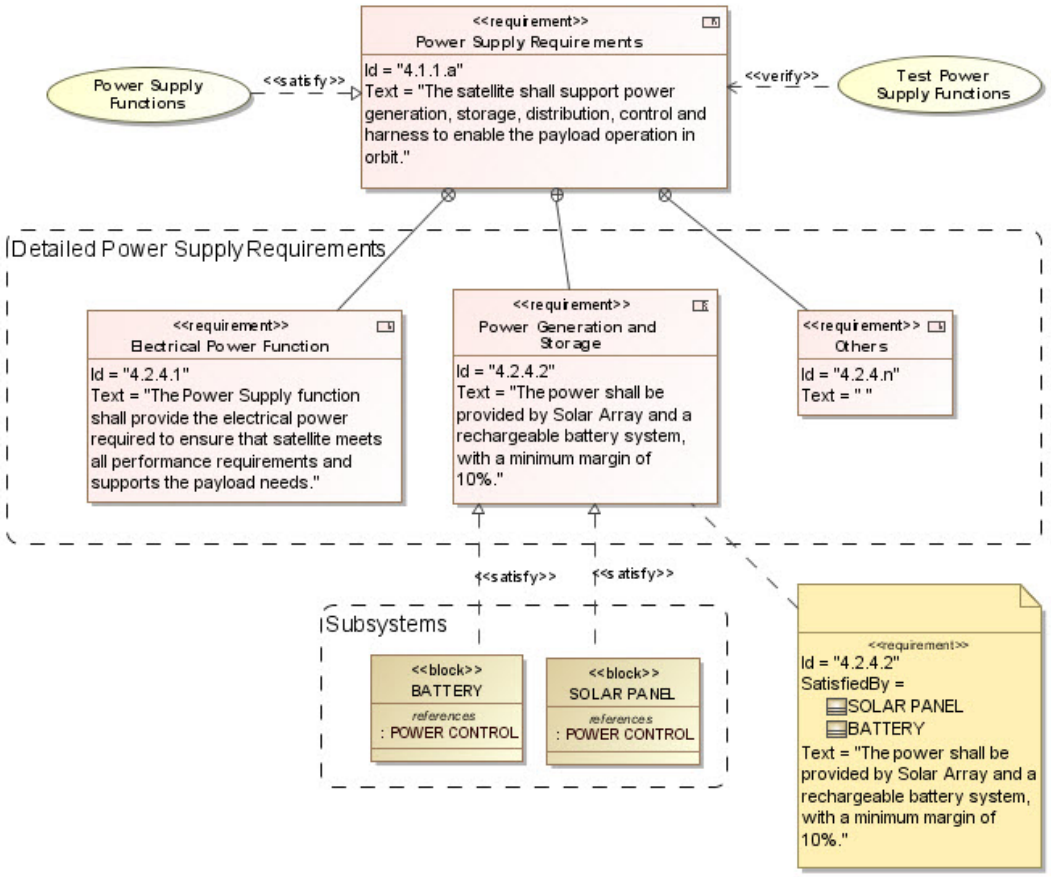


Figura 4.4 - Diagrama de Requisitos do Subsistema de Potência

Outros tipos de relacionamentos envolvendo requisitos estão descritos na Tabela 4.1.

Tabela 4.1 - Relacionamento de Requisitos

Elemento	Notação	Descrição
Containment	$\oplus$	O relacionamento containment é usado para representar como requisitos estão contidos nas especificações (pacotes), ou como um requisito complexo pode ser dividido em uma série de requisitos mais simples, sem acrescentar ou alterar seu resultado.
Derivação	≪deriveReqt≫	O relacionamento de derivação ocorre entre o requisito fonte e um requisito derivado, baseado na análise do requisito fonte.
Satisfação	≪satisfy≫	O relacionamento de satisfação é utilizado para afirmar que um elemento do modelo corresponde à concepção ou satisfaz a implementação de um determinado requisito.
Verificação	≪verify≫	O relacionamento de verificação é usado entre um requisito e um caso de teste ou outro elemento chamado para indicar a forma de verificar que a exigência seja satisfeita.
Refinamento	≪refine≫	O relacionamento de refinamento é usado para reduzir a ambiguidade em um requisito, relacionando ele com outro elemento do modelo que esclarece a exigência.
Rastro	≪trace≫	O relacionamento de rastro é uma forma de uso geral para relacionar um requisito e qualquer outro elemento do modelo, útil para exigências relacionadas a documentação, etc.
Cópia	≪copy≫	O relacionamento de cópia diz respeito a cópia de um requisito original para a exigência, para apoiar a reutilização de requisitos.

Um exemplo de diagrama de casos de uso é apresentado na Figura 4.5 onde alguns atores, casos de uso e seus relacionamentos são listados. No diagrama apresentado, um exemplo pode ser descrito para o ator *Satellite Control Center* que por sua vez dispara vários casos de uso, entre eles o caso de uso *On-Board Management Functions*, que ao ser disparado irá disparar outros três casos devido ao seu tipo de relacionamento (≪include≫). Existe também a possibilidade do caso de uso *On-Board Management Functions* invés de ser disparado pelo ator *Satellite Control Center* ser disparado pelo caso de uso *Satellite Commissioning and OrbOps* através do relacionamento ≪extend≫.

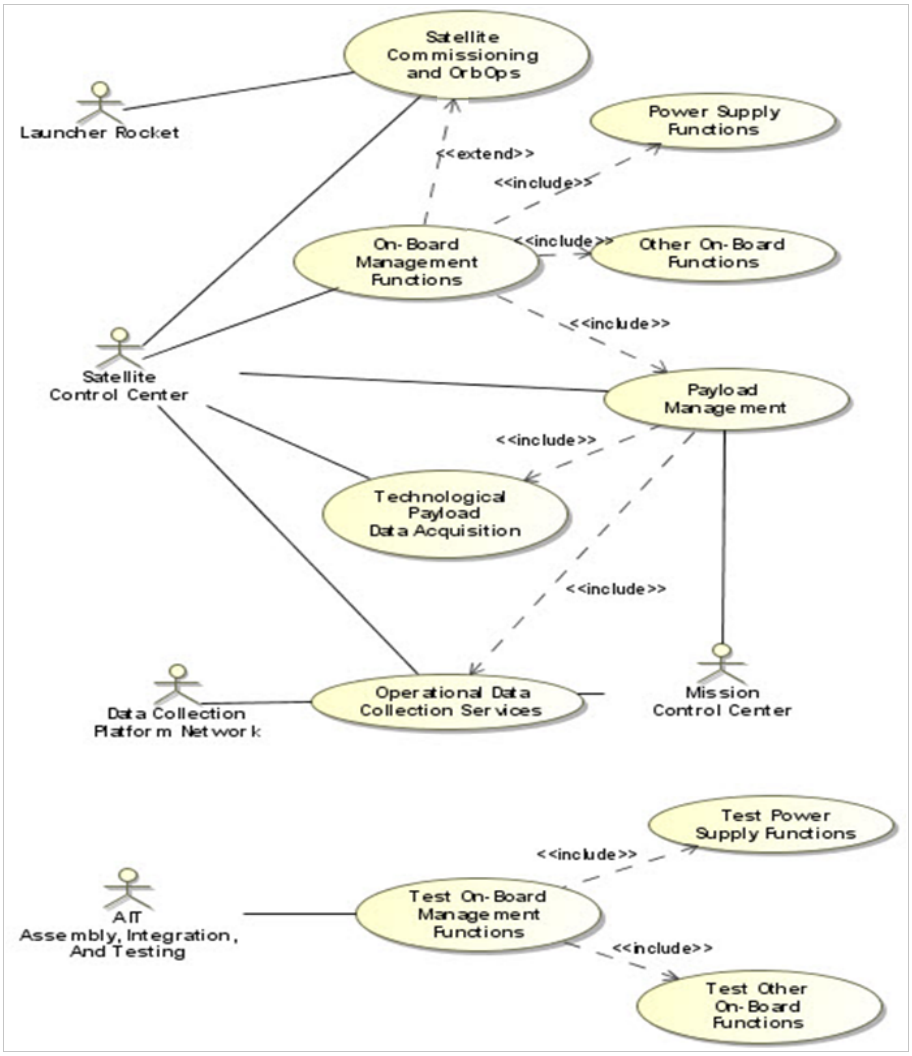


Figura 4.5 - Diagrama Simplificado de Casos de Uso do ITASAT

No centro da atividade de gerenciamento de requisitos está a rastreabilidade. Esta é definida como a capacidade de se poder acompanhar os efeitos do estabelecimento de um requisito em ambas as direções de uma árvore de requisitos (por exemplo: para baixo, partindo de um requisito e chegando a projeto ou, para cima, ou seja, partindo do projeto e chegando a outros requisitos). A árvore de requisitos possibilita a rastreabilidade durante todo o ciclo de vida do projeto (SPÍNOLA; ÁVILA, 2007).

A dificuldade envolvida com a rastreabilidade está no grande volume de informações geradas. As informações decorrentes de requisitos de um componente do projeto devem ser catalogadas e associadas aos outros componentes de forma que o impacto da mudança de um componente possa ser avaliado nos demais componentes do sistema. A rastreabilidade demanda um trabalho extenso que, sem o auxílio de ferramentas adequadas pode se tornar exaustivo e causar erros de projeto.

Para implementar a rastreabilidade, usualmente é elaborada a chamada matriz de rastreabilidade, em conjunto com a especificação de requisitos, que permite visualizar as relações de dependência entre os componentes de um sistema que constituem os rastros de cada requisito. Estes rastros podem ser de dois tipos:

- Pré-rastreabilidade: os rastros (artefatos: plano de negocio, atas de reunião com o usuário) que fundamentaram a criação do requisito.
- Pós-rastreabilidade: os rastros (artefatos: código, documentos) que se formaram a partir do requisito criado.

A SysML possibilita ao engenheiro de sistema rastrear os requisitos e correspondentes componentes que fazem parte do sistema através de uma matriz de rastreabilidade como a apresentada na Figura 4.6. A matriz permite ao engenheiro de sistema relacionar os elementos do modelo e descobrir qual elemento está se relacionando com outro e qual a sua origem.

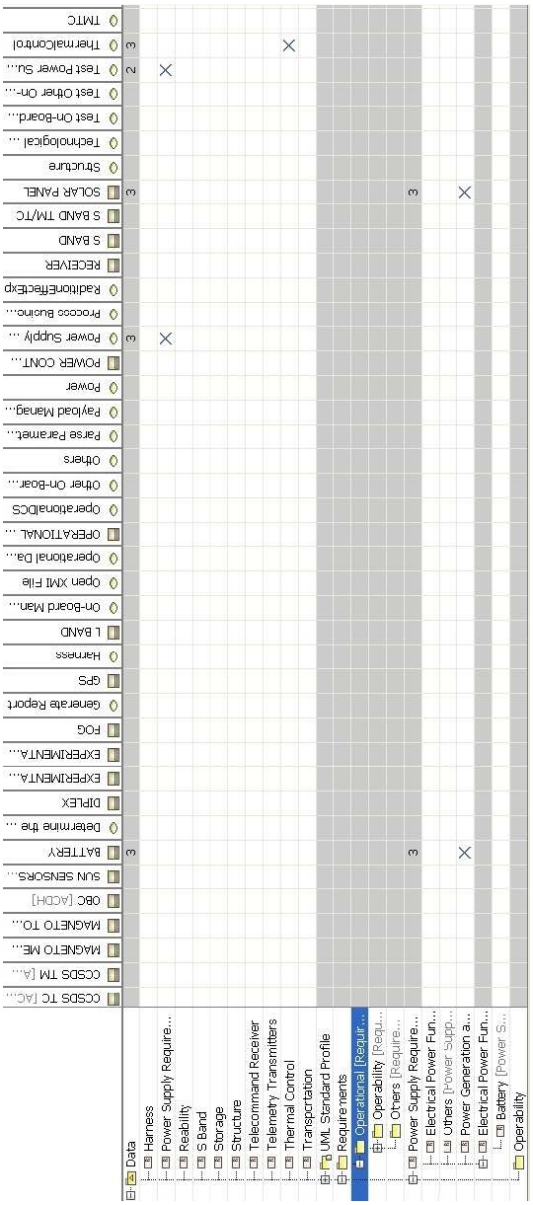
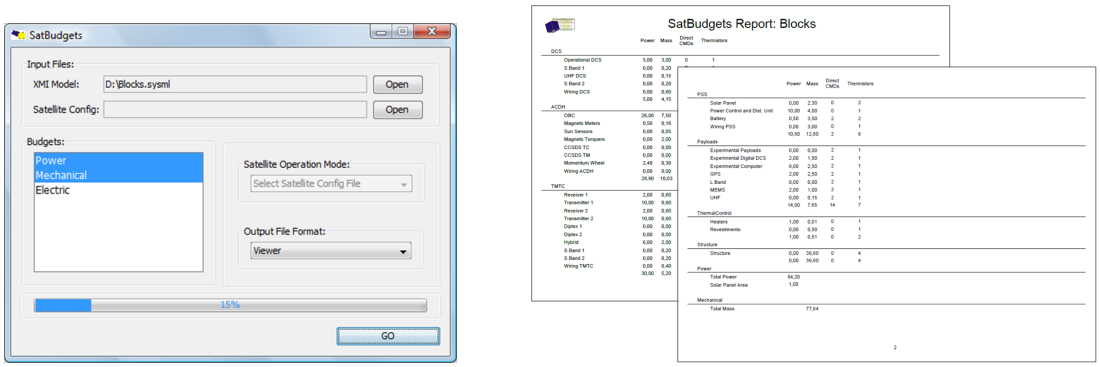


Figura 4.6 - Tabela de Rastreabilidade

5 SATBUDGETS

SatBudgets é o protótipo de uma ferramenta concebida neste trabalho para auxiliar na fase conceitual de um projeto de satélite gerando automaticamente balanços (mecânico, elétrico, entre outros) para um projeto de satélite. Um estudo de caso foi aplicado fazendo uso de dados do projeto do satélite ITASAT.

A Figura 5.1(a) apresenta a tela principal da ferramenta SatBudgets e sumariamente, a Figura 5.1(b) a planilha de balanço gerada pela mesma.



(a) Tela Principal (b) Relatório

Figura 5.1 - SatBudgets e Geração de Relatórios

5.1 Ferramentas Utilizadas

O protótipo da ferramenta SatBudgets foi desenvolvido na linguagem Java, fazendo uso somente de ferramentas e APIs (*Application Programming Interface*) de livre distribuição, sendo este um dos requisitos para o desenvolvimento da ferramenta.

Existem outras alternativas às ferramentas escolhidas, sendo estas tomadas por serem as mais convenientes para o projeto. O conhecimento prévio de utilização e grande comunidade de usuários foram alguns dos fatores que levaram a estas escolhas, a saber: (1) TopCased - diagramação SysML; (2) Eclipse - IDE de desenvolvimento; (3) JDOM - manipulação de arquivos XML; (4) DROOLS - gerenciador de regras de negócio; (5) JasperReports - API para geração de relatório e (6) iReports - designer de relatório.

5.1.1 TOPCASED

O TOPCASED é uma ferramenta que possibilita a modelagem SysML de sistemas, desde o mais simples até o mais complexo diagrama.

Esta ferramenta baseia-se no Galileo (Eclipse 3.5) e requer uma JVM 1.5 (*Java Virtual Machine*). Seu download pode ser feito como um aplicativo autônomo ou pode ser instalado diretamente através do gerenciador de instalação do Eclipse (LESCOT, 2009).

A Figura 5.2 apresenta a tela principal da ferramenta TOPCASED, onde é demonstrada a modelagem de um diagrama de blocos para o projeto do satélite ITASAT.

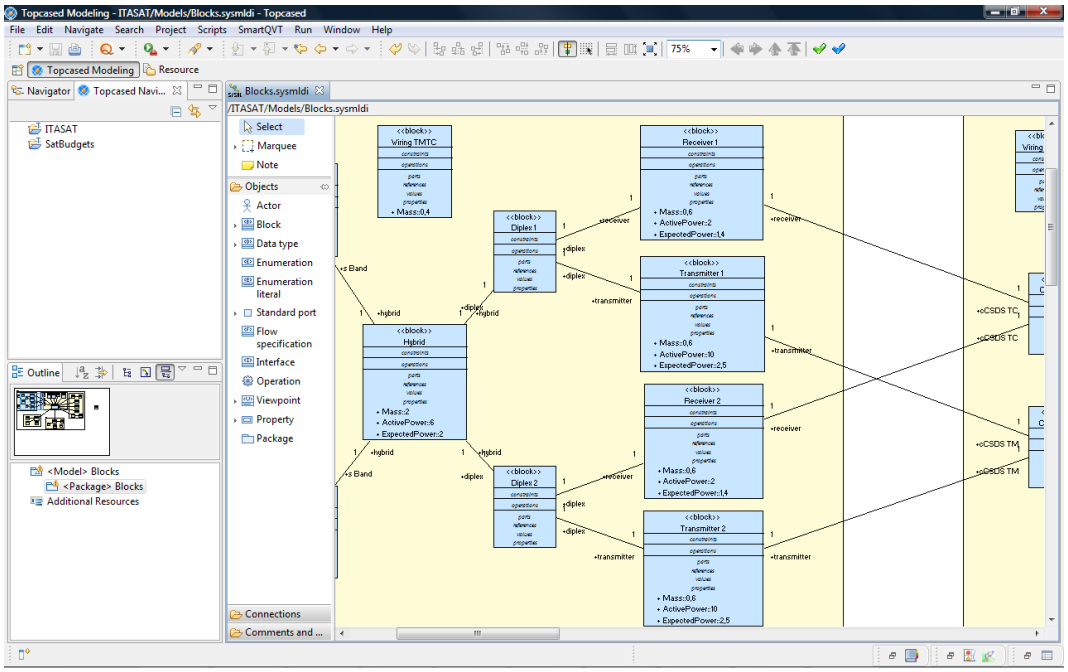


Figura 5.2 - TOPCASED - Diagramador SysML

5.1.2 Eclipse 3.5 (Galileo)

O Eclipse (BURNETTE, 2005) é uma IDE (*Integrated Development Environment*) desenvolvida em Java, com código aberto para a construção de programas de computador. O projeto Eclipse foi iniciado na IBM que desenvolveu a primeira

versão do produto e doou-o como software livre para a comunidade.

Hoje, o Eclipse é a IDE Java mais utilizada no mundo. Possui como características marcantes o uso da SWT (*Standard Widget Toolkit*) (NORTHOVER; WILSON, 2004) e não do *Swing* como biblioteca gráfica. A forte orientação ao desenvolvimento baseado em plug-ins e o amplo suporte ao desenvolvedor com centenas de plugins procuram atender as diferentes necessidades de diferentes programadores (BURNETTE, 2005).

A Figura 5.3 apresenta a tela principal de IDE Eclipse com trecho de código da implementação da ferramenta SatBudgets.

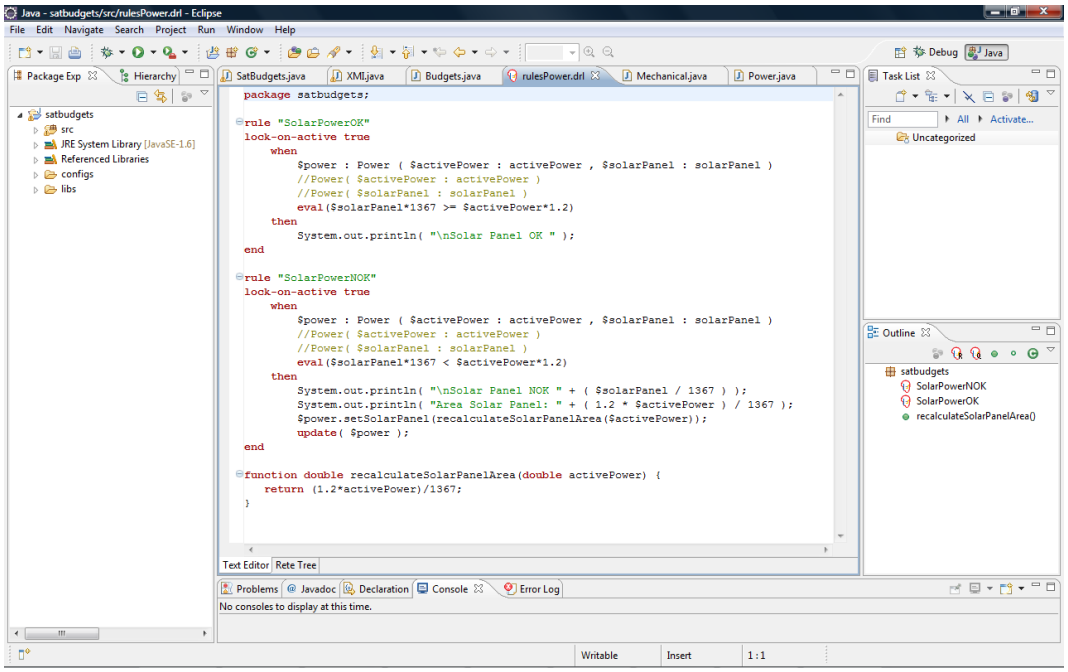


Figura 5.3 - IDE Eclipse

Mediante plugins, o Eclipse pode ser usado não só para desenvolvimento em Java, mas também em C/C++ e até mesmo PHP.

5.1.3 JDOM

A API JDOM (*Java Document Object Model*) foi utilizada para a manipulação do modelo SysML do projeto do satélite. É uma API de código aberto baseado

em Java que cria um modelo de objeto do documento XML (*Extensible Markup Language*). Esta API fornece uma forma fácil e eficiente de representar este documento para manipulação.

JDOM integra com o DOM (*Document Object Model*) e SAX (*Simple API for XML*), utilizando *parsers* externos para construir documentos (JDOM, 2009).

5.1.4 DROOLS

Drools é um mecanismo baseado em regras que foi lançado pela Codehaus e posteriormente adotado como um projeto do JBoss, conhecido também como JBoss Rules. É um sistema de gerenciamento de regras de negócios (BRMS - *Business Rules Management System*) que tem como um de seus principais objetivos separar as regras de negócios dos códigos da aplicação. Com isso, Drools permite atualizações das regras sem a necessidade de recompilação do sistema.

Em Drools, o domínio da aplicação é representado através das classes do sistema e as necessidades de uso do sistema geram as regras. Uma regra tem uma ou mais condições (ou fatos), que levam a uma ou mais ações (ou conseqüências). As regras não podem ser estruturadas como um comando de seleção IF-THEN-ELSE, pois Drools trabalha apenas com condições verdadeiras. Uma regra Drools é tratada em uma memória interna chamada de ***WorkingMemory***. A Figura 5.4 representa a estrutura de uma regra Drools.

```
rule "<name>"
    <atributo>
when
    <condições>
then
    <ações>
end
```

Figura 5.4 - Sintaxe de Regra DROOLS

5.1.4.1 Atributos da Regra

Atributos da regra fornecem uma forma declarativa para influenciar o seu comportamento. A Figura 5.5 apresenta um diagrama com os possíveis atributos que podem ser utilizados em uma regra.

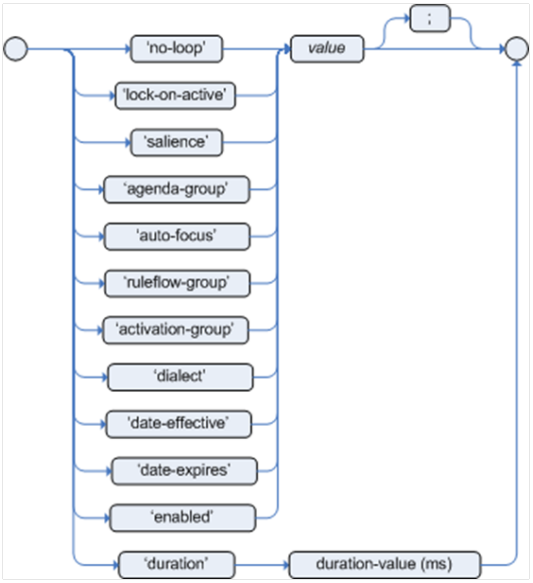


Figura 5.5 - Atributos de uma Regra DROOLS

5.1.4.2 LHS (when) - Elementos Condicionais

LHS (do inglês *Left Hand Side*) é o nome comum para a parte condicional da regra. Ela consiste de zero ou mais elementos condicionais. Se o LHS é deixado vazio, ele é re-escrito como eval(true), o que significa que essa condição é sempre verdadeira.

5.1.4.3 RHS (then) - Ações

RHS (do inglês *Right Hand Side*) é o nome comum para a parte de ação ou conseqüência da regra, esta parte pode conter uma lista de ações para serem executadas. A principal proposta da RHS é inserir e modificar as informações existentes na WorkingMemory.

5.1.5 JasperReports

A biblioteca JasperReports é uma poderosa e flexível ferramenta de geração de relatório que tem a capacidade de oferecer rico conteúdo sobre a tela, para a impressora ou em PDF, HTML, CSV, XLS, RTF ou arquivos XML (DANCIU, 2002).

A biblioteca é inteiramente escrita em Java e pode ser usada em uma variedade de aplicativos habilitados para Java, inclusive J2EE ou aplicativos da Web, para

gerar conteúdo dinâmico.

O JasperReports organiza dados de acordo com o *design* do relatório definido em um arquivo XML. Estes dados podem vir de várias fontes de dados, incluindo dados relacionais, coleções ou matrizes de objetos Java.

A fim de preencher um relatório com os dados, o design do relatório XML deve ser compilado em primeiro lugar. Através da compilação, um objeto de design do relatório é gerado e então serializado, a fim de armazená-lo em disco ou enviá-lo através da rede. Este objeto serializado é então usado quando o aplicativo quer preencher o *design* do relatório com os dados especificados

Para preencher um design de relatório, o motor precisa receber os dados do relatório. Estes dados podem ser obtidos de várias formas. Alguns desses dados podem ser passados como parâmetros de relatório, mas a maioria dos dados podem ser encontrados na fonte de dados do relatório.

O resultado do relatório, obtido mediante a operação de enchimento, é um novo objeto que representa o documento pronto para impressão. Este é também serializado para armazenamento em disco ou transferência da rede. Ele pode ser visto diretamente, usando o visualizador do JasperReports ou pode ser exportado para outros formatos mais populares como PDF, HTML ou XML.

Uma representação gráfica do fluxo de trabalho da API JasperReport é apresentada na Figura 5.6.

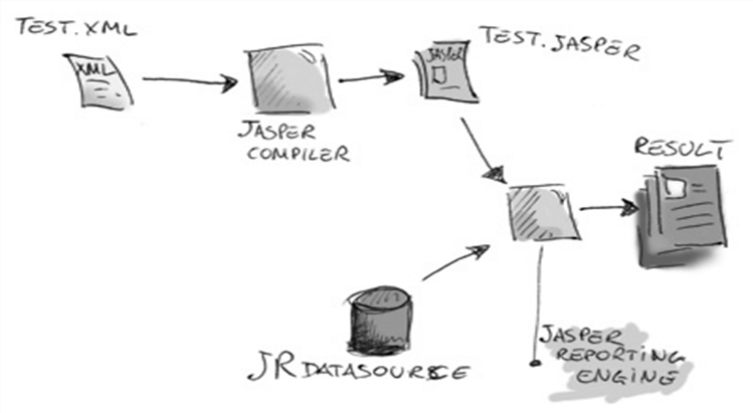


Figura 5.6 - Fluxo de Trabalho da API JasperReports

5.1.6 iReport

O iReport é um *designer* de relatório de código aberto para o JasperReports, disponível para todos os principais sistemas operacionais sob a GNU *General Public License*. O iReport possibilita a criação de layouts complexos contendo gráficos, imagens, sub-relatórios, tabelas de referência cruzada e muito mais (JASPERSOFT, 2009).

A Figura 5.7 apresenta a tela principal de ferramenta iReport onde está sendo projetado o relatório que será utilizado na ferramenta SatBudgets.

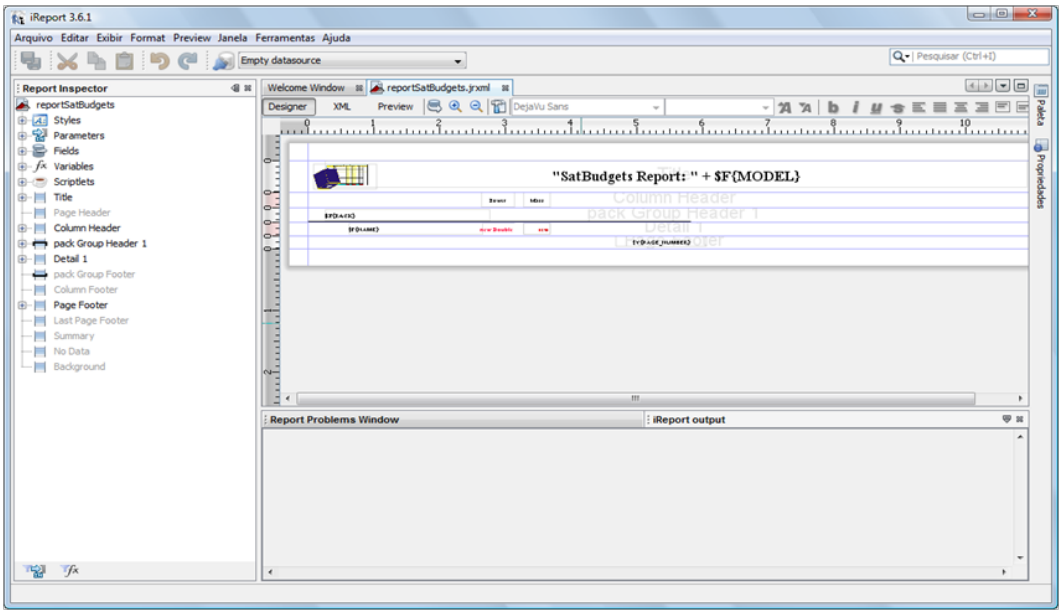


Figura 5.7 - iReport - Ferramenta de Projeto de Relatórios

5.2 Implementação

A ferramenta SatBudgets foi desenvolvida na linguagem de programação Java utilizando a IDE Eclipse.

Alguns casos de uso estabelecidos para a ferramenta SatBudgets são apresentados na Figura 5.8, onde o caso de uso "Open XMI File" representa a abertura do arquivo XMI (*XML Metadata Interchange*) que contém o modelo SysML do projeto do satélite. Já o caso de uso "Parse Parameters" representa a extração dos parâmetros do modelo. O caso de uso "Process Business Rules" processa

as regras disparando outros casos de uso mais específicos, como o "Power" por exemplo, através do relacionamento ≪include≫. E para finalizar, um último caso de uso apresentado é "Generate Report", que quando disparado irá gerar e apresentar o relatório do balanço ao engenheiro de sistema com base nos parâmetros modelados.

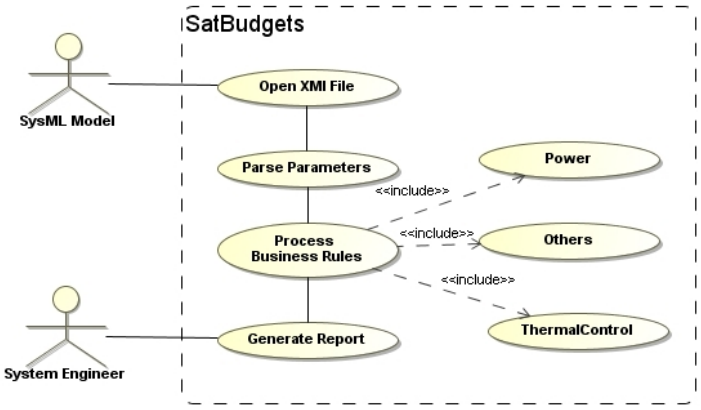


Figura 5.8 - Diagrama de Casos de Uso da Ferramenta SatBudgets

As classes modeladas para a implementação da ferramenta são apresentadas na Figura 5.9 e suas descrições são apresentadas a seguir.

- a classe XMI é responsável por tratar o arquivo XMI exportado da modelagem;
- a classe PARSER irá extrair marcadores (Tabela 5.1) das informações dos equipamentos do satélite, valores dos parâmetros, entre outros;
- a classe DISPATCHER é responsável por atribuir as informações extraídas ao processamento;
- a classe BUDGET irá envolver todas as regras de negócio da engenharia de sistemas de satélite para cada aspecto particular a calcular;
- existe um conjunto de classes que são responsáveis por representar as regras de negócio associadas aos sistemas do satélite, sendo essas regras definidas por especialistas em subsistemas de satélite, e
- finalmente, a classe REPORT é responsável por gerar o relatório.

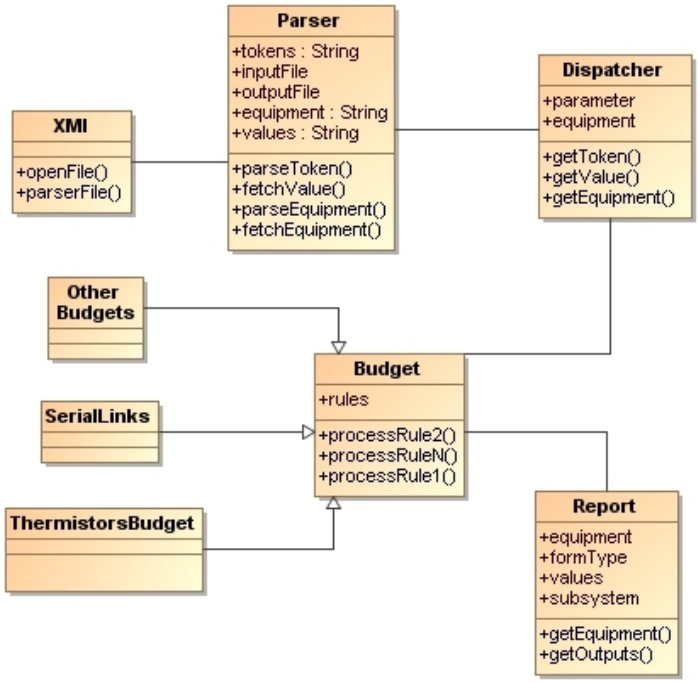


Figura 5.9 - Diagrama de Classes da Ferramenta SatBudgets

As clases XMI, PARSER e DISPATCHER foram desenvolvidas utilizando a API JDOM. A classe BUDGETS utiliza API DROOLS como motor de inferência para processamento das regras associadas aos balanços que serão calculados. Já a classe REPORT foi desenvolvida fazendo uso da API JasperReports para modelagem e geração do relatório.

A Tabela 5.1 apresenta algumas das Tags SysML, estruturas de linguagem de marcação que consistem em breves instruções, tendo uma marca de início e outra de fim, que são utilizados pela ferramenta SatBudgets para extração de dados do modelo e atribuição ao processamento das regras.

Tabela 5.1 - Tags SysML

Elemento	Sintaxe SysML
Pacote	uml:Package
Bloco	sysML:Block
Propriedade do Bloco	uml:Property

Fonte: OMG (2006)

5.3 Funcionamento

A Figura 5.10 representa o fluxo de trabalho da ferramenta desde a modelagem SysML do satélite, as etapas realizadas pela ferramenta SatBudgets até a fase final da ferramenta com a geração do relatório.

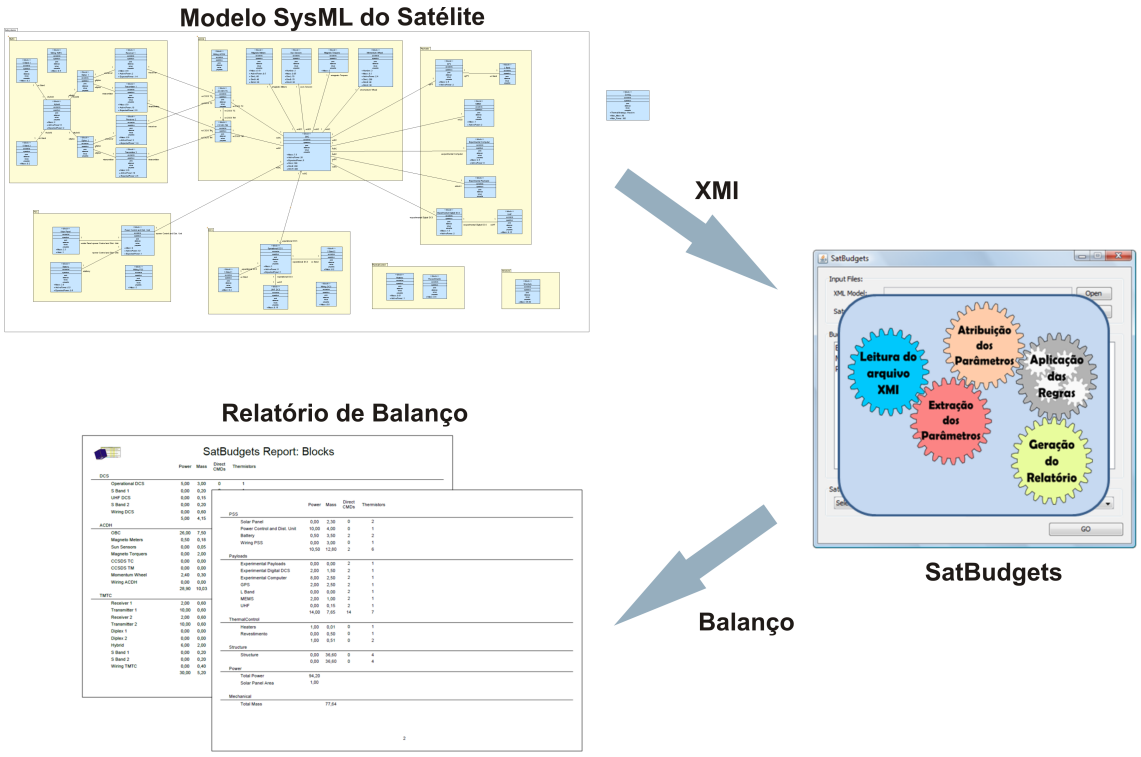


Figura 5.10 - Fluxo de Trabalho da SatBudgets

Cada uma das etapas do fluxo representado na Figura 5.10 é descrita a seguir.

A Figura 5.11 apresenta a tela principal da ferramenta SatBudgets, onde existe o campo *XMI Model*, utilizado para indicar o modelo para o qual o balanço será calculado. O campo *Satellite Config* não foi implementado nesta versão da ferramenta, a idéia de uso deste campo é que sirva para selecionar um arquivo, diagrama de estados, onde estarão descritos cada um dos modos de operação do satélite. No campo *Budgets* é selecionado qual ou quais balanços serão calculados para um determinado modelo.

Para selecionar o modo de operação do satélite para o calculo do balanço é

utilizado o campo *Satellite Operation Mode*, sendo os modos de operação carregados a partir do arquivo indicado no campo *Satellite Config*. Para definir qual o formato de saída do relatório é utilizado o campo *Ouput File Format*, podendo ser visualizado na própria ferramenta selecionando a opção Viewer, ou salvo em disco nos formatos PDF, XLS e HTML.

Existe ainda uma barra de progresso que indica o progresso da execução das tarefas da ferramenta. Finalmente um botão de execução *GO*, que dá início a execução da ferramenta depois que todos os campos citados anteriormente estejam configurados.

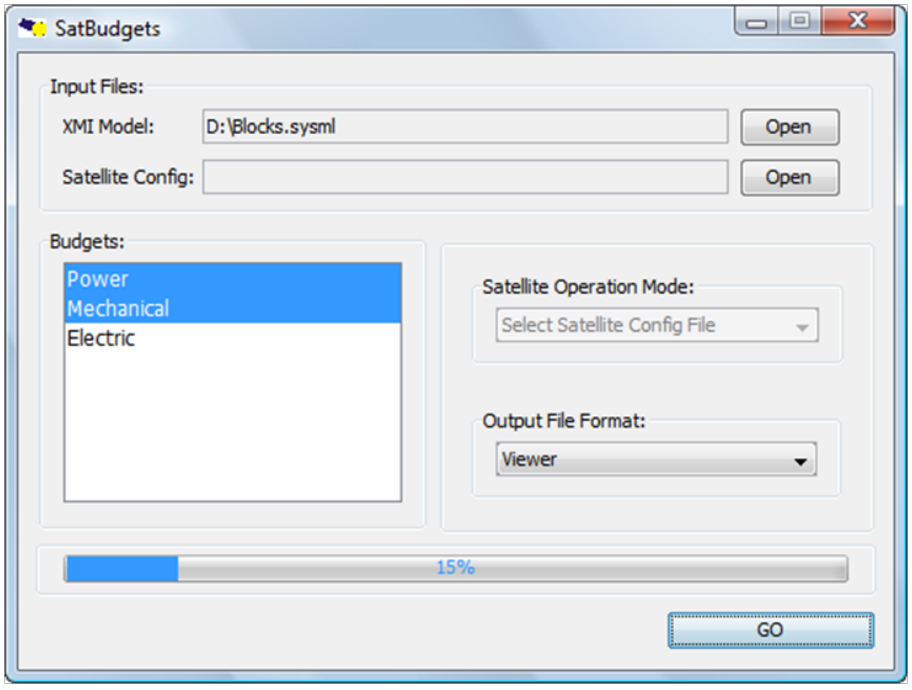


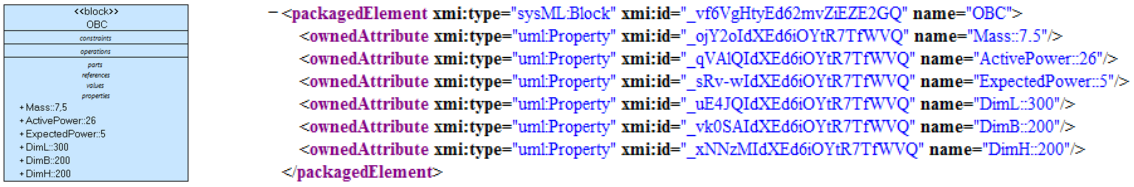
Figura 5.11 - Tela Principal da SatBudgets

5.3.1 Modelagem SysML

Um modelo SysML do sistema pretendido deve ser produzido na fase de projeto conceitual seguindo a metodologia MDE. Este modelo deve conter os requisitos e parâmetros desejados para o sistema. Os modelos utilizados neste trabalho foram desenvolvidos utilizando a ferramenta TOPCASED, apresentada na Seção 5.1.1.

A ferramenta TOPCASED utiliza o formato XMI para armazenar os modelos criados por ela. Este modelo servirá de entrada para a ferramenta SatBudgets, que utiliza as classes XMI, PARSER e DISPATCHER para extrair os parâmetros modelados, por exemplo, parâmetros mecânicos (massa, posição), elétricos (potência, links de comunicação) entre outros.

A Figura 5.12(a) representa o modelo gráfico desenvolvido pelos engenheiros e a Figura 5.12(b) representa como este modelo é mapeado utilizando a linguagem XML, onde cada tag representa um objeto da classe org.jdom.Element. A figura 5.12(a), em particular, representa o bloco correspondente ao Computador de Bordo do satélite ITASAT e alguns parâmetros.



(a) Bloco (b) Código XMI

Figura 5.12 - Mapeamento do Diagrama SysML em XMI

5.3.2 Leitura XMI

A leitura do modelo é feita pela classe implementada XMI. O trecho de código a seguir mostra como a classe XMI é utilizada para fazer a abertura do modelo SysML.

```
Document document = null;
SAXBuilder builder = new SAXBuilder();
document = builder.build(file);
Element modelXML = document.getRootElement();
```

O objeto *document* representa o documento XMI (modelo do projeto). Builder representa o *parser* que é utilizado para analisar o documento XML, neste caso foi utilizado o *parser* SAXBuilder, e processar a estrutura do documento para dentro do objeto document criado. O método build() é utilizado para processar o documento XML, criando um documento (*Document*) que irá conter a estru-

tura do arquivo XMI. Após isso, é recuperado o elemento root através do método getRootElement() da classe Document, que irá retornar um elemento que é representado pela classe Element.

5.3.3 Extração dos Parâmetros

Após extraído o elemento *root*, é criada uma lista com os seus filhos. A partir dessa lista é feita a recuperação dos elementos, que será feita pela classe PARSER criada na ferramenta SatBudgets.

A seguir é apresentado um trecho de código da classe PARSER.

```
List elementos = modelXML.getChildren();
Iterator iterator = elementos.iterator();
Element child = (Element) iterator.next();
// XMI Namespace
Namespace xmiNS = Namespace.getNamespace("xmi","http://www.omg.org/XMI");
child.getAttributeValue("type",xmiNS);
// XML Attribute
child.getAttributeValue("name");
```

O método getChildren() é utilizado para se obter uma lista de objetos da classe Element. Um *namespace* é criado para permitir que se tenha o tipo do elemento (apresentados na Tabela 5.1) que está sendo lido, por exemplo, se é um pacote, um bloco ou uma propriedade. O método getAttributeValue() é utilizado para retornar o valor de um atributo dado um nome, no caso do exemplo, o atributo tem o nome "*name*".

5.3.4 Atribuição dos Parâmetros

Selecionado o valor do atributo, esse elemento é enviado para a classe DISPATCHER que vai fazer a atribuição dos valores lidos no bloco correspondente para serem usados no processamento das regras.

O trecho de código a seguir representa como a classe DISPATCHER faz a atribuição de uma propriedade a um bloco.

```
block.setName(child.getAttributeValue("name"));
block.setProps(child.getAttributeValue("name"));
```

5.3.5 Aplicação das Regras

A classe BUDGET é responsável pelo processamento dos parâmetros, para isso, utiliza a API DROOLS.

As regras de negócio utilizadas no SatBudgets foram obtidas mediante entrevista a um especialista de Engenharia de Sistemas focado em um determinado subsistema de satélite. As regras obtidas foram então descritas em linguagem algoritmica estruturada e transformadas em regras de produção para que pudessem ser processadas pelo Motor de Inferência da ferramenta DROOLS utilizada. Neste trabalho foi minimamente ilustrada a aplicação deste método para regras dos seguintes subsistemas: (1) Potência, (2) Supervisão de Bordo e (3) Controle Térmico.

Um exemplo de conversão de uma das regras definidas na ferramenta SatBudgets para o formato DROOLS é apresentado nas Figuras 5.13 e 5.14. A Figura 5.13 representa a descrição da regra, referente ao subsistema de energia do satélite. A regra de negócio diz que o painel solar do satélite deve gerar uma energia igual ou superior a 20% da energia consumida pelo satélite. Caso o painel solar não seja capaz de gerar essa energia, uma nova área deve ser calculada para que o painel solar possa atender a este requisito.

1. FOR each equipment Ej in the first column and its power consumption Pj DO:
   TotalPower = summation of (Ej * Pj) for j in 1 ..n

2. SolarPower = (Solar Panel Area) * 1.367Watt m^2

3. IF SolarPower greater or equal to (1.2*TotalPower) THEN
   SolarPowerOK is TRUE
   ELSE
   SolarPowerOK is FALSE

4. IF SolarPowerOK is FALSE THEN
   SolarPanelArea = (1.2*TotalPower) / 1367Watt m^2

5. Return (SolarPanelArea) , (SolarPowerOK)

Figura 5.13 - Descrição da Regra de Negócios para Verificação da Área do Painel Solar

A Figura 5.14 representa a mesma regra de negócio citada anteriormente convertida para o formato DROOLS.

```
package satbudgets;

rule "SolarPowerOK"
lock-on-active true
    when
        $power : Power ( $activePower : activePower , $solarPanel : solarPanel )
        eval($solarPanel*1367 >= $activePower*1.2)
    then
        System.out.println( "\nSolar Panel OK " );
end

rule "SolarPowerNOK"
lock-on-active true
    when
        $power : Power ( $activePower : activePower , $solarPanel : solarPanel )
        eval($solarPanel*1367 < $activePower*1.2)
    then
        System.out.println( "\nSolar Panel NOK " + ( $solarPanel / 1367 ) );
        System.out.println( "Area Solar Panel: " + ( 1.2 * $activePower ) / 1367 );
        $power.setSolarPanel(recalculateSolarPanelArea($activePower));
        update( $power );
end

function double recalculateSolarPanelArea(double activePower) {
   return (1.2*activePower)/1367;
}
```

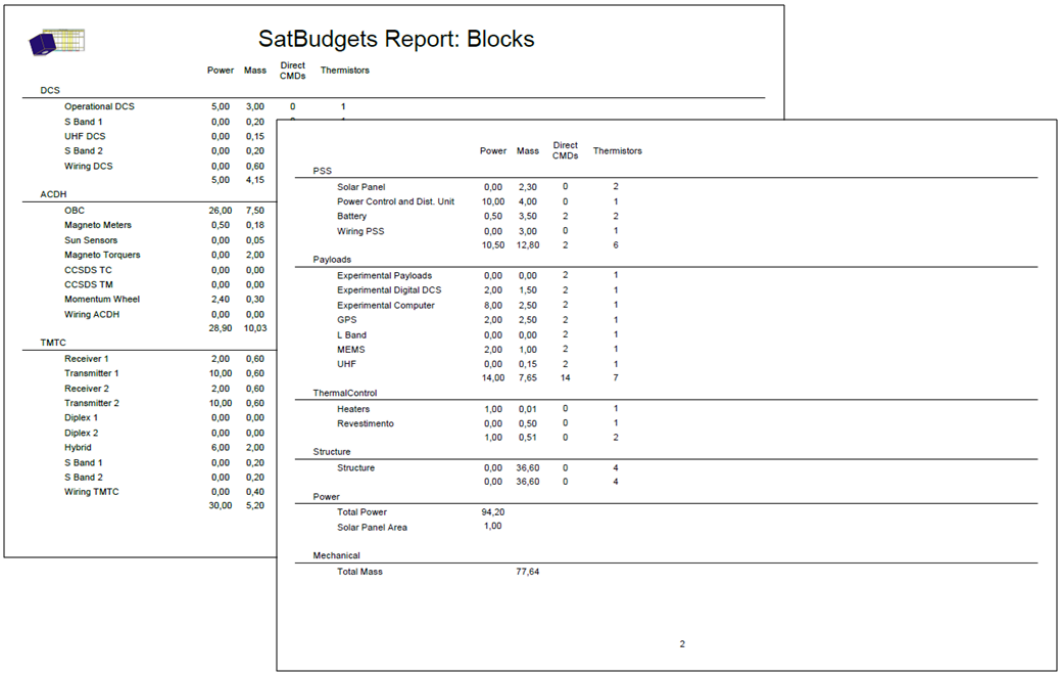
Figura 5.14 - Regra da Área do Painel Solar Convertida para DROOLS

5.3.6 Geração do Relatório

A última etapa realizada pela ferramenta é a geração do relatório contendo o balanço e sua apresentação aos responsáveis pelo sistema para que seu resultado seja avaliado. A responsável por esta etapa é a classe REPORT. Se o resultado do balanço estiver conforme planejado, pode ser iniciada a fase de produção do sistema. Caso contrário, alterações no modelo devem ser feitas e novos balanços gerados até que o resultado desejado seja atingido.

A Figura 5.15(a) apresenta um exemplo do relatório gerado pela ferramenta SatBudgets. A ferramenta permite que o engenheiro de sistemas visualize o relatório através de um visualizador ou salve o relatório nos formatos PDF, XLS e HTML. Se a opção escolhida for salvar o relatório, o mesmo será salvo na mesma pasta onde se encontra o modelo do sistema.

A Figura 5.15(b) apresenta um trecho do relatório gerado pela ferramenta SatBudgets onde o balanço mecânico gerado a partir do modelo não apresenta nenhum problema com relação a especificação/regras definidas. Já a Figura 5.15(c) apresenta um trecho do relatório onde o balanço mecânico contém problema, neste exemplo a massa ultrapassa o limite definido para o projeto. Quando ocorre algum problema no cálculo do balanço, é fornecido destaque ao valor referente ao problema.



SatBudgets Report: Blocks

	Power	Mass	Direct CMDs	Thermistors
DCS				
Operational DCS	5,00	3,00	0	1
S Band 1	0,00	0,20		
UHF DCS	0,00	0,15		
S Band 2	0,00	0,20		
Wiring DCS	0,00	0,60		
	5,00	4,15		
ACDH				
OBC	26,00	7,50		
Magneto Meters	0,50	0,18		
Sun Sensors	0,00	0,05		
Magneto Torquers	0,00	2,00		
CCSDS TC	0,00	0,00		
CCSDS TM	0,00	0,00		
Momentum Wheel	2,40	0,30		
Wiring ACDH	0,00	0,00		
	28,90	10,03		
TMTC				
Receiver 1	2,00	0,60		
Transmitter 1	10,00	0,60		
Receiver 2	2,00	0,60		
Transmitter 2	10,00	0,60		
Diplex 1	0,00	0,00		
Diplex 2	0,00	0,00		
Hybrid	6,00	2,00		
S Band 1	0,00	0,20		
S Band 2	0,00	0,20		
Wiring TMTC	0,00	0,40		
	30,00	5,20		

	Power	Mass	Direct CMDs	Thermistors
PSS				
Solar Panel	0,00	2,30	0	2
Power Control and Dist. Unit	10,00	4,00	0	1
Battery	0,50	3,50	2	2
Wiring PSS	0,00	3,00	0	1
	10,50	12,80	2	6
Payloads				
Experimental Payloads	0,00	0,00	2	1
Experimental Digital DCS	2,00	1,50	2	1
Experimental Computer	8,00	2,50	2	1
GPS	2,00	2,50	2	1
L Band	0,00	0,00	2	1
MEMS	2,00	1,00	2	1
UHF	0,00	0,15	2	1
	14,00	7,65	14	7
ThermalControl				
Heaters	1,00	0,01	0	1
Revestimento	0,00	0,50	0	1
	1,00	0,51	0	2
Structure				
Structure	0,00	36,60	0	4
	0,00	36,60	0	4
Power				
Total Power	94,20			
Solar Panel Area	1,00			
Mechanical				
Total Mass		77,64		

2

(a) Balanço Gerado pela ferramenta SatBudgets

Mechanical	
Total Mass	77,64

(b) Exemplo de Balanço - OK

Mechanical	
Total Mass	86,94

(c) Exemplo de Balanço - NOK

Figura 5.15 - Balanço Gerado pela ferramenta SatBudgets

Finalizado o processamento da ferramenta SatBudgets, cabe ao grupo responsável pelo projeto tomar as decisões finais. Se o projeto deve entrar em execução/produção ou se alguma alteração deve ser feita no modelo do projeto.

6 CONSIDERAÇÕES FINAIS

Os sistemas espaciais requerem bons princípios de engenharia de sistemas para lidarem com as questões de crescente complexidade do sistema. Logo um bom planejamento pode evitar erros que podem comprometer todo o empreendimento.

Neste sentido, foi utilizada neste trabalho uma nova abordagem para projeto conceitual de satélites empregando MDE e visando atender melhor os aspectos de engenharia de requisitos de projeto. Esta abordagem permite reutilizar informações em diferentes projetos de satélites, bem como facilitar a integração e gestão das atividades de engenharia de sistemas.

A aplicação desta abordagem neste trabalho foi baseada no uso da linguagem SysML para modelagem de satélites e na ferramenta proposta SatBudgets para dar apoio à fase de projeto conceitual.

Partindo da modelagem SysML, a proposta da ferramenta SatBudgets é gerar automaticamente balanços (mecânico, elétrico, entre outros) de projeto de satélite para mostrar a viabilidade de construção do sistema ou a necessidade de alterações no projeto do sistema. Um estudo de caso foi aplicado ao projeto do satélite universitário ITASAT.

Neste contexto, MDE e SatBudgets mostraram-se adequadas para serem adotadas em projetos futuros de satélites. A ideia de desenvolvimento da ferramenta SatBudgets pode ainda ser aplicada em outras áreas, por exemplo em projetos de veículos lançadores como já demostrado com o IAE (Instituto de Atividades Espaciais).

6.1 Trabalhos Futuros

Este trabalho poderá ser posteriormente estendido implementando alguns dos tópicos apresentados a seguir:

- Armazenamento de regras de negócio do sistema em uma base de dados;
- Integração a um ambiente de simulação de satélites;

- A implementação de *web services* para processamento especializado de regras e/ou integração de aplicativos heterogêneos de engenharia de sistemas de satélites;
- O desenvolvimento de um ambiente colaborativo para fornecer dados e suporte via web;
- Implementação de uma classe para ativação do módulo correspondente aos modos de operação do satélite;
- Inclusão de gráficos no relatório para facilitar a visualização dos balanços, e
- Engenharia direta e reversa de sistemas de satélites.

REFERÊNCIAS BIBLIOGRÁFICAS

AGÊNCIA ESPACIAL BRASILEIRA (AEB); INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS (INPE); INSTITUTO TECNOLÓGICO DE AERONÁUTICA (ITA). **ITASAT**. 2005. Disponível em: <www.itasat.ita.br>. Acesso em: abr. de 2009. 16

____. **ITASAT satellite specification**. 2008. U1100-SPC-001 REV.00. 17

ALLEN, R.; GARLAN, D. A formal basis for architectural connection. **ACM Transactions on Software Engineering and Methodology**, v. 6, n. 3, p. 213–249, Sept. 1997. 11

ALMEIDA, F. J. Research and choice of a methology for the conceptual design. **Revista de Ciência & Tecnologia**, v. 8, n. 16, p. 31–42, Dez. 2000. 1, 5

BURNETTE, E. **Eclipse IDE**: pocket guide. 1. ed. Sebastopol: O'Reilly Media, 2005. 128 p. ISBN 9780596100650. 32, 33

CARDOSO, A. C.; MARQUES, A. S. **Sistemas baseados em regras**. Sistemas Integrados de Apoio a Decisão (SIAD), 2007. Acesso em: mai. de 2009. Disponível em: <http://mestradosiad.blogspot.com/2007/11/sistemas-baseados-em-regras.html>. 14

DANCIU, T. **The JasperReports ultimate guide**. 2002. Disponível em: <http://www.jaspersoft.com/jasperreports-ultimate-guide>. Acesso em: set. de 2009. 35

FERNANDES, D. B. **Análise de sistemas orientada ao sucesso**: por que os projetos atrasam? 1. ed. Rio de Janeiro: Ciência Moderna, 2005. 272 p. ISBN 8573934298. 7

FONDEMENT, F.; SILAGHI, R. Defining model driven engineering processes. In: INTERNATIONAL WORKSHOP IN SOFTWARE MODEL ENGINEERING (WISME), 3., INTERNATIONAL CONFERENCE ONE THE UNIFIED MODELING LANGUAGE (UML), 7., 2004, Lisbon. **Proceedings...** Lausanne: Swiss Federal Institute of Technology, 2004. p. 7. 8

FRIEDENTHAL, S.; MOORE, A.; STEINER, R. **A practical guide to SysML**. 1. ed. Oxford: Morgan Kaufmann, 2008. 560 p. ISBN 978-0-12-374379-4. 19

GARLAN, D. An introduction to the aesop system. Carnegie Mellon University, School of Computer Science, Pittsburgh, p. 12, Jul. 1995. Disponível em: <http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/htmlaesop-overview.ps>. Acesso em: sept. 2009. 11

GORLICK, M. M.; RAZOUK, R. R. Using weaves for software construction and analysis. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE13 '91), 13., 1991, Austin, Texas, USA. **Proceedings...** Los Alamitos, CA, USA: IEEE Computer Society Press, 1991. p. 23–34. 11

INTERNATIONAL COUNCIL ON SYSTEMS ENGINEERING (INCOSE). **INCOSE Foundation**. 2009. Disponível em: <http://www.incose.org>. Acesso em: jul. de 2009. 19

JASPERSOFT. **iReports Tutorials & Help**. JasperSoft Corp., 2009. Disponível em: <http://jasperforge.org//website/ireportwebsite/IR%20Website/ir_documentation.html?header=project&target=ireport>. Acesso em: nov. de 2009. 37

JDOM. **JDOM Documentation**. 2009. Disponível em: <http://www.jdom.org/downloads/docs.html>. Acesso em: out. de 2009. 34

LARSON, W. J.; WERTZ, J. R. **Space mission analysis and design**. 3. ed. Dordrecht, Netherlands: Microcosm Press, 1999. 969 p. ISBN 9780792359012. 15

LEONOR, B. B. F.; SANTOS, W. A. dos; STEPHANY, S. A model-driven requirements engineering approach to conceptual satellite design. In: BRAZILIAN SYMPOSIUM ON AEROSPACE ENGINEERING AND APPLICATIONS / CTA DLR WORKSHOP ON DATA ANALYSIS AND FLIGHT CONTROL, 3., 2009, São José dos Campos, SP, Brasil. **Anais...** São José dos Campos: CTA/ITA/AAB/DLR, 2009. p. 10. 1

LESCOT, J. **TOPCASED v3**: installation guide. Sep. 2009. 32

LUCKHAM, D. C.; KENNEY, J. J.; AUGUSTIN, L. M.; VERA, J.; MANN, W.; BRYAN, D. Specification and analysis of system architecture using rapide. **IEEE Transactions on Software Engineering**, v. 21, n. 4, p. 336–355, Apr. 1995. 11

MAGEE, J.; KRAMER, J. Dynamic structure in software architectures. **ACM SIGSOFT Software Engineering Notes**, Association for Computing Machinery (ACM), New York, v. 21, n. 6, p. 3–14, Nov. 1996. ISSN 0163-5948. 11

MEDVIDOVIC, N.; OREIZY, P.; ROBBINS, J. E.; TAYLOR, R. N. Using object-oriented typing to support architectural design in the c2 style. In: ACM SIGSOFT'96 SOFTWARE ENGINEERING NOTES. **Proceedings...** New York, NY, USA: Association for Computing Machinery (ACM), 1996. v. 21, n. 6, p. 24–32. ISSN 0163-5948. 11

MEDVIDOVIC, N.; TAYLOR, R. N. A classification and comparison framework for software architecture description languages. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, v. 26, n. 1, p. 70–93, Jan. 2000. ISSN 0098-5589. 5, 9, 11, 12, 83, 85, 87, 89

MEIJERS, M. A. H. A.; HAMANN, R. J.; ZANDBERGEN, B. T. C. Applying systems engineering to university satellite projects. In: ANNUAL INTERNATIONAL SYSMPOSIUM INCOSE, 14., 2004, Toulouse. **Proceedings...** Toulouse, 2004. p. 11–19. 2

MILESKI, A. M. **Brasil desenvolve satélite universitário**. Jan. 2007. Disponível em: <http://www.defesanet.com.br/zz/space_itasat.htm>. Acesso em: mar. de 2009. 17

NASA. **Artificial satellites**. 2009. Disponível em: <http://www.nasa.gov/worldbook/artificial_satellites_worldbook.html>. Acesso em: apr. de 2009. 15

NORTHOVER, S.; WILSON, M. **SWT**: the standard widget toolkit. 1. ed. Boston: Addison-Wesley Professional, 2004. 592 p. ISBN 0-321-25663-8. 33

OMG. **SysML specification**. Needham: OMG, May 2006. 260 p. 39

____. **SysML**. 2009. Disponível em: <http://www.omgsysml.org/>. Acesso em: abr. de 2009. 2, 19

____. **UML**. 2009. Disponível em: <http://www.uml.org/>. Acesso em: abr. de 2009. 2, 19

RICH, E.; KNIGHT, K. **Artificial intelligence**. 2. ed. New York: Mcgraw-hill, 1991. 621 p. 13

RIEMENSCHNEIDER, R.; MORICONI, M. **Introduction to SADL 1.0 a language for specifying software architecture hierarchies**. Mar. 1997. 33 p. Disponível em: <http://www.csl.sri.com/papers/sadl-intro/>. Acesso em: ago. de 2009. 11

RUSSEL, S.; NORVIG, P. **Artificial intelligence**: a modern approach. Upper Saddle River, Nj: Prentice-hall, 1995. 932 p. (Prentice Hall Series in Artificial Intelligence). 13

SANTOS, M. F. **Sistemas de conhecimento**. Portugal: Universidade do Minho, 2006. 13

SANTOS, W. A. dos; LEONOR, B. B. F.; STEPHANY, S. A knowledge-based and model-driven requirements engineering approach to conceptual satellite design. **Lecture Notes in Computer Science - Conceptual Modeling - ER 2009**, v. 1, p. 487–500, 2009. 1

SCHMIDT, D. C. Model-driven engineering. **IEEE Computer Society**, p. 25–31, Feb. 2006. 2, 8

SHAW, M.; DELINE, R.; ZELESNIK, G. Abstractions and implementations for architectural connections. In: INTERNATIONAL CONFERENCE ON CONFIGURABLE DISTRIBUTED SYSTEMS, 3., 1996, Annapolis. **Proceedings...** Annapolis, Maryland: IEEE, 1996. p. 2–10. 11

SMITH, P.; DAWDY, A.; TRAFTON, T.; NOVAK, R.; PRESLEY, S. **Concurrent design at aerospace**. Los Angeles: Crosslink, 2001. 5-11 p. 1

SOUZA, A. D. de; VIVI, R. de O.; GOTO, S. S. F. **Linguagens de descrição de arquiteturas - ADLs**. Campinas: Instituto de Computação - Unicamp, 2004. Disponível em: <www.ic.unicamp.br/~rodolfo/Cursos/mc722/2s2004/g08-adl-texto.pdf>. Acesso em: mar. de 2009. 10

SOUZA, F. J. Bases de dados e engenharia de conhecimento. **Revista TI**, 1999. Disponível em: <http://www.timaster.com.br/revista/artigos/main_artigo.asp?codigo=35>. Acesso em: mar. de 2010. 3

SOUZA, P. N. de. **Curso introdutório em tecnologia de satélites**. São José dos Campos: INPE, 2003. (INPE-9605-PUD/126). 15

SPíNOLA, R. O.; ÁVILA, A. L. Introdução à engenharia de requisitos. **Engenharia de Software Magazine**, Edição Especial, n. 1, p. 46–52, 2007. 6, 7, 28

VESTAL, S. **MetaH programmer's manual**. Minneapolis, MN: Honeywell Technology Center, Apr. 1996. Technical Report. 11

WEILKIENS, T. **Systems engineering with SysML/UML**: Modeling, analysis, design. 1. ed. Oxford: Morgan Kaufmann, 2007. 307 p. ISBN 978-0-12-374274-2. 21

ANEXO A - A MODEL-DRIVEN REQUIREMENTS ENGINEERING APPROACH TO CONCEPTUAL SATELLITE DESIGN

Este trabalho, desenvolvido durante a pesquisa de Mestrado em Computação Aplicada, foi apresentado durante o evento **2009 Brazilian Symposium on Aerospace Engineering and Applications / 3rd CTA-DLR Workshop on Data Analysis and Flight Control** que ocorreu no período de 14-16 de setembro de 2009, em São José dos Campos - SP, no Instituto Tecnológico de Aeronáutica - ITA, na sessão "S11 - Systems Engineering and Project Management".

A MODEL-DRIVEN REQUIREMENTS ENGINEERING APPROACH TO CONCEPTUAL SATELLITE DESIGN

Bruno Bustamante Ferreira Leonor, brunobfl@yahoo.com.br
Walter Abrahão dos Santos, walter@dss.inpe.br
National Space Research Institute – INPE
São José dos Campos, SP -Brazil

Stephan Stephany, stephan@lac.inpe.br
National Space Research Institute – INPE
São José dos Campos, SP -Brazil

Abstract*. Satellite systems are growing even more complex, making technical issues a significant driver of costs. The increasing complexity of these systems makes requirements engineering activities both more important and more difficult. Additionally, nowadays strong market competition drive companies to improve the efficiency with which they design and manufacture space products and systems. This imposes a heavy burden on systems-of-systems engineering skills and particularly on requirements engineering which is an important phase in a system's life cycle. A poorly attained requirement engineering approach may cause design failures, cost overrun and delays. One solution is to underpin the preliminary conceptual satellite design with computer-based information reuse and integration to deal with interdisciplinary nature of this problem domain. This can be attained by taking a model-driven engineering approach (MDE), in which models are the main artifact during system development. MDE is an emergent approach that tries to address system complexity by the intense use of models. This work outlines the use of SysML (Systems Modeling Language) for requirements engineering, more specifically requirements management and traceability. It is intended to use this approach in the conceptualphase of a semi-professional satellite system called ITASAT, currently being built by INPE and some Brazilian universities.*

Keywords*: Model-Driven Engineering, ADL, SysML, Satellite.*

1. INTRODUCTION

Space systems are complex systems designed to perform specific functions for a specified design life. Satellite projects, for instance, demand lots of resources, from human to financial, as well accounting for the impact they play on society. This requires good planning in order to minimize errors and not jeopardize the whole mission.

Therefore satellite conceptual design plays a key role in the space project lifecycle as it caters for specification, analysis, design and verification of systems without actually having a single satellite built. Conceptual design maps client needs to product use functions and is where functional architecture (and sometimes the physical architecture) is decided upon.

Moreover, the lack of a clear vision of the satellite architecture hinders team understanding and communication, which in turn often increases the risk of integration issues. The conceptual satellite design phase has been lacking efficient support.

SysML is a new systems modeling language that supports specification, analysis, design, verification and validation of a broad range of complex systems (Soares and Vrancken, 2007). This work proposes to employ SysML as a satellite architecture description language in order to enable information reuse between different satellite projects. Additionaly, this approach facilitates knowledge integration and management over systems engineering activities. One of them is requirements engineering, more specifically requirements management and tracebility. This is an important phase in the life cycle of satellite systems.

This work shows the main advantages of having user requirements being graphically modeled, their relationships explicitly mapped, and system decomposition considered in the early system development activities. In addition, requirements traceability is enhanced by using the SysML Requirements tables. The approach is illustrated by a list of user requirements for ITASAT, semi-professional satellite system, currently being built by INPE and some Brazilian universities.

This work is organized as follows. Section 2 presents a short introdution to satellites and to SysML as an architecture description language. Section 3 shows the SysML satellite modeling. Section 4 covers the SysML satellite requirements enginerring. Section 5 describes further future work. Finally, Section 6 summarizes this research report.

2. BACKGROUND

This background section presents an overview of the satellite and SysML which will be important for the paper context.

2.1. Overview to satellites

A satellite has generally two main parts:

- The bus or platform where the main supporting subsystems reside and;
- The payload, the part that justifies the mission.

As an example, Figure 1 shows the Equatorial Atmosphere Research Satellite (EQUARS), a typical scientific satellite envisaged by INPE (Chamon, 2004). Its various payloads are aimed at studying the equatorial low, middle and upper atmosphere-ionosphere. To achieve this objective, the satellite shall be placed in a near equatorial orbit at 750 km Low Earth Orbit (LEO) altitude, and 20 degrees of inclination, in order to view low latitude regions of the Earth.

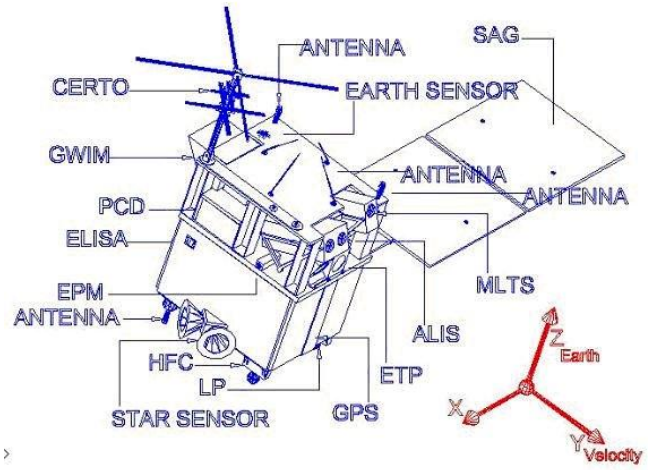


Figure 1. EQUARS satellite (Chamon, 2004)

A typical satellite bus has a series of supporting subsystems as depicted in Figure 2. The satellite system is built around a system bus also called the On-Board Data Handling (OBDH) bus. The main On-Board Computer (OBC) is the bus master which here incorporates functionalities from command and telemetry units for simplicity. The clients on the system bus are the various satellite subsystems and the payloads. Satellite subsystems may have their own computer and may even have an internal bus. The flight software manages all the spacecraft functionalities into a coordinated fashion.

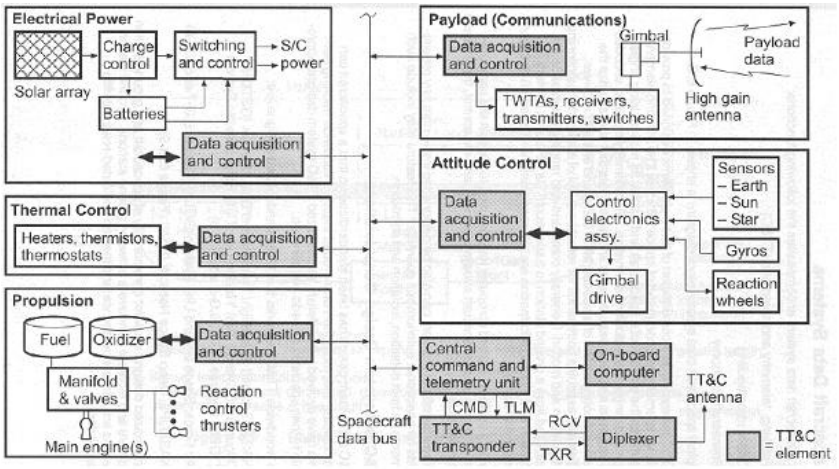


Figure 2. Block diagram of a typical satellite (de Souza, 2002)

Any satellite is composed by subsystems pertaining to either paylod or the bus (also called service module or platform). The payload depend on the satellite mission. The control segment on the ground monitors and controls the platform subsystems. These are typically the following:Structure of the satellite:

- Power;
- Propulsion;

- Stabilization and Attitude Control;
- Thermal Control;
- Environmental Control and;
- Telemetry, Tracking and Command.

2.2. SysML as an Architecture Description Language

System modeling based on an architecture description language is a way to keep the engineering information within one information structure. Using an architecture description language is a good approach for the satellite systems engineering domain.

Architectures represent the elements implementing the functional aspect of their underlying products. The physical aspect is sometimes also represented, for instance when the architecture represents how the software is deployed on a set of computing resources, like a satellite.

SysML is a domain-specific modeling language for systems engineering and it supports the specification, analysis, design, verification and validation of various systems and systems-of-systems (SysML, 2009). It was developed by the Object Management Group (OMG) (OMG, 2009) in cooperation with the International Council on Systems Engineering (INCOSE) (INCOSE, 2009) as a response to the request for proposal (RFP) issued by the OMG in March 2003. The language was developed as an extension to the actual standard for software engineering, the Unified Modeling Language (UML) (UML, 2009) also developed within the Object Management Group consortium.

Basically, SysML is used for representing system architectures and linking them with their behavioral components and functionalities. By using concepts like Requirements, Blocks, Flow Ports, Parametric Diagrams and Allocations, it is simple to achieve a profitable way to model systems. A comparison between SysML1.0 to UML2.0 in term of re-use is presented in Figure 3. It summarizes the various diagrams available in SysML (SysML, 2009).

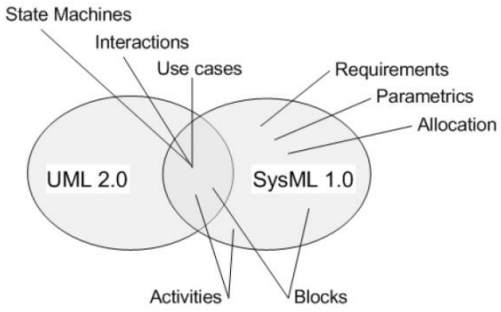


Figure 3. Relationship between UML and SysML (SysML, 2009)

As shown in Figure 4, Requirements, Parametrics and Allocations are new diagrams available only in SysML. Activity and Block diagrams are reused from UML2.0 and extended in SysML. Finally, State Machines, Interactions and Use cases are reused from UML2.0 without modifications.

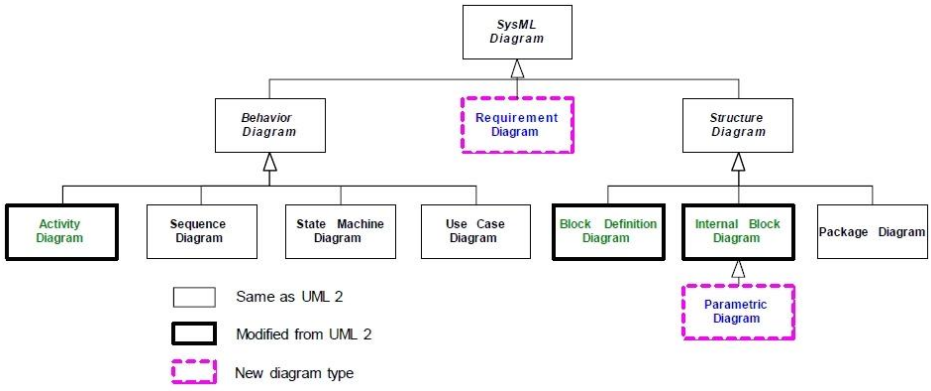


Figure 4. SysML diagram taxonomy (SysML)

This work explores some of the SysML capabilities through an example, the ITASAT student satellite system (Carvalho *et al*, 2008). The application of SysML presented in this work covers only some the diagrams available in SysML due to paper scope and page restrictions.

3.CONCEPTUAL SATELLITE DESIGN VIA SysML

Systems Engineering attacks the problem of design complexity of engineering products as it grows larger, more complex and are required to operate as part of a system.

The approach taken is formal and systematic since the great complexity requires this rigor. Another feature of systems engineering is its holistic view and it involves a top-down synthesis, development, and operation. This suggests the decomposition of the system into subsystems and further into components (Dieter, 1991).

3.1.Motivation for the Satellite SysML Modeling

Space Systems Engineering is a subclass of the previous mentioned in the sense that it is primarily concerned with space systems, e.g. satellite systems. Therefore it deals with the development of systems, including hardware, software, man-in-the-loop, facilities and services for space applications.

The satellite conceptual stage follows the transformation of customer needs into product functions and use cases, and precedes the design of these functions across the space engineering disciplines (for example, mechanical, electrical, software, etc.).

Model-Driven Engineering (MDE) is the systematic use of models as primary engineering artifacts throughout the engineering lifecycle (Schmidt, 2006). MDE can be applied to software, system, and data engineering.

MDE technologies, with a greater focus on architecture and corresponding automation, yield higher levels of abstraction product development. This abstraction promotes simpler models with a greater focus on problem space. Combined with executable semantics this elevates the total level of automation possible. This work argues that MDE is quite suitable for information reuse and integration as will be shown later.

3.2.The SysML Modeling Approach

SysML allows incrementally refinable description of conceptual satellite design and product architecture. This helps systems engineers which are concerned with the overall performance of a system for multiple objectives (e.g. mass, cost, and power). The systems engineering process methodically balances the needs and capabilities of the various subsystems in order to improve the systems performance.

SysML elements in the design represent abstractions of artifacts in the various engineering disciplines involved in the development of the system. The design represents how these artifacts collaborate to provide the product functionalities. The size, volume, and mass constraints often encountered in satellite development programs, combined with increasing demands from customers to get more capability into a given size, make systems engineering methods particularly important for this domain.

This paper explores some of the diagrams available in SysML through the example of the ITASAT satellite system by basically, exploring the block diagram and top-level requirement diagram, both shown in short detail.

SysML diagrams allow information reuse since they can be emploied in other similar satellite projects by adaptation and dealing with project variabilities. An exploration of these features for the on-board software design of satellites is shown in (dos Santos, 2008). As an example, take the use case diagram which describes a specific use of the system by a particular actor, i.e., some outside stimulus. The use case diagram represents a fully factored model since use cases are decomposed to find pieces that can be reused in multiple use cases. Its fragments are then composed into use cases using relationships like “extends” and “includes” precluding redundancies.

SysML allows the utilization of use case diagrams which were inherited from the UML without changes (Balmelli, 2007). Use case diagrams are emploied to describe systems functionalities by its actors looking forward to achieving a system goal. The use case diagram can be seen as functionalities and\or capabilities which are implemented by means of interaction between actors and the use cases. The use case diagram has been widely applied to specify system requirements.

The interaction between ITASAT actors and some key use cases is shown in Figure 5. This diagram depicts five actors and how they relate to the use cases that they trigger in the high-level system view. The figure still describes schematically the composition of a series of low-level use cases hierarchically modeled by employing an <<include>> dependancy relationship between them. SysML also allows the representation of test use cases which will further explored in the validation, verification and testing project phases. Figure 5 depicts, as an example, the *Test On-Board Management Functions* use case and how are its <<include>> dependancies are related to other two test use cases, *Test Other On-Board Functions* and, *Test Power Supply Functions*.

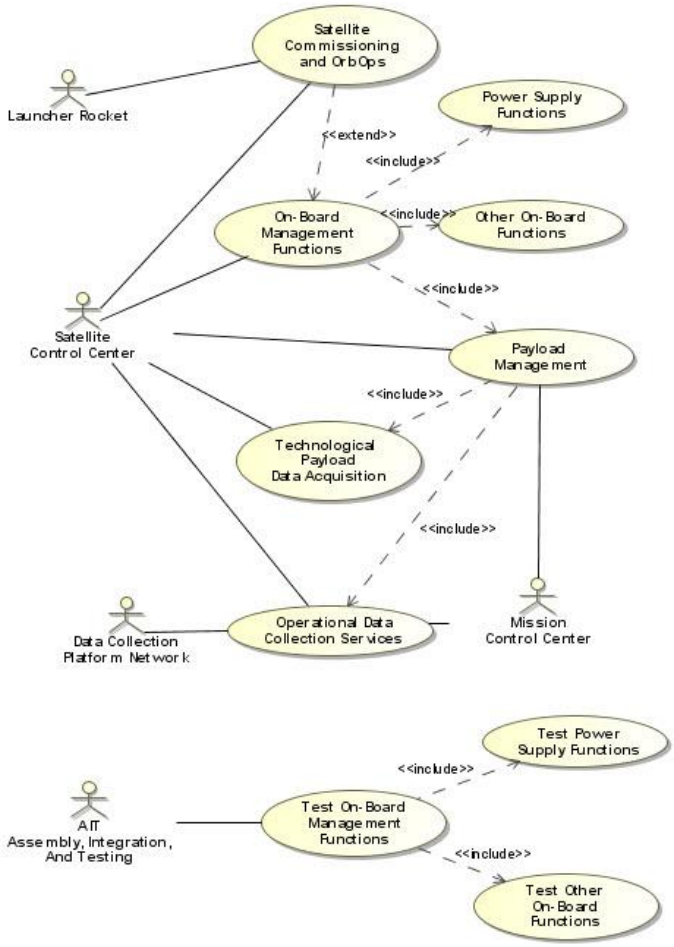


Figure 5. ITASAT high-level use case to specify system requirements (Carvalho *et al*, 2008)

The SysML block diagram is used to show features and high-level relationships. It is used to allow systems engineer to separate basically the responsibilities of the hardware team from the software team. Figure 6 shows the various ITASAT blocks and their interdependencies.

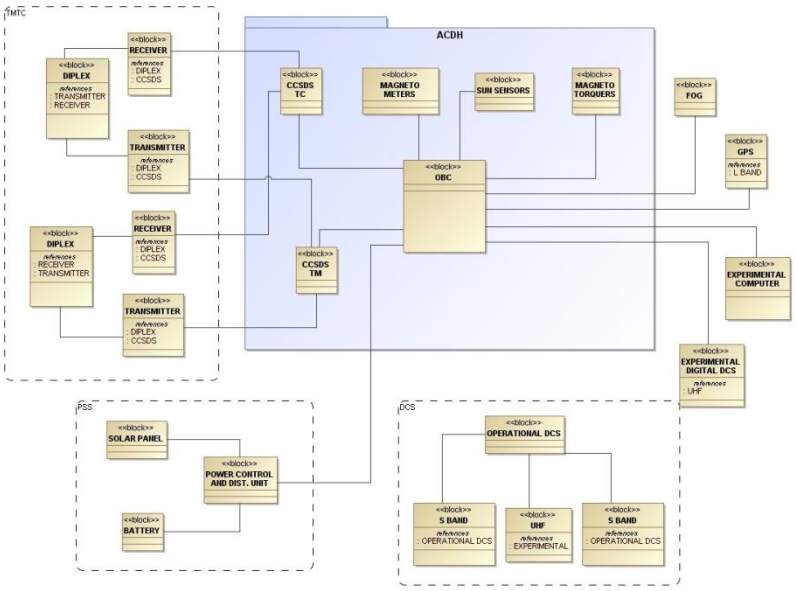


Figure 6. The ITASAT satellite SysML block diagram (Carvalho *et al*, 2008)

The requirements diagram plays a key role into the SysML model as requirements present in this diagram can also appear in other SysML diagrams linking the problem and solution spaces. Furthermore, the requirements diagram notation provides a means to show the relationships among requirements including constraints, see Figure 7 for details. This topic is of high importance to this work hence it is further developed in the next section.

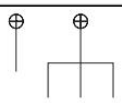
Diagram Element	Notation	Description
Containment Path		The containment relationship is used to represent how requirements are contained in specifications (packages), or how a complex requirement can be partitioned into a set of simpler requirements without adding or changing their meaning.
Derivation Path	«deriveReqt»	A derive relationship occurs between a source requirement and a derived requirement, based on analysis of the source requirement.
Satisfaction Path	«satisfy»	A satisfy relationship is used to assert that a model element corresponding to the design or implementation satisfies a particular requirement.
Verification Path	«verify»	A verify relationship is used between a requirement and a test case or other named element to indicate how to verify that the requirement is satisfied.
Refinement Path	«refine»	The refine relationship is used to reduce ambiguity in a requirement by relating it to another model element that clarifies the requirement.
Trace Path	«trace»	A trace relationship is a general-purpose way to relate a requirement and any other model element, useful for relating requirements to documents, etc.
Copy Path	«copy»	The copy relationship relates a copy of a requirement to the original requirement, to support reuse of requirements.

Figure 7. Relationships used in a SysML requirement diagram (OMG–SysML, 2009)

4. THE MODEL-DRIVEN REQUIREMENTS ENGINEERING APPROACH

The process of requirements engineering involves various key activities, such as elicitation, specification, prioritization and management of requirements. This section shows how SysML can enable a model-driven requirements engineering approach to conceptual satellite design.

The SysML standard identifies relationships that enable the modeler to relate requirements to other requirements as well as to other model elements. These include relationships for defining a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, and refining requirements (SysML, 2009).

Figure 8 shows a simplified view of the ITASAT requirement tree structure (Carvalho *et al*, 2008). It also shows how a constraint is attached to a low-level requirement and how traceability may be established.

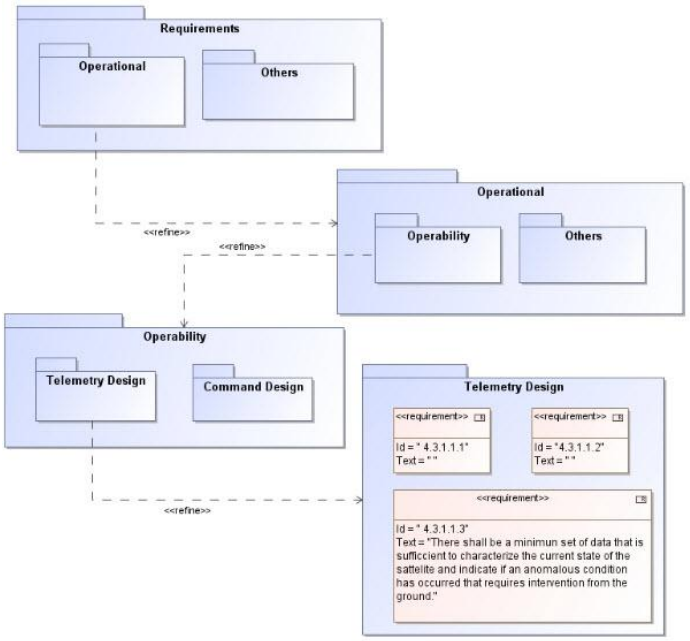


Figure 8. Requirements tree structure for the ITASAT satellite (Carvalho *et al*, 2008)

After top-level requirements are ellicited, then starts the decomposition of every system requirement into progressively lower levels of design. This is done by defining the lower-level functions which determine how each function must be performed. Allocation assigns the functions and its associated performance requirements to a lower level design element. Decomposing and allocation starts at the system level, where requirements derive directly from mission needs, and proceeds through segment, subsystem, and component design levels (Larson and Wertz, 2004). This process must also warrant closure at the next higher level meaning that satisfying lower-level requirements warrants performance at the next level. Additionally, it roundtrips all requirements tracing back to satisfying mission needs.

Managing requirements is the capability of tracing all system components to output artifacts that have resulted from their requirement specifications (forward tracing) as well as the capability of identifying which requirement has generated a specific artifact or product (backward tracing) (Pressman, 1995).

The great difficulty on tracing requirements is responding the following questions: *What to track?* and *How to track?*. One can say that a requirement is traceable when it is possible to identify *who has originated it, why it exists, which are the requirements related to it? how is it related to other project information*. These information is used to identify all requirement\elements affected by Project changes. The specification of requirements can facilitate the communication between the various project stakeholder groups.

There are several published works on requirement engineering and the most common way they employ to requirement tracking is by posing basic questions about the underlying domain (Aurum, 2005). Unfortunately, such questionaire does not offer generally any classification on the sufficient elements in order to identify all model elements.

By using a SysML requirements diagram, system requirements can be grouped, which contributes to enhance project organization showing explicitly the various relationship types between them (Soares and Vrancken, 2008). These include relationships for defining requirements hierarchy or containment, deriving requirements, satisfying requirements, verifying requirements and refining requirements (OMG-SysML, 2009). Moreover, the SysML requirements diagram can be emploied to standardize how requirements are documented following all their possible relationships. This can provide systems specification as well as be used for requirements modeling.

New requirements can be created during the requirement analysis phase and can be related to the existing requirements or complement the model. Figure 9 presents an excerpt from the ITASAT requirements diagram which utilizes the <<deriveReqt>> relationship type showing the derived *Satellite State* requirement from the source *Telemetry Design* requirement inside the Operability requirement SysML package. This allows, for example, a link between high-level (user oriented) and low-level (system oriented) requirements which contributes to explicitly relates the dependency of user requirements mapped into systems requirements.

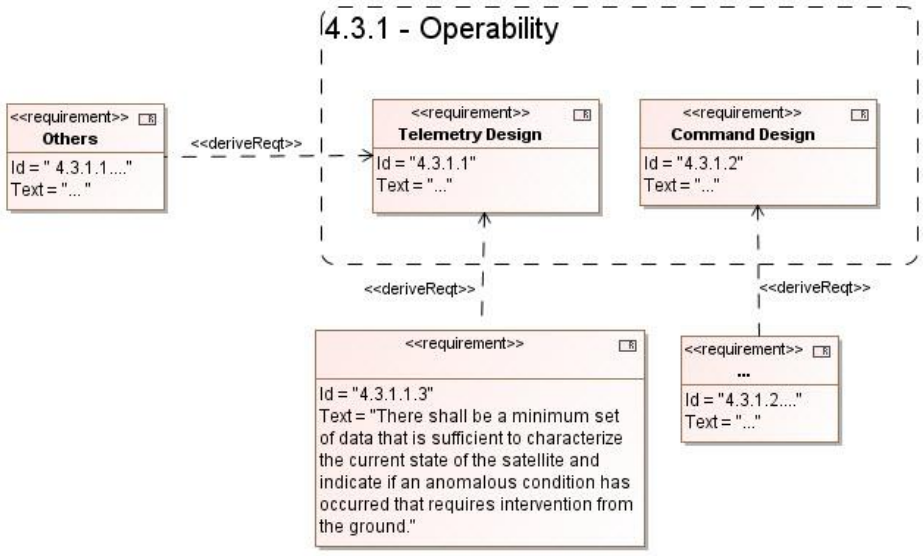


Figure 9. An excerpt of the ITASAT requirements diagram with a <<*deriveReqt*>> relationship (Carvalho *et al*, 2008)

Similarly, Figure 10 presents another excerpt from the ITASAT power subsystem requirements diagram which utilizes three relationships. Requirements are abstract classes with no operations neither attributes. Subrequirements are related to their "father" requirement by utilizing the <<containment>> relationship type. This is shown in Figure 10 as many subrequirements from the *Power Supply Requirements* requirement are connected employing <<containment>> relationships. The "father" requirement can be considered a package of embedded requirements.

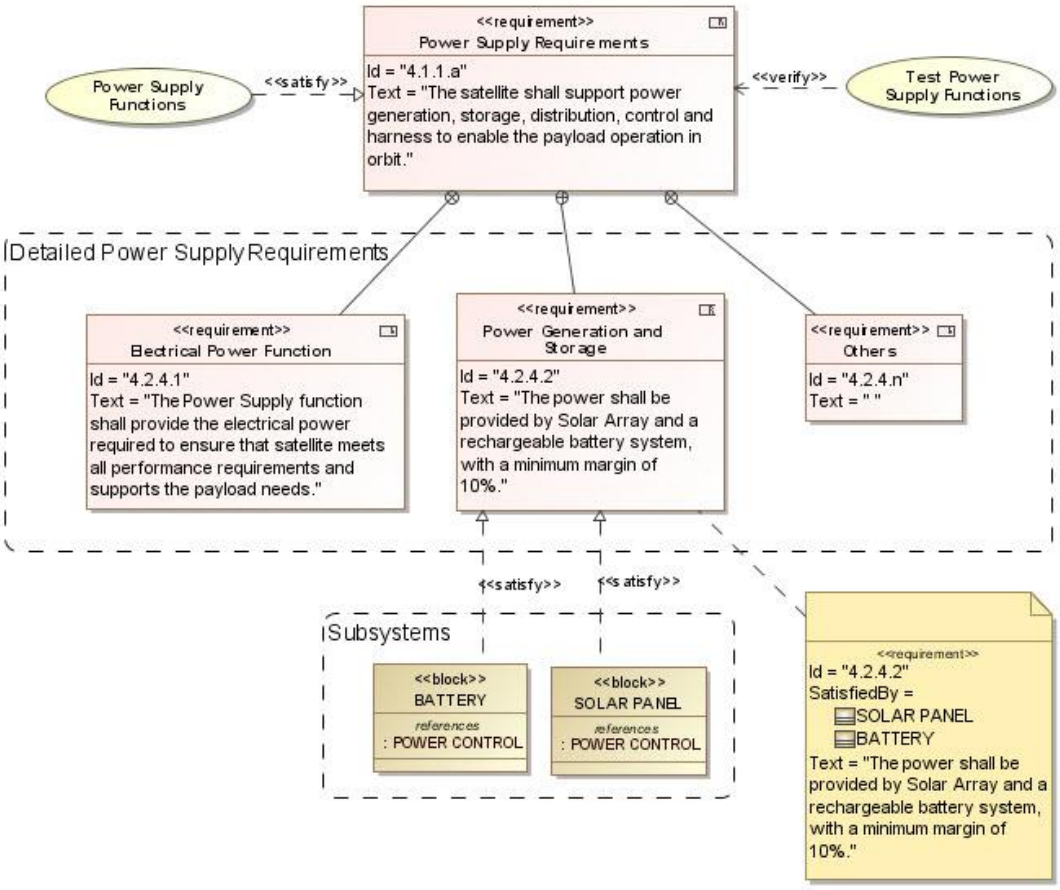


Figure 10. An excerpt of the ITASAT power subsystem requirements diagram with containment <<*satisfy*>> and <<*verify*>> relationships (Carvalho *et al*, 2008)

Additionally, Figure 10 presents the <<satisfy>> relationship type which shows how a model satisfies one or more requirements. It represents a dependency relationship between a requirement and a model element, in this case the *Power Supply Functions* use case is satisfied by the *Power Supply Requirements*. Finally, it is shown the <<verify>> relationship type where the *Test Power Supply Functions* test use case is verified by the functionalities provided by the *Power Supply Requirements*. This may include standard verification methods for inspection, analysis, demonstration or test.

Lastly, SysML allows requirements traceability by using tabular notations. This allows model elements to be traced in SysML via requirements tables which may contain fields like: identifier (ID), name, which requirement is related to it, what type of relationship is held among them.

One such SysML tabular notation for requirements traceability is shown in Figure 11 which is suitable for crossrelating model elements. The figure shows a requirement matrix table where cross-tracing is done between requirements, blocks defined in the ITASAT block diagram and high-level use cases. This table is quite important as it allows requirements traceability towards issues like:

- Understanding of requirements origin;
- Project scope management;
- Requirements change management;
- Impact analysis of requirement changes;
- Impact analysis of requirement test failures (i.e., if a test fails then a requirement may not be met);
- Verification if all system requirements are mapped into implementation and;
- Validation if the final application only performs what it was previously expected.

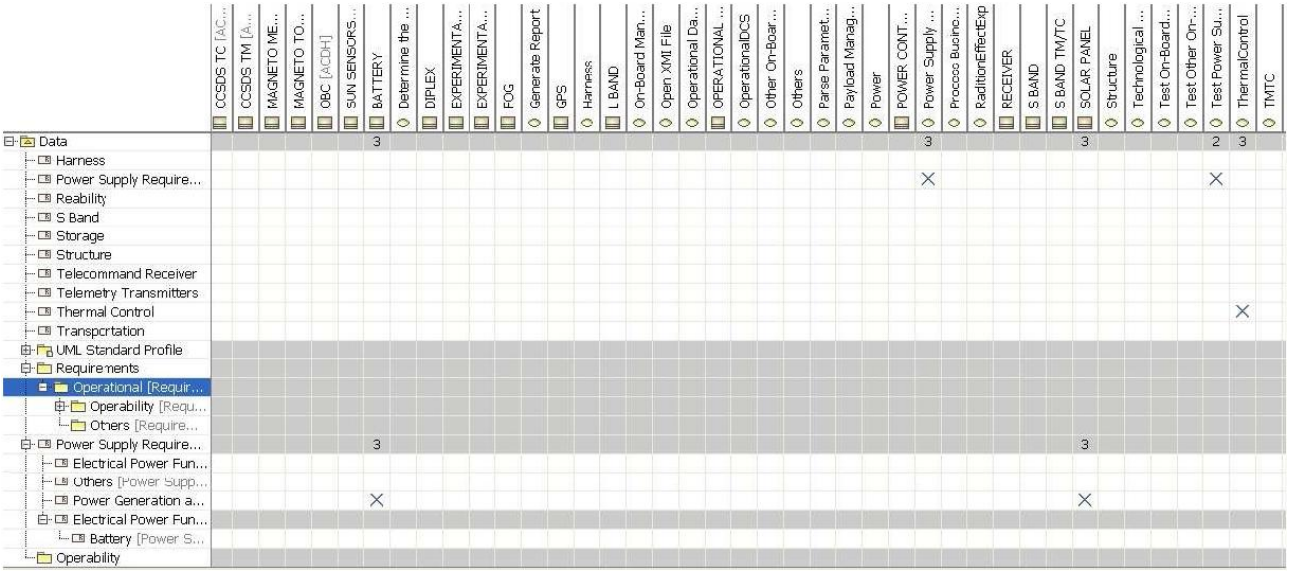


Figure 11. The tabular matrix notation used to display power related requirements and their relationships to other model elements (Carvalho *et al*, 2008)

Additionally, requirements can be traced also by navegating through SysML requirement diagrams on the anchors points shown in Figure 10 by means of stand-out notes. The anchors contain information like the relationship type and with which model element the requirement is related and vice-versa, given a model element it may reference all requirements related to this element. Doing so, it allows a quick and simple way to identify, prioritize and improve requirements traceability.

The resources provided by the SysML are by far beyond the capabilities here presented due to paper page limitation.

5. FUTURE WORK

Currently more complete ITASAT SysML modeling is also expected which include:

- Enhancing Block Diagram representation to model detailed subsystems and components, and ports describing their interfaces;
- Checking dependencies (e.g. analytical) between structural properties expressed using constraints and represented using the parametric diagram;
- Exploring features behavior modeling, namely interactions, state machine and activities;
- Employing SysML for providing a mechanism to relate different aspects of the model and to enforce traceability across it;
- Developement of a novel knowledge-based software tool, named SatBudgets, is being performed to support preliminary budgetings on conceptual satellite design which demands interdisciplinary skills and;
- Provide an interface for Eclipse IDE aggregation as well as for docking to an in-house Satellite Simulator.

6. CONCLUSIONS

Space systems requires strong systems engineering to deal with systems-of-systems complex issues, manufacturing demands and keep risks managable. Requirements Engineering for these systems is a difficult issue as when this is poorly performed, various problems may occur, such as failures, cost overrun and delays. A case study was presented in this work introducing the use of SysML satellite modeling for requirements engineering. This allows user requirements being graphically modeled, their relationships being explicitly mapped and, system decomposition being considered in the early system development activities. By employing SysML as a satellite architecture description language, it also enables information reuse between different satellite projects as well as it facilitates knowledge integration and management on systems engineering activities. This work will be further extended to implement MDE automation concepts into the ordinary workflow of satellite systems engineering.

7. REFERENCES

Aurum, A. W., 2005, “Engineering and Managing Software Requirements”. Springer Verlag, New York.

Balmelli, L., 2007, "An overview of the systems modeling language for products and systems development", Journal of Object Technology, vol. 6, No. 6, pp. 149-177.
Carvalho, T. and et al., 2008, " Itasat satellite specification. Technical report", INPE U1100-SPC-01 Internal Report.
Chamon, M., 2004, "Equars satellite specification. Technical report", INPE E30000-TSD-042 Internal Report.
de Souza, P. N., 2002, "Space Project Management ", CITS lecture notes. Slides - INPE – Brazilian National Space Research Institute.
Dieter, G. E., 1991, "Engineering Design - a Materials and Processing Approach", McGraw-Hill International Edition, New York.
dos Santos, W. A., 2008, "Adaptability, Reusability and Variability on Software Systems for Space On-Board Computing", ITA Ph.D. Thesis, S. J. Campos, Brazil.
INCOSE - International Council on Systems Engineering, 2009, http://www.incose.org.
Larson, W. J., Wertz, J. R., 2004, "Space Mission Analysis and Design". McGraw-Hill, El Segundo, USA, 3rd edition.
OMG - Object Management Group, 2009, http://www.omg.org.
OMG-SysML, 2009, "SysML 1.0 Specification", http://www.omgsysml.org/.
Pressman, R. S., 1995, "Engenharia de Software", Makron Books.
Schmidt, D. C., 2006, "Model-driven engineering", IEEE Computer, vol. 39, No. 2, pp.25–31.
Soares, M. dos S., Vrancken, J., 2007, "Requirements specification and modeling through sysml", Proceedings of IEEE-SMC, pp. 1735-1740.
Soares, M. dos S., Vrancken, J., 2008, "Model-driven user requirements specification using sysml", Journal of Software, vol. 3, No. 6, pp. 57-68.
SysML, 2009, "System Modeling Language", http://www.sysml.org.
UML, 2009, "Unified Modeling Language", http://www.uml.org.

8. RESPONSIBILITY NOTICE

The authors are the only responsible for the material included in this paper.

ANEXO B - A KNOWLEDGE-BASED AND MODEL-DRIVEN REQUIREMENTS ENGINEERING APPROACH TO CONCEPTUAL SATELLITE DESIGN

Este trabalho, desenvolvido durante a pesquisa de Mestrado em Computação Aplicada, foi apresentado durante o evento **28th International Conference on Conceptual Modeling** que ocorreu no período de 9-12 de novembro de 2009, em Gramado - RS, na Universidade Federal do Rio Grande do Sul - UFRGS, na sessão "ERI01 - Industrial Session".

A Knowledge-Based and Model-Driven Requirements Engineering Approach to Conceptual Satellite Design

Walter A. Dos Santos, Bruno B.F. Leonor, and Stephan Stephany

INPE - National Space Research Institute, São José dos Campos, Brazil
walter@dss.inpe.br, brunobfl@yahoo.com.br, stephan@lac.inpe.br

Abstract. Satellite systems are becoming even more complex, making technical issues a significant cost driver. The increasing complexity of these systems makes requirements engineering activities both more important and difficult. Additionally, today's competitive pressures and other market forces drive manufacturing companies to improve the efficiency with which they design and manufacture space products and systems. This imposes a heavy burden on systems-of-systems engineering skills and particularly on requirements engineering which is an important phase in a system's life cycle. When this is poorly performed, various problems may occur, such as failures, cost overruns and delays. One solution is to underpin the preliminary conceptual satellite design with computer-based information reuse and integration to deal with the interdisciplinary nature of this problem domain. This can be attained by taking a model-driven engineering approach (MDE), in which models are the main artifacts during system development. MDE is an emergent approach that tries to address system complexity by the intense use of models. This work outlines the use of SysML (Systems Modeling Language) and a novel knowledge-based software tool, named SatBudgets, to deal with these and other challenges confronted during the conceptual phase of a university satellite system, called ITASAT, currently being developed by INPE and some Brazilian universities.

1 Introduction

Space systems are complex systems designed to perform specific functions for a specified design life. Satellite projects, for instance, demand lots of resources, from human to financial, as well accounting for the impact they play on society. This requires good planning in order to minimize errors and not jeopardize the whole mission.

Therefore satellite conceptual design plays a key role in the space project lifecycle as it caters for specification, analysis, design and verification of systems without actually having a single satellite built. Conceptual design maps client needs to product use functions and is where functional architecture (and sometimes the physical architecture) is decided upon.

A.H.F. Laender et al. (Eds.): ER 2009, LNCS 5829, pp. 487–500, 2009.

Moreover, the lack of a clear vision of the satellite architecture hinders team understanding and communication, which in turn often increases the risk of integration issues. Hence, the conceptual satellite design phase demands efficient support.

Some past approaches to model-driven requirements engineering and related issues have been reported in the literature [10] [3] [14] [15] [16] [1].

This work innovates by employing SysML as a satellite architecture description language, enabling information reuse between different satellite projects as well as it facilitates knowledge integration and management over systems engineering activities. One of them is requirements engineering, more specifically requirements management and tracebility. This is an important phase in the life cycle of satellite systems. This work shows the main advantages of having user requirements being graphically modeled, their relationships explicitly mapped, and system decomposition considered in the early system development activities. In addition, requirements traceability is enhanced by using the SysML Requirements tables. The approach is illustrated by a list of user requirements for the ITASAT satellite.

Furthermore, in order to mitigate risks, this work also proposes a software tool, named SatBudgets that supports XML Metadata Interchange (XMI) information exchange between a satellite SysML model and its initial requirements budgetings via a rule-based knowledge database captured from satellite subsystems experts.

This work is organized as follows. Section 2 presents a short introduction to satellites, the ITASAT project and to SysML. Section 3 shows the SysML satellite modeling. Section 4 covers the SysML satellite requirements engineering. Section 5 introduces the SatBudgets software tool to illustrate information reuse and integration in this domain as well as describes further future work. Finally, Section 6 summarizes this research report.

2 Background

This section presents an overview of the ITASAT satellite and SysML which will be important for the paper context.

2.1 The ITASAT Satellite Project and Its Systems Rationale

A satellite has generally two main parts: (1) The bus or platform where the main supporting subsystems reside; and (2) The payload, the part that justifies the mission. A typical satellite bus has a series of supporting subsystems as depicted in Figure 1. The satellite system is built around a system bus also called the On-Board Data Handling (OBDH) bus.

The bus, or platform, is the basic frame of the satellite and the components which allow it to function in space, regardless of the satellite's mission. The control segment on the ground monitors and controls these components. The platform consists of the following components: (1) Structure of the satellite;

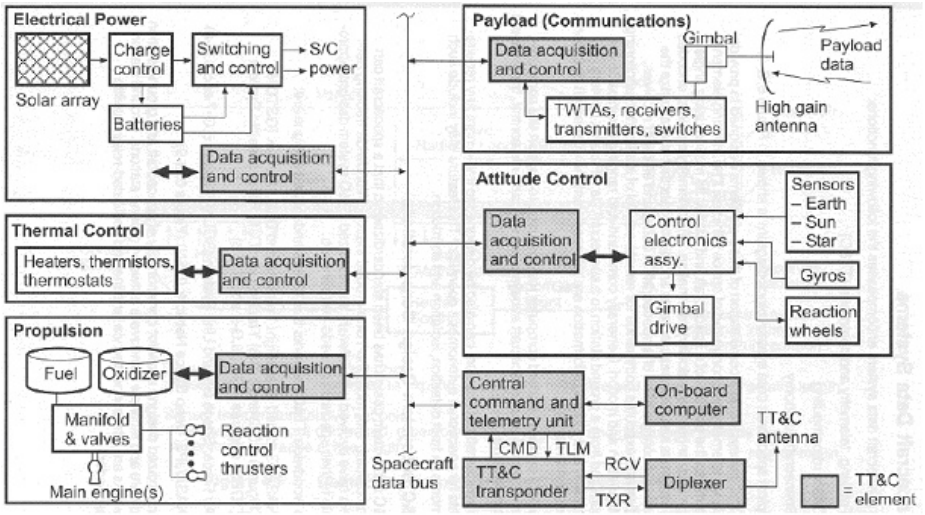


Fig. 1. Block diagram of a typical satellite [16]

(2) Power; (3) Propulsion; (4) Stabilization and Attitude Control; (5) Thermal Control; (6) Environmental Control; and (7) Telemetry, Tracking and Command.

The ITASAT satellite is part of the Small Technological Satellite Development Program funded by Brazilian Space Agency (AEB) with technical coordination of INPE and academic coordination of the Aeronautics Institute of Technology (ITA).

The ITASAT Mission entails the development, the launch and the operation of a small university satellite for use in a low Earth and low inclination orbit, capable of providing operational data collection services to the Brazilian Environmental Data Collection System (DCS), besides testing in orbit experimental payloads.

The general architecture of the ITASAT System is shown in the Figure 2 which includes: (a) The ITASAT satellite with the Data Collection System (DCS) and experimental payloads (space segment); (b) The existing Tracking, Telemetry and Command (TT&C) ground segment with Cuiabá and Alcântara tracking stations and (c) The existing Data Collection ground segment, including the Data Collection Platforms (DCP) networks.

The ITASAT satellite requires all bus functions mentioned earlier to its payloads but propulsion as no orbit maneuvers are foreseen. The Systems rationale for its detailed design follows a N-Tiered development and organization of requirements: (a) Level 0 (Mission Objective) - from which the requirements elicitation process is motivated; (b) Levels 1 and 2, are respectively focused on the definition of "science" and "high-level engineering" requirements; (c) Level 3 (Sub-system Requirements) - where engineering requirements are organized into groups (e.g., ground segment; communications segment; satellite segment)

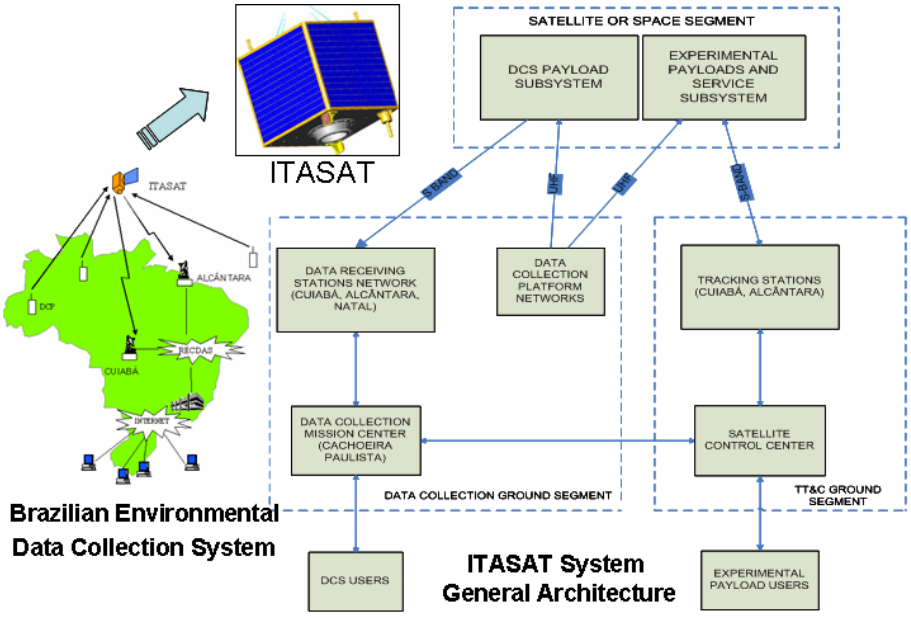


Fig. 2. ITASAT System general architecture [4]

suitable for team development; (d) Levels 4 and 5 requirements are targeted to a specific subsystem (e.g., its payloads on-board) or component (e.g., a printed circuit board) and so on.

This process generates the ITASAT Specification and Documentation Tree and also implicitly generates a highly coupled requirements tree, as depicted in Figure 3, which complicates somewhat systems engineering trade studies so far being manually performed.

For instance, on previous INPE satellite projects, the required electrical capacity for batteries is derived primarily from the power budgeting and orbital parameters of mission statement since batteries are used during eclipse times to provide power. Nevertheless, this is also coupled to others budgetings like mass, structure, etc. The lessons learned from these chained updates, due to coupling issues, justify *per se* an MDE approach to the conceptual design.

2.2 SysML as an Architecture Description Language

System modeling based on an architecture description language is a way to keep the engineering information within one information structure. Using an architecture description language is a good approach for the satellite systems engineering domain.

Architectures represent the elements implementing the functional aspect of their underlying products. The physical aspect is sometimes also represented,

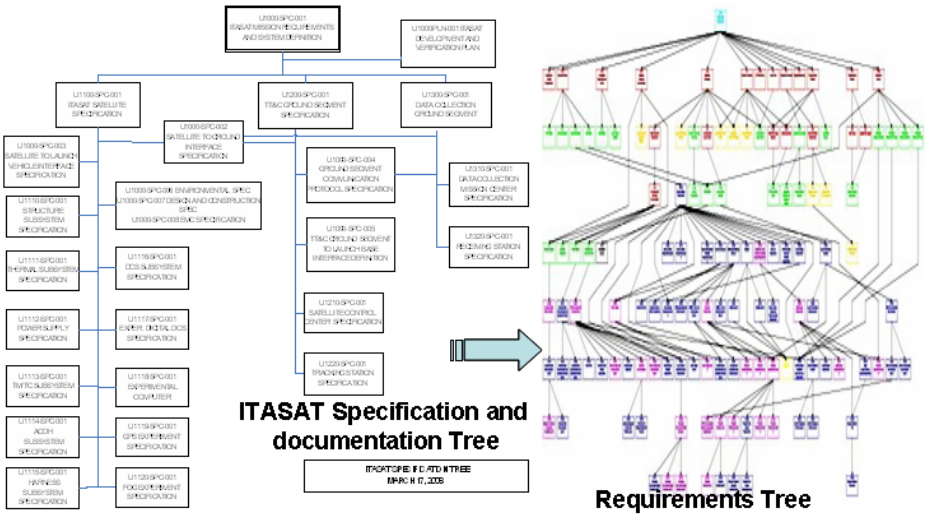


Fig. 3. Tree structure of ITASAT documents [4] and requirements coupling [1]

for instance when the architecture represents how the software is deployed on a set of computing resources, like a satellite.

SysML is a domain-specific modeling language for systems engineering and it supports the specification, analysis, design, verification and validation of various systems and systems-of-systems [17]. It was developed by the Object Management Group (OMG) [11] in cooperation with the International Council on Systems Engineering (INCOSE) [8] as a response to the request for proposal (RFP) issued by the OMG in March 2003. The language was developed as an extension to the actual standard for software engineering, the Unified Modeling Language (UML) [18] also developed within the OMG consortium.

Basically, SysML is used for representing system architectures and linking them with their behavioral components and functionalities. By using concepts like Requirements, Blocks, Flow Ports, Parametric Diagrams and Allocations, it is simple to achieve a profitable way to model systems [17].

This work explores some of the SysML capabilities through an example, the ITASAT student satellite system [4]. The application of SysML presented in this work covers only some the diagrams available in SysML due to paper scope and page restrictions.

3 Conceptual Satellite Design via SysML

Systems Engineering attacks the problem of design complexity of engineering products as it grows larger, more complex and are required to operate as part of a system.

The approach taken is formal and systematic since the great complexity requires this rigor. Another feature of systems engineering is its holistic view and

it involves a top-down synthesis, development, and operation. This suggests the decomposition of the system into subsystems and further into components [5].

3.1 Motivation for the Satellite SysML Modeling

Space Systems Engineering is a subclass of the previous mentioned in the sense that it is primarily concerned with space systems, e.g. satellite systems. Therefore it deals with the development of systems, including hardware, software, man-in-the-loop, facilities and services for space applications.

The satellite conceptual stage follows the transformation of customer needs into product functions and use cases, and precedes the design of these functions across the space engineering disciplines (for example, mechanical, electrical, software, etc.).

Model-Driven Engineering (MDE) is the systematic use of models as primary engineering artifacts throughout the engineering lifecycle [14]. MDE can be applied to software, system, and data engineering. MDE technologies, with a greater focus on architecture and corresponding automation, yield higher levels of abstraction product development. This abstraction promotes simpler models with a greater focus on the problem space. Combined with executable semantics this elevates the total level of automation possible.

3.2 The SysML Modeling Approach

SysML allows incrementally detailed description of conceptual satellite design and product architecture. This helps systems engineers which are concerned with the overall performance of a system for multiple objectives (e.g. mass, cost, and power). The systems engineering process methodically balances the needs and capabilities of the various subsystems in order to improve the systems performance, deliver on schedule and on expected cost.

SysML elements in the design represent abstractions of artifacts in the various engineering disciplines involved in the development of the system. The design represents how these artifacts collaborate to provide the product functionalities. The size, volume, and mass constraints often encountered in satellite development programs, combined with increasing demands from customers to get more capability into a given size, make systems engineering methods particularly important for this domain.

This paper explores some of the diagrams available in SysML through the example of the ITASAT satellite system by basically, exploring the block diagram and top-level requirement diagram, both shown in short detail.

SysML diagrams allow information reuse since they can be employed in other similar satellite projects by adapting and dealing with project variabilities. An exploration of these features for the on-board software design of satellites is shown in [6].

SysML allows the utilization of use case diagrams which were inherited from the UML without changes [3]. The use case diagram has been widely applied to specify system requirements. The interaction between ITASAT actors and some

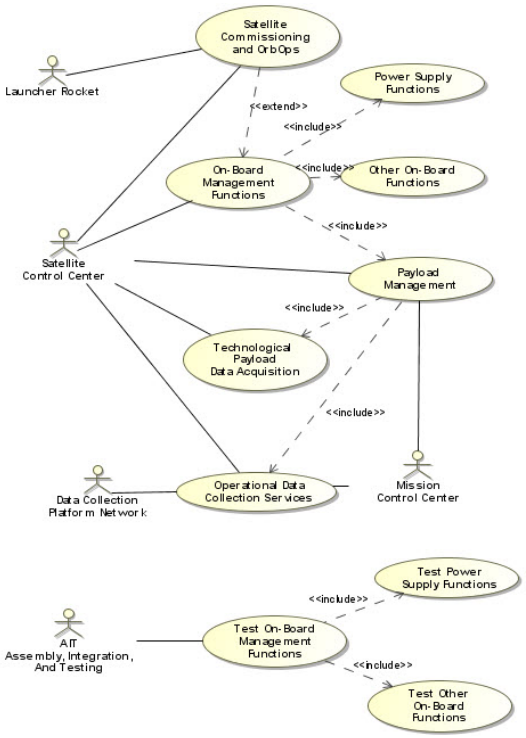


Fig. 4. ITASAT high-level use cases to specify system requirements

key use cases is shown in Figure 4. This diagram depicts five actors and how they relate to the use cases that they trigger in the high-level system view. The figure still describes schematically the composition of a series of low-level use cases hierarchically modeled by employing an ≪include≫ dependency relationship between them. SysML also allows the representation of test use cases which will be further explored in the validation, verification and testing project phases. Figure 4 depicts, as an example, the *Test On-Board Management Functions* use case and how its ≪include≫ dependencies are related to other two test use cases, *Test Other On-Board Functions* and, *Test Power Supply Functions*.

The SysML block diagram is used to show features and high-level relationships. It is used to allow systems engineer to separate basically the responsibilities of the hardware team from the software team. Figure 5 shows the various ITASAT blocks and their interdependencies.

The requirements diagram plays a key role into the SysML model as requirements present in this diagram can also appear in other SysML diagrams linking the problem and solution spaces. Furthermore, the requirements diagram notation provides a means to show the relationships among requirements including constraints. This topic is of high importance to this work hence it is further developed in the next section.

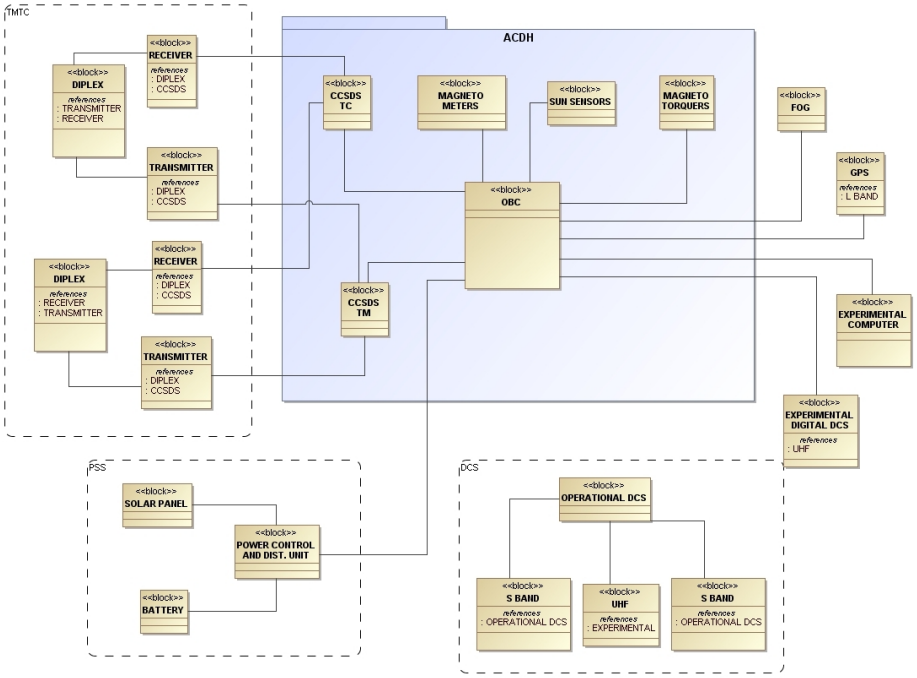


Fig. 5. The ITASAT satellite SysML block diagram

4 The Model-Driven Requirements Engineering Approach

The process of requirements engineering involves various key activities, such as elicitation, specification, prioritization and management of requirements. By using SysML, this section applies this to the satellite conceptual design.

The SysML standard identifies relationships that enable the modeler to relate requirements to other requirements as well as to other model elements [17]. Figure 6 shows a simplified view of the ITASAT requirement tree structure [4]. It also shows how a constraint is attached to a low-level requirement and how traceability may be established.

After top-level requirements are elicited, then starts the decomposition of every system requirement into progressively lower levels of design. This is done by defining the lower-level functions which determine how each function must be performed. Allocation assigns the functions and its associated performance requirements to a lower level design element. Decomposition and allocation starts at the system level, where requirements derive directly from the mission needs, and then proceeds through each segment, subsystem, and component design levels [9]. This process must also warrant closure at the next higher level meaning that satisfying lower-level requirements warrants performance at the next level. Additionally, it roundtrips all requirements tracing back to satisfying mission needs.

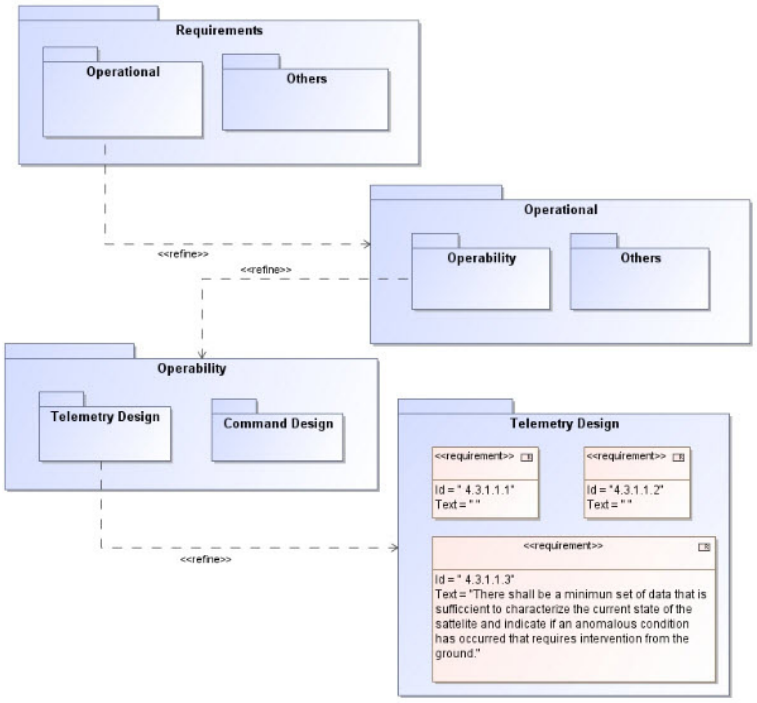


Fig. 6. Requirements tree structure for the ITASAT satellite

Managing requirements is the capability of tracing all system components to output artifacts that have been resulted from their requirement specifications (forward tracing) as well as the capability of identifying which requirement has generated a specific artifact or product (backward tracing) [13].

The great difficulty on tracing requirements is responding the following questions: *What to track?* and *How to track?*. One can say that a requirement is traceable when it is possible to identify *who has originated it, why it exists, which are the requirements related to it? how is it related to other project information*. These information is used to identify all requirement\elements affected by project changes. The specification of requirements can facilitate the communication between the various project stakeholder groups.

There are several published works on requirement engineering and the most common way they employ to requirement tracking is by posing basic questions about the underlying domain [2]. Unfortunately, such questionnaire does not offer generally any classification on the sufficient elements in order to identify all model elements.

By using a SysML requirements diagram, system requirements can be grouped, which contributes to enhance project organization showing explicitly the various relationship types between them [15]. These include relationships for defining requirements hierarchy or containment, deriving requirements, satisfying requirements, verifying requirements and refining requirements [12].

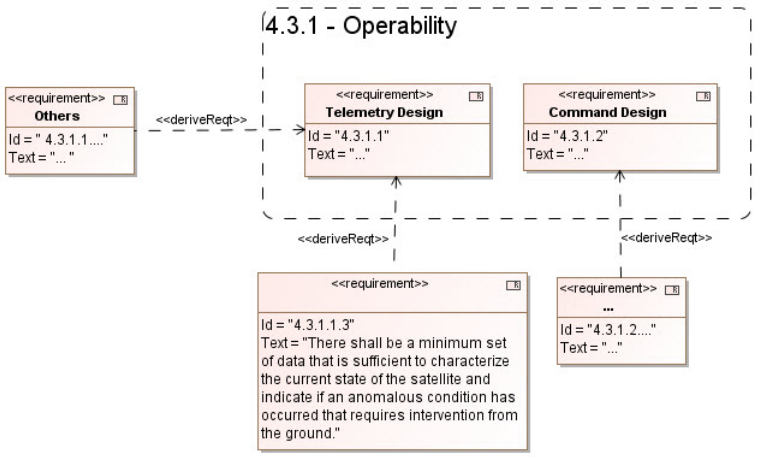


Fig. 7. An excerpt of the ITASAT requirements diagram with a ≪*deriveReqt*≫ relationship

Moreover, the SysML requirements diagram can be employed to standardize how requirements are documented following all their possible relationships. This can provide systems specification as well as be used for requirements modeling.

New requirements can be created during the requirement analysis phase and can be related to the existing requirements or complement the model. Figure 7 presents an excerpt from the ITASAT requirements diagram which utilizes the ≪deriveReqt≫ relationship type showing the derived *Satellite State* requirement from the source *Telemetry Design* requirement inside the Operability requirement SysML package. This allows, for example, a link between high-level (user oriented) and low-level (system oriented) requirements which contributes to explicitly relates the dependency of user requirements mapped into systems requirements.

Similarly, Figure 8 presents another excerpt from the ITASAT power subsystem requirements diagram which utilizes three relationships. Requirements are abstract classes with no operations neither attributes. Subrequirements are related to their "father" requirement by utilizing the ≪containment≫ relationship type. This is shown in Figure 8 as many subrequirements from the *Power Supply Requirements* requirement are connected employing ≪containment≫ relationships. The "father" requirement can be considered a package of embedded requirements.

Additionally, Figure 8 presents the ≪satisfy≫ relationship type which shows how a model satisfies one or more requirements. It represents a dependency relationship between a requirement and a model element, in this case the *Power Supply Functions* use case is satisfied by the *Power Supply Requirements*. Finally, it is shown the ≪verify≫ relationship type where the *Test Power Supply Functions* test use case is verified by the functionalities provided by the *Power Supply Requirements*. This may include standard verification methods for inspection, analysis, demonstration or test.

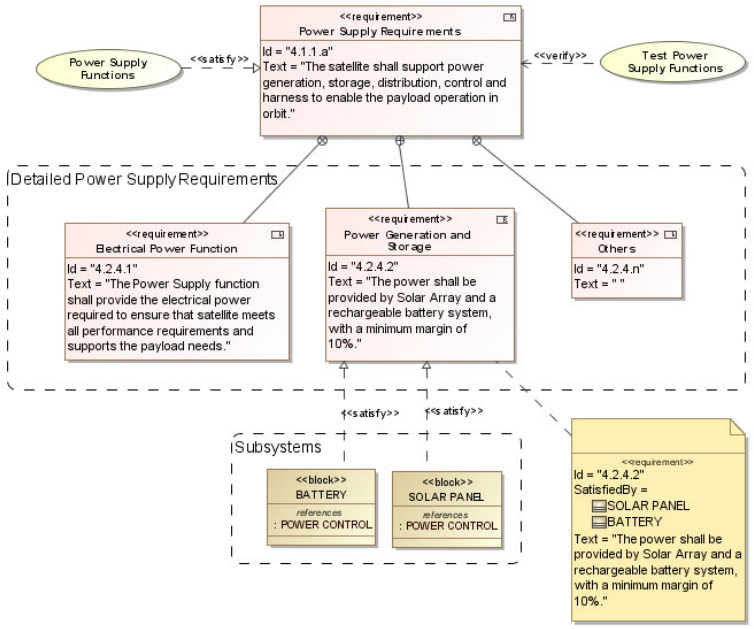


Fig. 8. An excerpt of the ITASAT power subsystem requirements diagram with containment, ≪*satisfy*≫ and, ≪*verify*≫ relationships

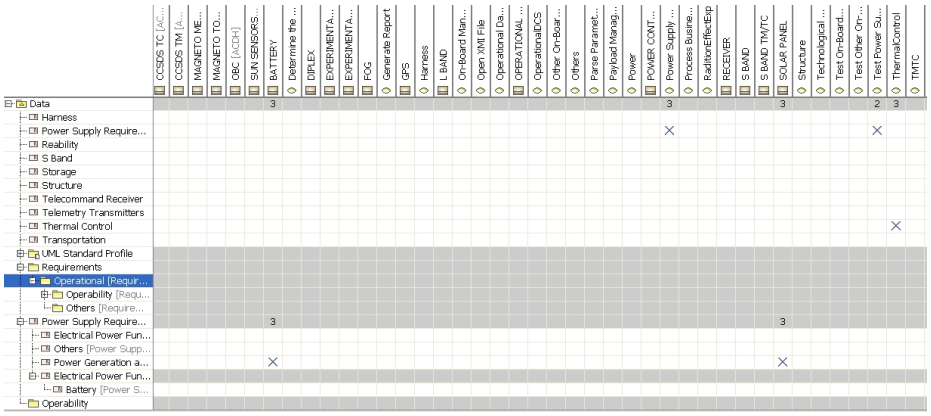


Fig. 9. The tabular matrix notation used to display power-related requirements and their relationships to other model elements

Lastly, SysML allows requirements traceability by using tabular notations. This allows model elements to be traced in SysML via requirements tables which may contain fields like: identifier (ID), name, which requirement is related to it, what type of relationship is held among them.

One such SysML tabular notation for requirements traceability is shown in Figure 9 which is suitable for cross-relating model elements. The figure shows

a requirement matrix table where cross-tracing is done between requirements, blocks defined in the ITASAT block diagram and high-level use cases. This table is quite important as it allows requirements traceability.

Additionally, requirements can be traced also by navigating through SysML requirement diagrams on the anchors points shown in Figure 8 by means of stand-out notes. The anchors contain information like the relationship type and with which model element the requirement is related and vice-versa, given a model element it may reference all requirements related to this element. Doing so, it allows a quick and simple way to identify, prioritize and improve requirements traceability. Nevertheless, the resources provided by the SysML are by far beyond the capabilities here presented due to paper page constraint.

5 The SatBudgets Software Tool and Future Work

After requirements analysis, starts the performance budgeting phase. As a case study, this work describes how a software tool, named SatBudgets, supports XMI information exchange between a satellite SysML model and its initial requirements budgetings. The software engineering activities for the SatBudgets tool are described hereafter and employs some MDE concepts enabling information reuse and integration.

The workflow of information from the satellite SysML model to the SatBudgets tool is depicted in Figure 10 and its final report spreadsheet which is employed by systems engineers for iterative designs. The sequence of events is: (a) An XMI file exported from the SysML modeling is read; (b) Parsing of key modeling parameters is performed; (c) Satellite Systems Engineering business rules are applied to infer performance budgetings; and (d) A final report is generated for systems engineers via a free Java report generator framework.

The SatBudgets tool links a SysML satellite model to activities for performance budgetings. The tool currently runs as a stand-alone Java application but it will be aggregated as a Eclipe IDE plugin [7] which already supports SysML as a plugin.

Currently a benchmark for the SatBudgets tool results are being performed. An upgrade to the tool will incorporate some additional functionalities, namely: (1) Model Roundtripping - changes to the spreadsheet will affect SysML model and vice-versa; (2) Web Service support for some specialized rule-processings; (3) Provide Database and web client support; (4) Enhance the database repertoire for Satellite Systems Engineering business rules; (5) Provide an interface to SatBudgets for Eclipse IDE aggregation and (6) Provide an interface to SatBudgets for docking to an in-house Satellite Simulator.

A more complete ITASAT SysML modeling is also expected which may include: (1) Enhancing Block Diagram representation to model detailed subsystems and components, and ports describing their interfaces; (2) Checking dependencies (e.g. analytical) between structural properties expressed using constraints and represented using the parametric diagram; (3) Exploring features behavior modeling, namely interactions, state machine and activities and;

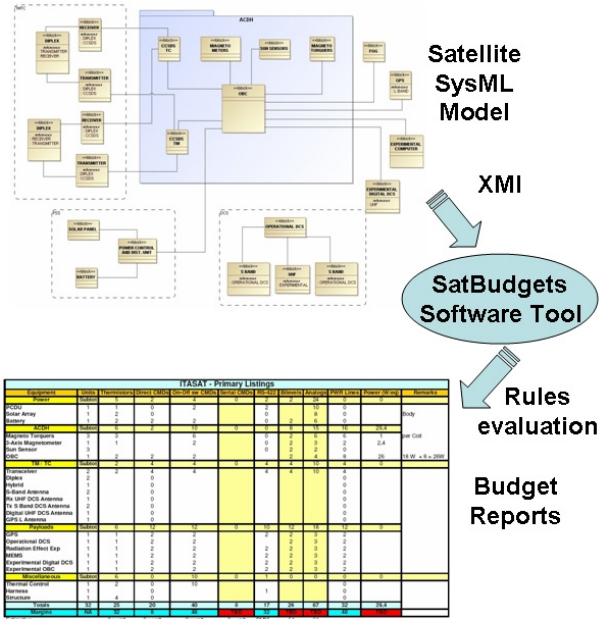


Fig. 10. Workflow for performance budgetings using the SatBudgets Tool

(4) Employing SysML for providing a mechanism to relate different aspects of the model and to enforce traceability across it.

6 Conclusions

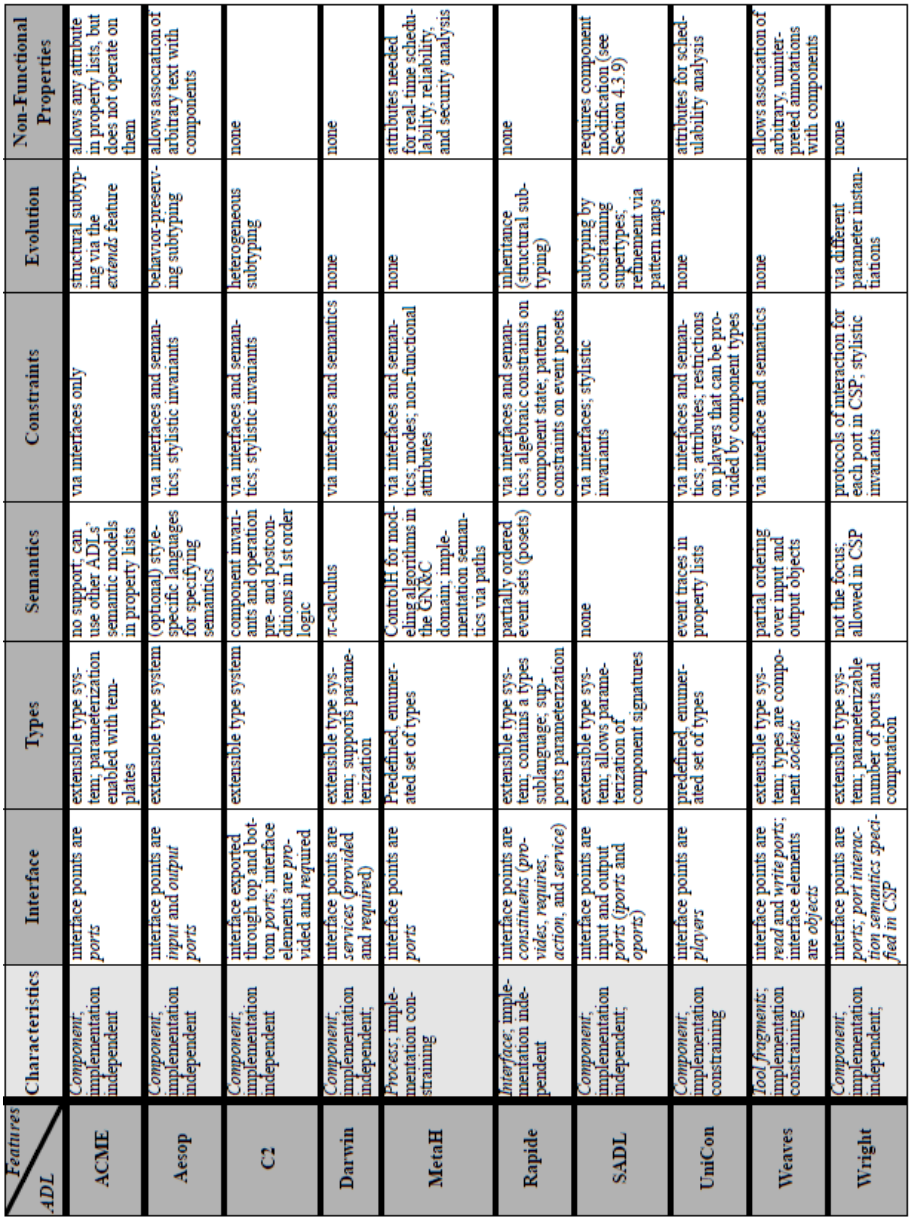
Space systems requires strong systems engineering to deal with systems-of-systems complex issues, manufacturing demands and mitigate risks. A case study was presented in this work introducing the use of SysML satellite modeling for requirements engineering and a novel knowledge-based software tool, named SatBudgets to support preliminary conceptual satellite design which demands interdisciplinary skills. By employing SysML as a satellite architecture description language, it enables information reuse between different satellite projects as well as it facilitates knowledge integration and management on systems engineering activities. This work will be further extended to implement MDE automation concepts into the ordinary workflow of satellite systems engineering.

References

1. Austin, M.A., et al.: PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. The Journal of the International Council on Systems Engineering 9(2), 129–145 (2006)
2. Aurum, A.W.: Engineering and Managing Software Requirements. Springer, Heidelberg (2005)

3. Balmelli, L.: An Overview of the Systems Modeling Language for Products and Systems Development. Journal of Object Technology (2007)
4. Carvalho, T.R., et al.: ITASAT Satellite Specification. INPE U1100-SPC-01 Internal Report (2008)
5. Dieter, G.E.: Engineering Design - a Materials and Processing Approach. McGraw-Hill International Edition, New York (1991)
6. Dos Santos, W.A.: Adaptability, Reusability and Variability on Software Systems for Space On-Board Computing. ITA Ph.D. Thesis (2008)
7. Eclipse, I.D.E.: Eclipse Foundation, http://www.eclipse.org/
8. INCOSE: International Council on Systems Engineering, http://www.incose.org
9. Larson, W.J., Wertz, J.R.: Space Mission Analysis and Design. McGraw-Hill, New York (2004)
10. Mazon, J.N., Pardillo, J., Trujillo, J.: A Model-Driven Goal-Oriented Requirement Engineering Approach for Data Warehouses. ER 2007 workshops
11. OMG: Object Management Group, http://www.omg.org
12. SysML, O.M.G.: 1.0 Specification, http://www.omgsysml.org/
13. Pressman, R.S.: Software Engineering- a Practitioner's Approach McGraw-Hill (2007)
14. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer (2006)
15. Soares, M., dos, S., Vrancken, J.: Model-Driven User Requirements Specification using SysML. Journal of Software (2008)
16. Souza, P.N.: CITS Lecture Notes. Slides - INPE (2002)
17. SysML: System Modeling Language, http://www.sysml.org
18. UML: Unified Modeling Language, http://www.uml.org

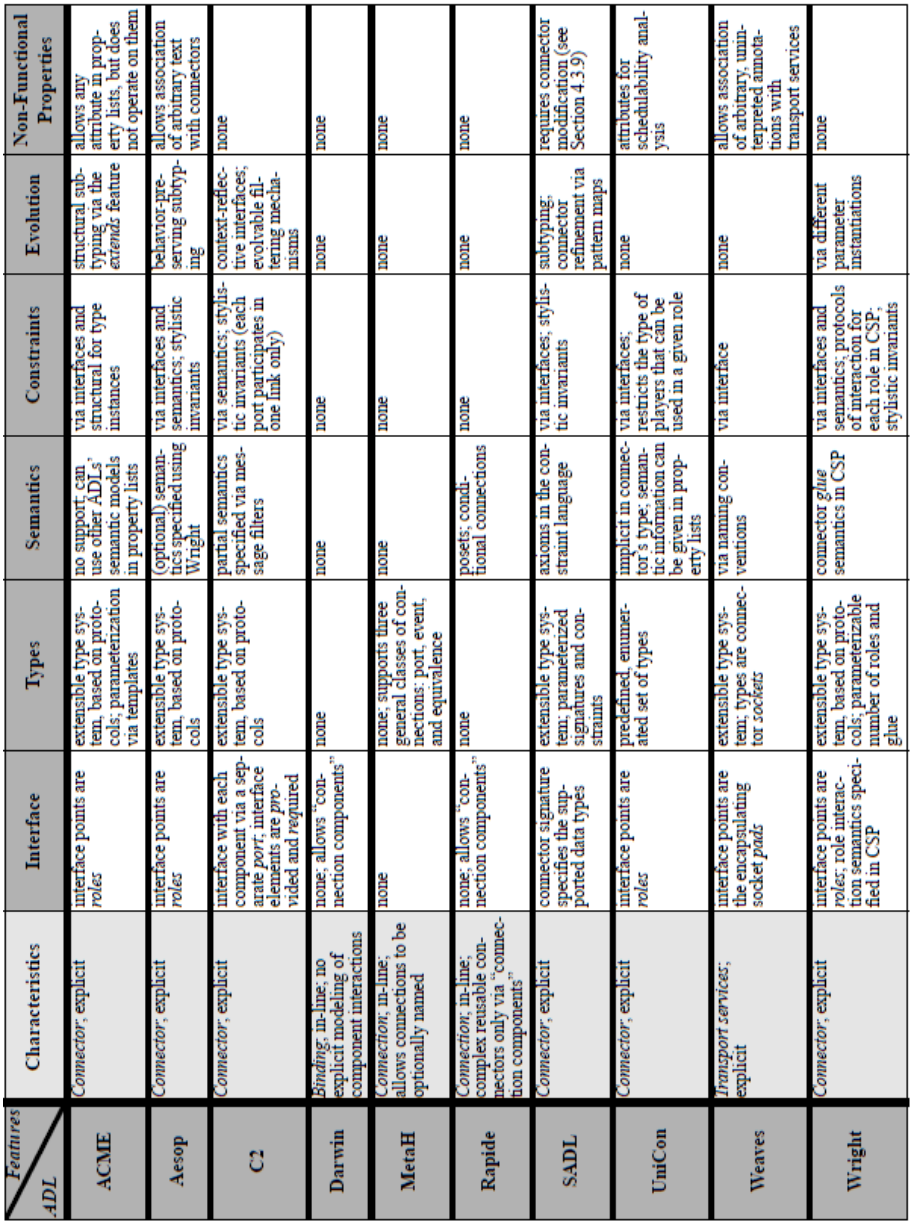
ANEXO C - COMPONENTES



Features / ADL	Characteristics	Interface	Types	Semantics	Constraints	Evolution	Non-Functional Properties
ACME	*Component*; implementation independent	interface points are *ports*	extensible type system; parameterization enabled with templates	no support; can use other ADLs' semantic models in property lists	via interfaces only	structural subtyping via the *extends* feature	allows any attribute in property lists, but does not operate on them
Aesop	*Component*; implementation independent	interface points are *input* and *output ports*	extensible type system	(optional) style-specific languages for specifying semantics	via interfaces and semantics; stylistic invariants	behavior-preserving subtyping	allows association of arbitrary text with components
C2	*Component*; implementation independent	interface exported through top and bottom *ports*; interface elements are *provided* and *required*	extensible type system	component invariants and operation pre- and postconditions in 1st order logic	via interfaces and semantics; stylistic invariants	heterogeneous subtyping	none
Darwin	*Component*; implementation independent;	interface points are *services* (*provided* and *required*)	extensible type system; supports parameterization	π-calculus	via interfaces and semantics	none	none
MetaH	*Process*; implementation constraining	interface points are *ports*	Predefined, enumerated set of types	ControlH for modeling algorithms in the GN&C domain; implementation semantics via paths	via interfaces and semantics; modes; non-functional attributes	none	attributes needed for real-time schedulability, reliability, and security analysis
Rapide	*Interface*; implementation independent	interface points are *constituents* (*provides*, *requires*, *action*, and *service*)	extensible type system; contains a types sublanguage; supports parameterization	partially ordered event sets (posets)	via interfaces and semantics; algebraic constraints on component state; pattern constraints on event posets	inheritance (structural subtyping)	none
SADL	*Component*; implementation independent;	interface points are input and output *ports* (*iports* and *oports*)	extensible type system; allows parameterization of component signatures	none	via interfaces; stylistic invariants	subtyping by constraining supertypes; refinement via pattern maps	requires component modification (see Section 4.3.9)
UniCon	*Component*; implementation constraining	interface points are *players*	predefined, enumerated set of types	event traces in property lists	via interfaces and semantics; attributes; restrictions on players that can be provided by component types	none	attributes for schedulability analysis
Weaves	*Tool fragments*; implementation constraining	interface points are *read* and *write ports*; interface elements are *objects*	extensible type system; types are component *sockets*	partial ordering over input and output objects	via interface and semantics	none	allows association of arbitrary, unninterpreted annotations with components
Wright	*Component*; implementation independent;	interface points are *ports*; *port interaction semantics specified in CSP*	extensible type system; parameterizable number of ports and computation	not the focus; allowed in CSP	protocols of interaction for each port in CSP; stylistic invariants	via different parameter instantiations	none

Figura C.1 - Suporte para Modelagem de Componentes
Fonte: Medvidovic e Taylor (2000)

ANEXO D - CONECTORES



Features / ADL	Characteristics	Interface	Types	Semantics	Constraints	Evolution	Non-Functional Properties
ACME	*Connector*; explicit	interface points are *roles*	extensible type system, based on protocols; parameterization via templates	no support; can use other ADLs' semantic models in property lists	via interfaces and structural for type instances	structural subtyping via the *extends* feature	allows any attribute in property lists, but does not operate on them
Aesop	*Connector*; explicit	interface points are *roles*	extensible type system, based on protocols	(optional) semantics specified using Wright	via interfaces and semantics; stylistic invariants	behavior-preserving subtyping	allows association of arbitrary text with connectors
C2	*Connector*; explicit	interface with each component via a separate *port*; interface elements are *provided* and *required*	extensible type system, based on protocols	partial semantics specified via message filters	via semantics; stylistic invariants (each port participates in one link only)	context-reflective interfaces; evolvable filtering mechanisms	none
Darwin	*Binding*; in-line; no explicit modeling of component interactions	none; allows "connection components"	none	none	none	none	none
MetaH	*Connection*; in-line; allows connections to be optionally named	none	none; supports three general classes of connections: port, event, and equivalence	none	none	none	none
Rapide	*Connection*; in-line; complex reusable connectors only via "connection components"	none; allows "connection components"	none	posets; conditional connections	none	none	none
SADL	*Connector*; explicit	connector signature specifies the supported data types	extensible type system; parameterized signatures and constraints	axioms in the constraint language	via interfaces; stylistic invariants	subtyping; connector refinement via pattern maps	requires connector modification (see Section 4.3.9)
UniCon	*Connector*; explicit	interface points are *roles*	predefined, enumerated set of types	implicit in connector's type; semantic information can be given in property lists	via interfaces; restricts the type of players that can be used in a given role	none	attributes for schedulability analysis
Weaves	*Transport services*; explicit	interface points are the encapsulating socket *pads*	extensible type system; types are connector *sockets*	via naming conventions	via interface	none	allows association of arbitrary, uninterpreted annotations with transport services
Wright	*Connector*; explicit	interface points are *roles*; role interaction semantics specified in CSP	extensible type system, based on protocols; parameterizable number of roles and glue	connector *glue* semantics in CSP	via interfaces and semantics; protocols of interaction for each role in CSP; stylistic invariants	via different parameter instantiations	none

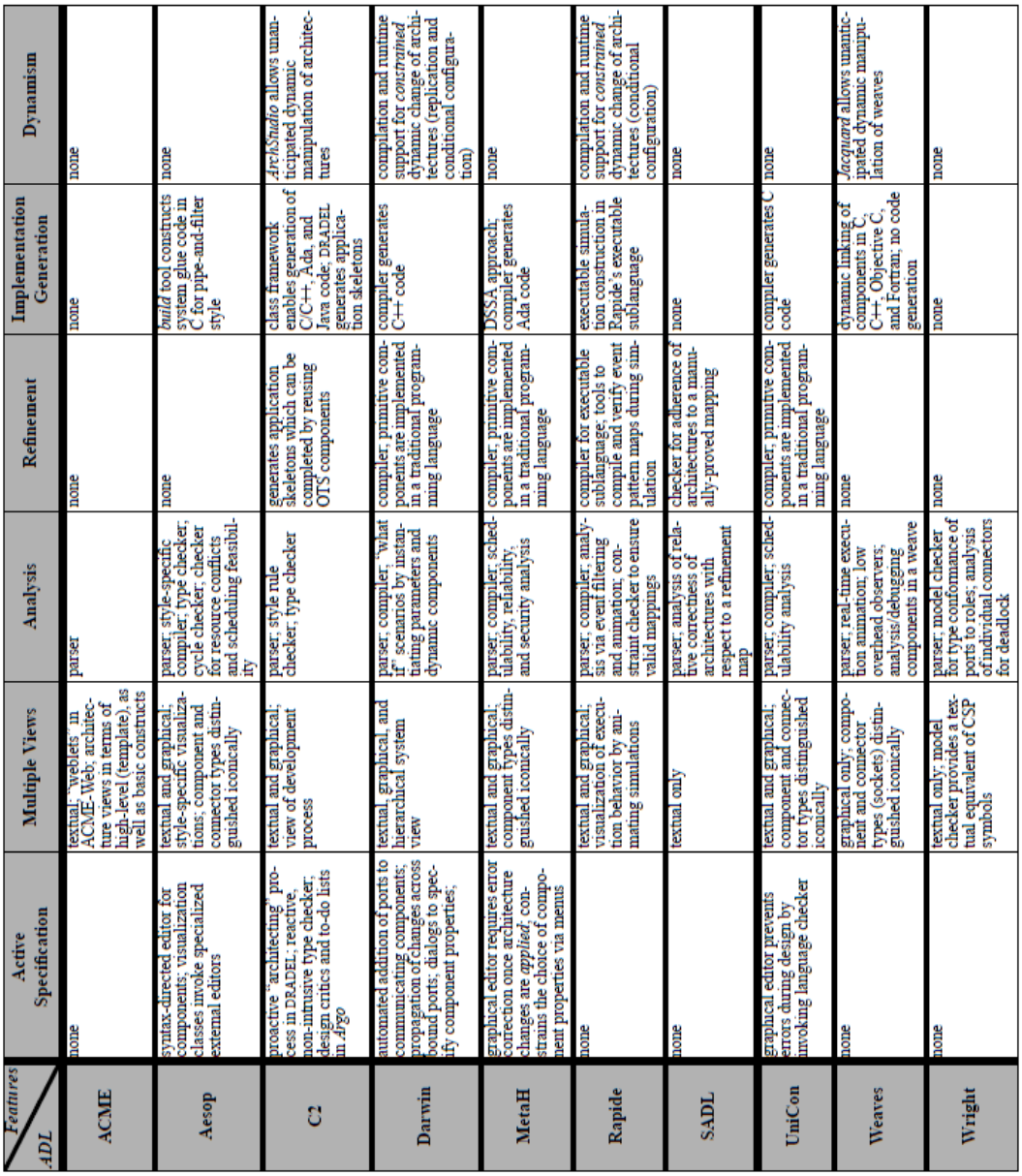
Figura D.1 - Suporte para Modelagem de Conectores
Fonte: Medvidovic e Taylor (2000)

ANEXO E - TOPOLOGIA (CONFIGURAÇÃO DE ARQUITETURA)

Features / ADL	Charact.	Understand.	Compos.	Refine./ Trace.	Heterogen.	Scalability	Evolution	Dynamism	Constraints	Non-Func. Properties
ACME	*Attachments*; explicit	explicit, concise textual specification	provided via templates, representations, and rep-maps	rep-maps	open property lists; required explicit mappings across ADLs	aided by explicit configurations; hampered by fixed number of roles	aided by explicit configurations; first-class families	none	ports may only be attached to roles and vice versa	allows any attribute in property lists, but does not operate on them
Aesop	*Configuration*; explicit	explicit, concise graphical specification; parallel type hierarchy for visualization	provided via representations	none	allows multiple languages for modeling semantics; supports development in C	aided by explicit configurations; hampered by fixed number of roles	no support for partial architectures; aided by explicit configurations	none	ports may only be attached to roles and vice versa; programmable stylistic invariants	none
C2	*Architectural Topology*; explicit	explicit, concise textual and graphical specification	allowed; supported via internal component architecture	none	enabled by internal component architecture; supports development in C++, Java, and Ada	aided by explicit configurations and variable number of connector ports; used in the construction of its own tool suite	allows partial architectures; aided by explicit configurations; minimal component interdependencies; heterogeneous connectors	unanticipated dynamism: element insertion, removal, and rewiring	fixed stylistic invariants	none
Darwin	*Binding*; in-line	in-line textual specification with many connector details; provides graphical notation	supported by language's composite component feature	supports system generation when implementation constraining	allows multiple languages for modeling semantics of primitive components; supports development in C++	hampered by in-line configurations	no support for partial architectures; hampered by in-line configurations	constrained dynamism: runtime replication of components and conditional configuration	provided services may only be bound to required services and vice versa	none
MetaH	*Connections*; in-line	in-line textual specification with many connector details; provides graphical notation	supported via macros	supports system generation; implementation constraining	supports development in Ada; requires all components to contain a process dispatch loop	hampered by in-line configurations	no support for partial architectures; hampered by in-line configurations	none	*applications* are constrained with non-functional attributes	supports attributes such as execution processor and clock period
Rapide	*Connect*; in-line	in-line textual specification with many connector details; provides graphical notation	mappings relate an architecture to an interface	refinement maps enable comparative simulations of architectures at different levels	supports development of executable simulations in Rapide's executable sublanguage	hampered by in-line configurations; used in large-scale projects	no support for partial architectures; hampered by in-line configurations;	constrained dynamism: conditional configuration and dynamic event generation	refinement maps constrain valid refinements; timed poset constraint language	timed poset model allows modeling of timing in the constraint language
SADL	*Configuration*; explicit	explicit, concise textual specification	mappings relate an architecture to a component	refinement maps enable correct refinements across styles	supports both fine- and coarse-grain elements	aided by explicit configurations; used in large-scale project	no support for partial architectures; aided by explicit configurations;	none	programmable stylistic invariants; refinement maps constrain valid refinements	architecture modified by adding constraints
UniCon	*Connect*; explicit	explicit textual and graphical specification; configuration description may be distrib-uted	supported through composite components and connectors	supports system generation; implementation constraining	supports only predefined component and connector types; supports component wrappers	aided by explicit configurations and variable number of connector roles	some support for partial architectures; aided by explicit configurations;	none	players may only be attached to roles and vice versa	none
Weaves	*Weave*; explicit	explicit, concise graphical specification	supported through composite sockets	supports system generation; implementation constraining	development in C, C++, Objective C, and Fortran; requires all tool fragments to provide a set of methods	aided by explicit configurations, sockets, and variable number of socket pads; used in large-scale project	allows partial architectures; aided by explicit configurations; support for families via socket-populated *frameworks*	unanticipated dynamism: element insertion, removal, and rewiring	precludes direct component-to-component links	allows association of arbitrary, uninterpreted annotations with weaves
Wright	*Attachments*; explicit	explicit, concise textual specification	computation and glue are expressible as architectures	none	supports both fine- and coarse-grain elements	aided by explicit configurations and variable number of roles; used in large-scale project	suited for partial specification; aided by explicit configurations	constrained dynamism: element insertion, removal, and rewiring	ports can only be attached to roles and vice versa; programmable stylistic invariants	none

Figura E.1 - Suporte para Modelagem de Topologia
Fonte: Medvidovic e Taylor (2000)

ANEXO F - FERRAMENTAS DE SUPORTE



Features / ADL	Active Specification	Multiple Views	Analysis	Refinement	Implementation Generation	Dynamism
ACME	none	textual; "weblets" in ACME-Web; architecture views in terms of high-level (template), as well as basic constructs	parser	none	none	none
Aesop	syntax-directed editor for components; visualization classes invoke specialized external editors	textual and graphical; style-specific visualizations; component and connector types distinguished iconically	parser; style-specific compiler; type checker; cycle checker; checker for resource conflicts and scheduling feasibility	none	*build* tool constructs system glue code in C for pipe-and-filter style	none
C2	proactive "architecting" process in DRADEL; reactive, non-intrusive type checker; design critics and to-do lists in *Argo*	textual and graphical; view of development process	parser; style rule checker; type checker	generates application skeletons which can be completed by reusing OTS components	class framework enables generation of C/C++, Ada, and Java code; DRADEL generates application skeletons	*ArchStudio* allows unanticipated dynamic manipulation of architectures
Darwin	automated addition of ports to communicating components; propagation of changes across bound ports; dialogs to specify component properties;	textual, graphical, and hierarchical system view	parser; compiler; "what if" scenarios by instantiating parameters and dynamic components	compiler; primitive components are implemented in a traditional programming language	compiler generates C++ code	compilation and runtime support for *constrained* dynamic change of architectures (replication and conditional configuration)
MetaH	graphical editor requires error correction once architecture changes are *applied*; constrains the choice of component properties via menus	textual and graphical; component types distinguished iconically	parser; compiler; schedulability, reliability, and security analysis	compiler; primitive components are implemented in a traditional programming language	DSSA approach; compiler generates Ada code	none
Rapide	none	textual and graphical; visualization of execution behavior by animating simulations	parser; compiler; analysis via event filtering and animation; constraint checker to ensure valid mappings	compiler for executable sublanguage; tools to compile and verify event pattern maps during simulation	executable simulation construction in Rapide's executable sublanguage	compilation and runtime support for *constrained* dynamic change of architectures (conditional configuration)
SADL	none	textual only	parser; analysis of relative correctness of architectures with respect to a refinement map	checker for adherence of architectures to a manually-proved mapping	none	none
UniCon	graphical editor prevents errors during design by invoking language checker	textual and graphical; component and connector types distinguished iconically	parser; compiler; schedulability analysis	compiler; primitive components are implemented in a traditional programming language	compiler generates C code	none
Weaves	none	graphical only; component and connector types (sockets) distinguished iconically	parser; real-time execution animation; low overhead observers; analysis/debugging components in a weave	none	dynamic linking of components in C, C++, Objective C, and Fortran; no code generation	*Jacquard* allows unanticipated dynamic manipulation of weaves
Wright	none	textual only; model checker provides a textual equivalent of CSP symbols	parser; model checker for type conformance of ports to roles; analysis of individual connectors for deadlock	none	none	none

Figura F.1 - Ferramentas de Suporte a Modelagem de Arquitetura
Fonte: Medvidovic e Taylor (2000)

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

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

Manuais Técnicos (MAN)

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

Notas Técnico-Científicas (NTC)

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

Relatórios de Pesquisa (RPQ)

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

Propostas e Relatórios de Projetos (PRP)

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

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Publicações Seriadas

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

Programas de Computador (PDC)

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

Pré-publicações (PRE)

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