UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

EDISON PIGNATON DE FREITAS

Metodologia Orientada a Aspectos para a Especificação de Sistemas Tempo-Real Embarcados Distribuídos

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, janeiro de 2007.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Freitas, Edison Pignaton de

Metodologia Orientada a Aspectos para a Especificação de Sistemas Tempo-Real Embarcados Distribuídos / Edison Pignaton de Freitas – Porto Alegre: Programa de Pós-Graduação em Computação, 2007.

171 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2007. Orientador: Carlos Eduardo Pereira.

1.Sistemas de tempo-real embarcados distribuídos. 2.Análise e especificação de requisitos não-funcionais. 3.Projeto orientado a aspectos. 4. RT-UML. I. Pereira, Carlos Eduardo. II. Título.

AGRADECIMENTOS

Ao concluir este trabalho, quero agradecer a DEUS, o supremo criador que tem me dado forças e condições de vencer os desafios e dificuldades de minha vida.

Aos meus pais, Edisson José de Freitas e Dílvia Maria Pignaton de Freitas, ao meu irmão Gustavo Adolfo Pignaton de Freitas, por todo carinho e apoio em todos os momentos de minha vida.

Ao meu orientador, Carlos Eduardo Pereira, por toda ajuda no desenvolvimento desta dissertação, bem como pela sua amizade, que muito facilitou nosso trabalho.

Ao amigo Marco Aurélio Wehrmeister, por todo trabalho realizado em conjunto, bem como por toda ajuda na realização deste trabalho.

Aos amigos e companheiros de estudo, Fabiano Costa Carvalho e Elias Teodoro Silva Jr., pelas valiosas discussões que tiveram importância decisiva neste trabalho, bem como pelos diversos trabalhos realizados em conjunto.

Aos professores Marcelo Pimenta e Flávio Rech Wagner e ao doutorando Eduardo Piveta pelos esclarecimentos de vários tópicos que muito contribuíram neste trabalho.

Aos meus chefes do 1º Centro de Telemática de Área do Exército Brasileiro pelo incentivo aos estudos através de dispensas que me permitiram assistir aulas e realizar este trabalho junto ao Instituto de Informática.

A amiga Adriana Dallacosta por sua amizade e por sua ajuda na questão metodológica da realização deste trabalho.

Aos colegas e amigos de laboratório pela convivência harmoniosa e momentos de descontração.

Ao corpo administrativo do Instituto de Informática e da biblioteca do Instituto por sua forma gentil e atenciosa de resolver todas as questões que a eles encaminhei.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AOSD	Aspect-Oriented Software Development
CASE	Computer Aided Software Engineering
DERAF	Distributed Embedded Real-time Aspects Framework
FR	Functional Requirement
IEEE	Institute of Electrical and Electronics Engineering
J2EE	Java 2 Platform Enterprise Edition
J2ME	Java 2 Platform Micro Edition
J2SE	Java 2 Platform Standard Edition
JDK	Java Development Kit
JPDD	Join Point Designation Diagram
LSE	Laboratório de Sistemas Embarcados
NFR	Non-Functional Requirement
OMG	Object Management Group
OO	Orientado à Objetos
OA	Orientado à Aspectos
QoS	Quality of Service
RT	Real-Time
RT-JAVA	Real-Time Java
RTOS	Real-Time Operation System
RTSJ	Real-Time Specification for Java
RT-UML	Real-Time Unified Modeling Language
SASHIMI	System As Software and Hardware In MIcrocontrolers
SEEP	Sistemas Eletrônicos Embarcados baseados em Plataformas
STrED	Sistemas de Tempo-real Embarcados Distribuídos
TR	Tempo-Real
TrED	Tempo-real Embarcado Distribuído
UAV	Unmanned Air Vehicle
UML	Unified Modeling Language
VANT	Veículo Aéreo Não-Tripulado

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Sistemas de tempo-real embarcados distribuídos se caracterizam pela complexidade e especificidade de seus projetos. Tanto a complexidade quanto a especificidade apresentam forte influência dos diversos requisitos ligados às restrições advindas das três características que distinguem tais sistemas, i.e. presença de fortes restrições temporais, restrições de sistemas embarcados e distribuição de processamento. Estes requisitos, chamados de requisitos não-funcionais, afetam diversas partes do sistema de maneira não uniforme, tornando-se por esta razão difícil o seu gerenciamento. Metodologias orientadas a objetos não apresentam mecanismos específicos para tratar tais requisitos, o que implica na aplicação de um significativo esforço ao se realizar o reuso ou a manutenção de componentes afetados por requisitos de natureza não-funcional. Novas tecnologias têm surgido com o objetivo de contornar este problema, notadamente a orientação a aspectos. Este paradigma propõe a separação no tratamento dos requisitos não-funcionais contribuindo com a modularização do sistema.

Esta dissertação propõe a aplicação de orientação a aspectos para a especificação de sistemas tempo-real embarcados distribuídos. Para isto realizou-se a adaptação de uma metodologia de desenvolvimento de sistemas orientada a aspectos, a FRIDA (From RequIrements to Design using Aspects), contextualizando-a para o domínio de interesse. A utilização desta metodologia provê suporte ao mapeamento de requisitos em elementos de projeto de modo a promover a rastreabilidade entre as fases de análise e projeto. Na fase de projeto é proposta a utilização de aspectos em conjunto com elementos do perfil RT-UML para o tratamento dos requisitos identificados e especificados na fase de análise.

Palavras-Chave: Sistemas de tempo-real embarcados distribuídos, análise e especificação de requisitos não-funcionais, projeto orientado a aspectos, RT-UML.

Aspect-Oriented Methodology to Specify Distributed Real-time Embedded Systems

ABSTRACT

Distributed real-time embedded systems generally have complex and very specific projects. Those characteristics are influenced by several requirements that have relation with constraints about the time, embedded and distribution restrictions. Those requirements, called non-functional requirements, can affect the whole system in a non-uniform way, what makes it difficult to handle with this kind of requirement. Object-oriented methodologies do not present specific mechanisms to handle those requirements, what imply in a significant effort to perform reuse and maintainability tasks in those components affected by non-functional requirements. New technologies are emerging to fulfill this gap, noteworthy the aspect orientation. This paradigm proposes the separation in handling functional and non-functional requirements, giving a contribution to the system modularity.

This dissertation proposes the use of aspect orientation to specify distributed real-time embedded systems. To support this proposal, it was performed an adaptation of an aspect-oriented method called FRIDA (From RequIrements to Design using Aspects). The use of this method supports the mapping of requirements in design model elements, in order to promote traceability between analysis and design phases. The presented approach proposes the use of RT-UML together with aspect oriented elements in design phase aiming to improve the handling of those requirements specified in the analysis phase.

Keywords: Distributed Real-time Embedded Systems, analysis and specification of non-functional requirements, aspect-oriented design, RT-UML.

1 INTRODUÇÃO

Sistemas de tempo-real estão sendo cada vez mais utilizados nas mais diversas áreas. A sua utilização mais expressiva se dá em sistemas críticos, tais como controle de aeronaves, automóveis e usinas nucleares, por exemplo. A especificação de sistemas deste porte é extremamente importante, uma vez que uma especificação que não expresse com exatidão suas características e funcionalidades desejadas pode ter resultados desastrosos.

Atualmente sistemas de tempo-real têm sido utilizados em ambientes distribuídos e embarcados, o que agrega a estes sistemas uma maior complexidade (SCHMIDT et al, 2002). Assim como os requisitos temporais, as características da distribuição e a relacionada ao fato do sistema ser embarcado apresentam vários requisitos não-funcionais.

Sistemas de tempo-real embarcados distribuídos (STrED) apresentam restrições ligadas a estas três características que impactam seus diversos componentes afetando de maneira não uniforme as suas funcionalidades. A preocupação em gerenciar este tipo de problema é difícil de ser concentrada em um único componente do sistema, quando se utilizam metodologias tradicionais de desenvolvimento, como as orientadas a objetos. Visando suplantar esta dificuldade, novas metodologias vêm surgindo, baseadas principalmente na idéia de desenvolvimento orientado a componentes e aspectos (TESANOVIC et al, 2004). Inicialmente, estas iniciativas tiveram seu foco voltado às fases finais do projeto, ou seja, estavam relacionadas especificamente à codificação (KICZALES, 1997). Com o aumento de sua utilização, percebeu-se que o seu emprego deveria estar presente também nas fases iniciais de análise de requisitos e design, e não apenas na fase de implementação (RASHID et al, 2002).

Uma justificativa da preocupação em se gerenciar tal problema é a dificuldade do reuso de componentes. Em sistemas de tempo-real distribuídos embarcados é a presença de tais características, chamadas de requisitos não-funcionais, o motivo que torna difícil dissociar partes do sistema visando o reuso (STANKOVIC, 2003).

Outro ponto importante relacionado aos requisitos não-funcionais diz respeito à manutenção e evolução do sistema. Como estes requisitos estão normalmente espalhados de forma caótica pelo sistema, qualquer mudança necessária gera muito trabalho, não só no sentido de identificar o impacto da mudança, mas principalmente para realizá-la de fato (BERTAGNOLLI, 2004).

Com a adoção de técnicas de engenharia de software pela comunidade desenvolvedora de sistemas de tempo-real, foi possível tornar estes sistemas cada vez mais modularizados, possibilitando desta maneira o vislumbre do reuso de certas partes dos projetos, beneficiando ainda sua manutenção. Deve-se destacar a contribuição da

utilização de metodologias orientadas a objetos para o projeto de tais sistemas (PEREIRA, 1994) (BECKER, 2003). Além disso, a utilização da linguagem UML para especificação e documentação de sistemas deste domínio possibilitou um significativo ganho de qualidade (WEHRMEISTER, 2005).

Apesar das contribuições trazidas pelo uso da orientação a objetos, o tratamento dos requisitos não-funcionais ainda representa um desafio no desenvolvimento de STrEDs. Buscando complementar a especificação de sistemas desta natureza, novas metodologias e técnicas têm sido propostas no sentido de compor, junto à orientação a objetos, ferramentas de suporte ao desenvolvimento que possibilitem contemplar também os requisitos não-funcionais. Tem-se então a proposta da utilização de orientação a aspectos, constituindo um novo paradigma (STANKOVIC, 2003).

Com a orientação a aspectos, observa-se o sistema de uma ótica diferente, não mais com uma valorização exagerada dos requisitos funcionais, mas sim atribuindo maior importância às características não-funcionais nele presentes. A proposta da sua utilização para o desenvolvimento de STrED apresenta uma forte justificativa, uma vez que seu principal foco é o atendimento de requisitos temporais, de distribuição e embarcados, requisitos estes que são notadamente características não-funcionais.

1.1 Motivação

A preocupação com metodologias de modelagem de sistemas é uma questão recorrente nas comunidades de desenvolvimento de sistemas computacionais. Isto ocorre devido ao elevado custo de desenvolvimento e manutenção dos sistemas quando não há uma preocupação com reuso e adaptabilidade. Metodologias de projeto baseadas no paradigma orientado a objetos trouxeram uma grande evolução no sentido de organizar o desenvolvimento de complexos sistemas computacionais, possibilitando o reuso e com isso a otimização da alocação de recursos.

Durante algum tempo, a comunidade desenvolvedora de sistemas de tempo-real relutou em adotar metodologias orientadas a objetos, uma vez que estas não atendiam as peculiaridades do domínio tempo-real. Posteriormente, metodologias orientadas a objetos foram adaptadas, tornando possível o seu uso em projetos de sistemas de tempo real (DOUGLASS, 1999). Com advento destas metodologias, a esta comunidade passou a contar com uma poderosa ferramenta que possibilitou um ganho significativo em seus projetos. Porém, a orientação a objetos ainda assim não atende completamente às necessidades de especificação de sistemas tempo-real, pois não consegue traduzir com total clareza alguns de seus requisitos. Isto fica ainda mais evidente quando se adiciona a característica de distribuição e/ou se torna o sistema embarcado.

Existem requisitos que repercutem no sistema como um todo e que dificilmente podem ser expressos em elementos unitários (i.e. classes, seus atributos e métodos) oferecidos pela orientação a objetos. Esses requisitos, chamados de não-funcionais, se caracterizam por se distribuírem pelo sistema, afetando vários componentes de maneira irregular, não apresentando, portanto, uma localidade que permita isolá-los com os elementos disponíveis no paradigma orientado a objetos.

Requisitos não-funcionais não são encontrados apenas em aplicações tempo real, mas também estão presentes em aplicações tolerantes a falhas, sistemas distribuídos e em aplicações com fortes requisitos de segurança. Para o correto tratamento desses requisitos, novas metodologias de projeto foram propostas, complementando a

orientação a objetos para o seu satisfatório tratamento. As principais metodologias desenvolvidas neste sentido são as baseadas em componentes (SZYPERSKI, 1998) e orientadas a aspectos (KATARA; MIKKONEN, 2001). A primeira visa especialmente à questão do reuso de subunidades em outros projetos, enquanto a segunda, além de se preocupar com o reuso, busca atender a adaptabilidade e a facilidade de manutenção, acabando com o espalhamento de código referente aos requisitos não-funcionais.

Com isso, novas metodologias orientadas a aspectos e baseadas em componentes foram propostas para o desenvolvimento de sistemas tempo-real (TESANOVIC et al, 2004). Inicialmente estas propostas abordavam principalmente a fase de implementação. Atualmente, a preocupação com os requisitos não-funcionais vem sendo focada também nas fases iniciais do ciclo de vida do sistema. A utilização do paradigma orientado a aspectos no início do projeto pode de fato auxiliar o estudo e esclarecimento dos requisitos não-funcionais, de forma a evitar mudanças indesejáveis no projeto em fases mais avançadas, mudanças que geralmente são extremamente onerosas.

Ao se extrapolar o conceito de aspectos às fases iniciais do projeto, surge um novo conceito chamado *early aspects* (RASHID et al, 2002). Sua idéia básica consiste em trazer os benefícios do uso de aspectos para as fases de análise de requisitos e design. O projeto de STrED, submetido à forte influência de requisitos não-funcionais, pode se beneficiar sobremaneira do uso desta nova tecnologia.

O vislumbre desta contribuição motiva a pesquisa da adaptação de uma metodologia de desenvolvimento orientada a aspectos voltada para STrED. Além disto, a avaliação do impacto do uso dos conceitos de orientação a aspectos para projetos neste domínio de aplicação também é necessária. Com isto, um vasto espaço de pesquisa se abre com diversas lacunas a serem preenchidas.

1.2 Objetivos do Trabalho

O objetivo principal desta dissertação é apresentar uma metodologia orientada a aspectos contextualizada para o domínio dos STrED. O foco desta metodologia é a separação da analise de requisitos funcionais e não-funcionais de modo a se obter um projeto que reflita as reais necessidades do sistema. Este trabalho busca ainda avaliar a contribuição que a adoção da orientação a aspectos fornece ao projeto de sistemas no domínio dos STrED. Esta avaliação será realizada através da aplicação de métricas que procurem evidenciar o ganho que o uso de aspectos traz ao projeto de STrED.

Estando inserido no projeto de pesquisa de Sistemas Eletrônicos Embarcados baseados em Plataforma (SEEP) (LSE, 2003), desenvolvido no Laboratório de Sistemas Embarcados (LSE) do Instituto de Informática da UFRGS, o presente trabalho se propõe ainda a incluir o tratamento de requisitos não-funcionais na metodologia de projeto do SEEP, como resultado da pesquisa e do trabalho de adaptação de uma metodologia orientada a aspectos ao domínio de interesse. A metodologia proposta se insere num trabalho maior que visa fornecer uma metodologia MDD (Model-Driven Development) que utiliza aspectos na fase de projeto e tem como meta a geração de código a partir do modelo do sistema. O referido trabalho encontra-se em andamento no contexto da tese de doutorado de Marco Aurélio Wehrmeister (WEHRMEISTER, 2006).

1.3 Organização do Trabalho

A organização do conteúdo deste trabalho busca primeiramente apresentar alguns conceitos fundamentais utilizados no seu desenvolvimento e posteriormente apresenta as contribuições propostas. Deste modo, a divisão do conteúdo encontra-se organizado como se segue.

O capítulo 2 apresenta os principais conceitos que servem de pano de fundo para a proposta desta dissertação. Neste capítulo têm destaque os seguintes assuntos: (1) caracterização da classe de sistemas alvo do estudo, i.e. sistemas de tempo-real embarcados e distribuídos; (2) o desenvolvimento orientado a objetos e sua aplicação ao domínio de interesse do estudo; (3) apresentação dos principais conceitos referentes aos requisitos não-funcionais de sistemas computacionais; (4) o desenvolvimento orientado a aspectos e seu uso no domínio de interesse; e (5) um resumo sobre métricas utilizadas para aferição de qualidade de sistemas computacionais desenvolvidos sob orientação a objetos e a aspectos.

No capítulo 3 são apresentados alguns trabalhos que propõe a utilização de orientação a aspectos nas fases de análise e projeto de sistemas computacionais, notadamente de STrED. Um terceiro item deste capítulo apresenta uma metodologia de desenvolvimento orientado a aspectos na qual esta dissertação está baseada.

No capítulo 4 encontra-se a adaptação de uma metodologia de desenvolvimento orientada a aspectos ao contexto de STrED. Ao longo do capítulo são expostas as contribuições do trabalho no sentido de adequar a metodologia escolhida ao desenvolvimento de STrED.

O capítulo 5 apresenta a aplicação da metodologia adaptada a dois estudos de caso. O primeiro se refere ao projeto de uma cadeira de rodas automatizada destinada a auxiliar pessoas portadoras de necessidades especiais. O segundo consiste no projeto de um veículo aéreo não-tripulado com diversas aplicações práticas possíveis, desde monitoramento de áreas de proteção ambiental, apoio à segurança pública, até o emprego em operações militares.

No capítulo 6 encontra-se a avaliação da utilização da metodologia. Com base nos estudos de caso apresentados no capítulo anterior, são aplicadas métricas para aferir a qualidade do projeto proposto. Uma comparação entre o projeto orientado a objetos e o orientado a aspectos avalia o ganho que a utilização de orientação a aspectos traz ao desenvolvimento de sistemas TrED.

O capítulo 7 traz as conclusões do presente trabalho e idéias que norteiam os trabalhos futuros que prosseguirão após a conclusão desta dissertação.

2 FUNDAMENTOS TEÓRICOS

Neste capitulo serão revistos alguns dos conceitos que fundamentam este trabalho. Inicialmente será apresentado um resumo com as principais características e preocupações relacionadas aos sistemas tempo-real embarcados e distribuídos. Em seguida é feita uma revisão sobre conceitos do desenvolvimento de sistemas orientado a objetos, notadamente no que se refere ao domínio de interesse do trabalho. Um resumo dos principais conceitos da orientação a aspectos é apresentado na seqüência. Deve-se ressaltar que, embora o presente trabalho não pretenda atingir a fase de implementação, é importante apresentar linguagens de programação como Java, e suas extensões tempo-real e orientada a aspectos, porque delas muitos conceitos foram abstraídos e são utilizados na fase de projeto e análise orientados a aspectos. O final do capítulo apresenta uma seção dedicada a métricas aplicáveis ao desenvolvimento de sistemas computacionais orientados a objetos e a aspectos, utilizadas para a aferição de sua qualidade.

2.1 Sistemas Tempo-real Embarcados Distribuídos

Os sistemas de tempo-real embarcados distribuídos (STrED) são aqueles que apresentam como característica marcante a precisão quanto à restrições temporais de funcionamento, têm um propósito específico (podendo ser inserido em sistemas maiores), e ainda apresentam seu processamento descentralizado em diversas unidades que trabalham de maneira cooperativa de tal forma que se atinja o objetivo computacional desejado (SCHMIDT et al, 2002).

Cada uma destas três características apresenta particularidades que determinam em conjunto a complexidade do sistema. Em seguida serão apresentados os principais pontos no que concerne cada uma destas características:

2.1.1 Sistemas de Tempo-real

Sistemas de tempo-real (TR) são aqueles que possuem fortes requisitos no atendimento de compromissos temporais, tais como tempo máximo de execução, jitter e deadline, além de robustez e segurança. A própria natureza do sistema traduz a importância dos requisitos temporais, porém, deve-se destacar ainda que tais requisitos têm tamanha importância que uma falha no seu atendimento pode causar um grande dano, podendo este dano ocorrer até mesmo no caso do funcionamento do sistema não ocorrer exatamente como o previsto (DOUGLASS, 1999).

Pode-se classificar os sistemas de tempo de real da seguinte forma (SHAW, 2001):

- *Hard Real-Time*: São aqueles nos quais os requisitos temporais devem ser rigorosamente atingidos. A perda de um prazo (*deadline*) representa uma falha do sistema, a qual acarreta consideráveis prejuízos materiais e financeiros. Neste tipo de sistema, todos os requisitos temporais devem ser atendidos sem exceção. Qualquer violação no atendimento de qualquer requisito temporal caracteriza uma falha. Neste grau de rigorosidade, um dado atrasado é um dado errado.

- *Soft Real-Time*: Neste tipo de sistema os prazos podem não ser cumpridos ocasionalmente, ou pode haver um atraso tolerável, ou mesmo as duas hipóteses, sem que isto cause falhas catastróficas. Eles são regidos, em sua maioria, por uma média de exigência temporal. Esta classificação é aplicável a sistemas que não apresentem rigorosidade temporal crítica.

Normalmente os sistemas de tempo real podem ser classificados em posições intermediárias entre os dois extremos (SHAW, 2001), levando-se em conta o quão rigorosos ou tolerantes eles são em relação ao cumprimento dos requisitos temporais. Podem-se citar como exemplos de sistemas *hard real-time* sistemas de automação industrial ou de controle de aeronaves. Já no outro extremo, sistemas de videoconferência podem ser elencados no rol dos sistemas *soft real-time*.

O principal aspecto a ser considerado nos sistemas de tempo-real é a previsibilidade com que eles executam suas atividades. Isto determina que tais sistemas sejam determinísticos, ou seja, seu comportamento temporal deve ser previsível em qualquer instante de seu funcionamento. Desta forma, contrariamente ao que se possa ser induzido a pensar, um sistema de tempo-real não precisa ser necessariamente rápido, ele deve sim ser previsível (STANKOVIC, 1998).

2.1.2 Sistemas Embarcados

Sistemas computacionais embarcados são aqueles desenvolvidos com uma missão especifica, opondo-se desta maneira ao conceito de sistema computacional de propósito geral. Eles são geralmente utilizados inseridos em sistemas maiores que os envolvem. Estes sistemas maiores apresentam necessidades específicas que são atendidas por unidades de processamento dedicadas que constituem o sistema embarcado (WOLF, 2000).

Atualmente os sistemas embarcados estão amplamente difundidos em aparelhos eletrônicos utilizados no cotidiano da vida moderna. Dvd-players, mp3-players, celulares e máquinas de lavar roupa, são algumas das inúmeras aplicações de sistemas embarcados. Estima-se que mais de 95% dos processadores produzidos em 2002 foram dedicados a aplicações embarcadas, e não processadores de propósito geral, como os utilizados em estações de trabalho e servidores, segundo números apresentados em (GANSSLE; BARR, 2003).

Os sistemas embarcados primam pela otimização da utilização de recursos. Desta forma, as principais preocupações no desenvolvimento de tais sistemas encontram-se em utilizar apenas os recursos necessários para o desempenho da atividade para a qual o sistema está sendo desenvolvido. Com isto, são produzidos sistemas com quantidades muito pequenas de memória, e principalmente reduzido consumo de potência (CARRO; WAGNER, 2003).

2.1.3 Sistemas Distribuídos

Sistemas distribuídos caracterizam-se pela distribuição espacial de unidades de processamento que em conjunto formam o sistema. Um sistema distribuído tem como objetivo dividir o trabalho de processamento proporcionando o resultado desejado sem que o usuário final do sistema perceba a descentralização. Para ele, o sistema se apresenta como uma única entidade (TANENBAUM, 2002).

Além de propiciar a união de unidades de processamento para proporcionar um maior poder computacional final, os sistemas distribuídos também são empregados em aplicações que se encontram descentralizadas especialmente por necessidades específicas de projeto, como ocorre na amostragem de dados e ação de atuadores num sistema eletrônico de direção de automóveis (*steer-by-wire*).

Os aspectos mais relevantes em sistemas distribuídos se referem a sua arquitetura, que pode ser cliente-servidor, *peer-to-peer* ou em camadas; controle de concorrência; compartilhamento de recursos; e controle da comunicação (SCHMIDT et al, 1998). Estas características são determinantes para a implementação do sistema, estando presentes em diversos dos seus componentes.

2.2 Desenvolvimento Orientada a Objetos

O desenvolvimento de sistemas computacionais orientado a objetos procura reunir numa só entidade o comportamento e estrutura dos dados de elementos do mundo real. Esta entidade é a classe.

Os principais conceitos envolvidos na orientação a objetos são (AMBLER, 2004):

- Abstração: trata da identificação das características essenciais de um determinado item;

- Encapsulamento: consiste no agrupamento de conceitos relacionados em um item específico, i.e. uma classe;

- Herança: representa um relacionamento que expressa uma classe como sendo "do tipo" de outra. Com isto transmiti-se a uma classe a mesma estrutura de outra sem a necessidade de duplicação. Isto tem dois impactos positivos, o primeiro seria a facilidade de reuso, enquanto o outro é a introdução do conceito de hierarquia entre classes;

- Polimorfismo: diferentes objetos podem responder a uma mesma mensagem de diferentes maneiras, isto permite que objetos interajam entre si sem conhecer seu tipo específico;

- Modularidade: os sistemas podem ser projetados de forma que as classes sejam desenvolvidas em módulos reusáveis;

- Reuso: a herança e os componentes modulares já caracterizam o reuso, mas este pode ser incrementado com a construção de bibliotecas de componentes que possam ser reutilizados.

Naturalmente a orientação a objetos não se limita à codificação de sistemas, o desenvolvimento orientado a objetos vai muito além disto, proporcionando ao projeto documentação que facilita o reuso e evolução do sistema. Para isto, uma linguagem

padrão é adotada. Esta linguagem, que é o padrão de fato do mercado, é a UML (Unified Modeling Language) formulada pela OMG (OMG, 2005).

A UML apresenta um conjunto de diagramas que possibilita o projeto do sistema desde a análise do problema até o projeto da solução. Diversos modelos fornecem visões distintas do sistema nas suas diversas fases dentro do projeto. Desta forma, para o estudo do problema ela fornece o diagrama de casos de uso, que tenta explicitar o que se deseja do sistema. Já para modelar a solução dispõe-se do diagrama de classes, diagrama de colaboração, diagrama de estados dentre outros.

A UML representa realmente uma ferramenta de grande valia ao desenvolvimento orientado a objetos. Porém, quando se trata de STrED, tanto a orientação a objetos quando a UML não se mostram suficientes para contemplar a complexidade envolvida nestes projetos. Para diminuir este problema foram apresentadas diversas propostas de extensão da UML de modo a adequá-la ao domínio tempo-real. Este esforço foi condensado pela OMG no perfil RT-UML (OMG, 2004). Um perfil UML é uma extensão do padrão da linguagem que refina certo conceito geral para dar uma representação mais detalhada a questões específicas de um dado domínio de aplicação. Isto se faz usando os mecanismos de extensão da UML (estereótipos, valores etiquetados e restrições).

Na seqüência será apresentado um resumo do perfil RT-UML.

2.2.1 O Perfil RT-UML

A RFP (*Request For Proposal*) de 1999 pedia "contribuições para um perfil UML que defina padrões de uso para a modelagem de aspectos relacionados com tempo, escalonabilidade e desempenho de sistemas tempo real" que deveriam:

- Permitir a construção de modelos que pudessem ser usados para fazer previsões quantitativas a respeito destas características;
- Facilitar comunicação de intenções de projeto entre desenvolvedores de maneira padronizada;
- Permitir interoperabilidade entre várias ferramentas de análise e projeto.

Além disso, para ser compatível com as tecnologias de tempo-real predominantes, paradigmas de projeto e técnicas de análise, o perfil deveria ser também totalmente aberto para aceitar novos desenvolvimentos nestas áreas.

Estes princípios levaram a um perfil que foi estruturado na forma da Figura 2.1. O pacote *Common Base* contém alguns *frameworks* que cobrem aspectos essenciais à grande maioria dos sistemas de tempo-real. No seu núcleo está um modelo de referência para modelagem de diferentes tipos de recursos. Dois *frameworks* fundamentais são adicionados, um para modelagem de tempo e outro para modelagem da concorrência, ambos derivados diretamente do modelo de referência. Esta base comum é usada para definir duas categorias diferentes de sub-perfis: os modelos de análise, para modelagem de requisitos como: escalonabilidade e desempenho, e os modelos de infra estrutura, para suporte à modelagem de tecnologias específicas, como *Real-Time* CORBA.

Encontra-se nestes pacotes a padronização da especificação de requisitos inerentemente ligados a características não-funcionais do sistema. Certamente a parte funcional não é negligenciada, muito pelo contrário, porém, como será apresentado a seguir, a maior contribuição desse perfil é a introdução de padrões para modelagem de restrições ao comportamento do sistema, bem como características desejadas de

desempenho, condições de concorrência e determinismo temporal, que correspondem a requisitos não-funcionais de grande relevância no projeto de sistemas tempo-real.

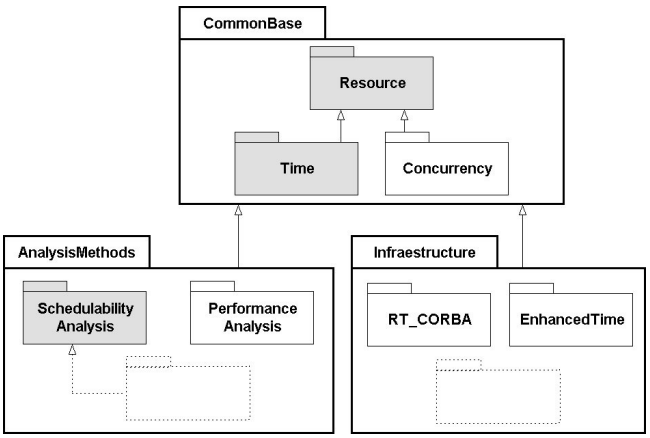

Figura 2.1: Estrutura do perfil UML-RT (OMG, 2004).

2.2.1.1 Fundamentos Conceituais

Uma característica presente na maioria dos softwares de tempo-real é que tanto o aspecto físico como o lógico são importantes. Enquanto a parte lógica da especificação determina a correção funcional do sistema, as propriedades físicas podem ter um impacto igualmente significante na sua validade. Isto acontece porque sistemas de tempo-real apresentam restrições quantitativas (caracterizadas como requisitos não-funcionais), como tempo de resposta, que dependem de atributos físicos das tecnologias usadas na implementação em baixo nível, tais como velocidade de processamento e freqüência do processador no qual as operações são executadas.

O aspecto físico do software é tradicionalmente representado através da noção de *recurso*. Note que um recurso em um sistema de software não é necessariamente um dispositivo físico. O conceito envolve, além de componentes como processadores e redes, dispositivos lógicos, como *buffers* e circuitos virtuais, embora sempre haja algo físico por trás de todo dispositivo lógico.

No modelo do perfil RT-UML, um recurso é visto como um servidor que provê um ou mais serviços para o seu cliente. A limitação física do recurso é representada através dos seus atributos de QoS (Qualidade de Serviço). Em geral, os atributos de QoS caracterizam *quão bem* o serviço de um recurso é realizado.

A Figura 2.2 mostra dois clientes ativos e potencialmente concorrentes acessando um recurso (ou serviço) monitor compartilhado. A característica de QoS é mostrada como um valor etiquetado dentro dos elementos de modelo correspondentes. Este exemplo também ilustra o valor de se incluir características de QoS no modelo. Facilmente se pode identificar, sem recorrer a nenhuma análise sofisticada, que o tempo de resposta nominal do monitor não é suficiente para atender ao *deadline* imposto pelo cliente 1. É possível que este tempo também seja insuficiente para atender ao segundo cliente, mas isto não pode ser determinado até que outros fatores, como outros objetos compartilhando o mesmo processador, sejam levados em conta.

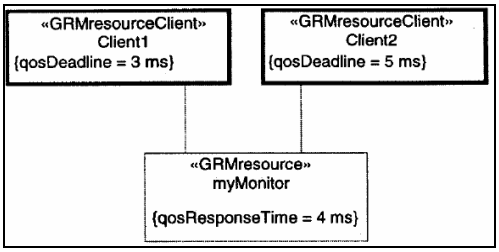

Figura 2.2: Exemplo de uma representação de serviço (ou recurso) (OMG, 2004).

2.2.1.2 Modelagem de Recursos

O *framework* para modelagem de recursos (RTresourceModeling) serve de base para que o projetista possa expressar os requisitos de qualidade de serviço (QoS) provenientes da aplicação. O modelo conceitual defendido neste perfil, constituído por recursos, serviços e requisitos de QoS, é formado pelos elementos exibidos no diagrama UML da Figura 2.3. Este modelo faz uma distinção clara entre os descritores (especificação) e as entidades em tempo de execução que implementam os requisitos. Os elementos no lado direito do diagrama representam os requisitos, sendo que cada um possui um tipo de instância correspondente. É importante ressaltar que os elementos mostrados neste diagrama não precisam, necessariamente, representar um estereótipo no perfil RT-UML, visto que neste caso a maioria deles representa entidades abstratas.

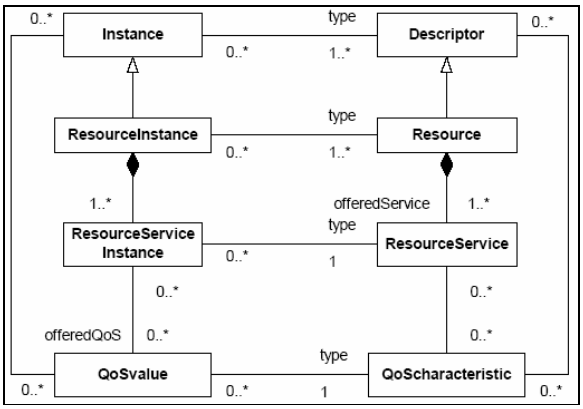

Figura 2.3: Componentes básicos do modelo geral de recursos (OMG, 2004).

2.2.1.3 Modelagem de Tempo

O padrão UML não impõe qualquer restrição na modelagem do tempo. Nem sequer assume que o tempo é contínuo ou discreto, ou que haja uma única referência de tempo no sistema. Esta flexibilidade é mantida no perfil RT-UML.

O *framework* para modelagem de tempo, onde são introduzidos conceitos para a especificação de restrições temporais, é derivado do *framework* para modelagem de recursos, como já foi dito. Os conceitos nele definidos são associados com estereótipos iniciados pelo prefixo RT. Este *framework* é particionado em quatro módulos distintos, relacionados entre si, conforme segue:

- Módulo para modelagem de tempo e valores de tempo (*TimeModel*);
- Módulo para modelagem de mecanismos de tempo (*TimingMechanisms*);
- Módulo para modelagem de eventos no tempo (*TimedEvents*);

- Módulo para modelagem de serviços temporais (*TimingServices*).

O módulo para modelagem de tempo, apresentado na Figura 2.4, discute os detalhes relacionados com a conceituação de tempo. É definido neste módulo o conceito de valores de tempo (time *value*), que se referem a uma medida de tempo. Também é definido o conceito de duração (*duration*), que representa o tempo passado entre dois instantes, sendo representada neste módulo por um intervalo de tempo (time *interval*).

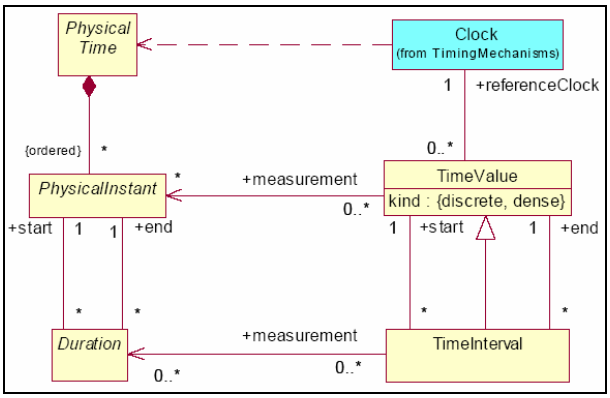

Figura 2.4: Base para modelagem de tempo (OMG, 2004).

2.2.1.4 Modelagem da Concorrência

O modelo de concorrência serve a dois propósitos primários:

- Permite ao projetista descrever um modelo suficientemente rico de objetos que executam concorrentemente e que se comunicam. Modelo este que pode servir com base para modelos de análise mais concretos.
- Permite aos provedores de infra-estrutura para sistemas de tempo real (p.ex. Sistemas Operacionais) descrever concorrência e mecanismos de comunicação em seus sistemas.

Este *framework* foi elaborado tendo em vista que concorrência é um aspecto fundamental na maioria dos sistemas tempo-real distribuídos, visto que existe uma forte interação destes com o meio físico, que é inerentemente concorrente. Os estereótipos definidos neste *framework* iniciam com o prefixo CR. Alguns exemplos são: «*CRaction*» e «*CRConcurrent*» que representam respectivamente a execução de uma ação e o conceito de unidade concorrente. O *framework* encontra-se dividido segundo o diagrama de classes apresentado na Figura 2.5, sendo que os seguintes pontos específicos são abordados em detalhes:

- Recursos concorrentes: são os recursos que representam os mecanismos para um comportamento concorrente (ou pseudo-concorrente) no sistema, sendo representados no sistema operacional por processos ou tarefas;
- Cenários concorrentes: representam seqüências de ações, geralmente interligadas, efetuadas por recursos concorrentes;
- Serviços de recursos concorrentes: representam serviços que possuem algum tipo de política de controle de acesso que os protege contra os efeitos não desejados da própria concorrência.

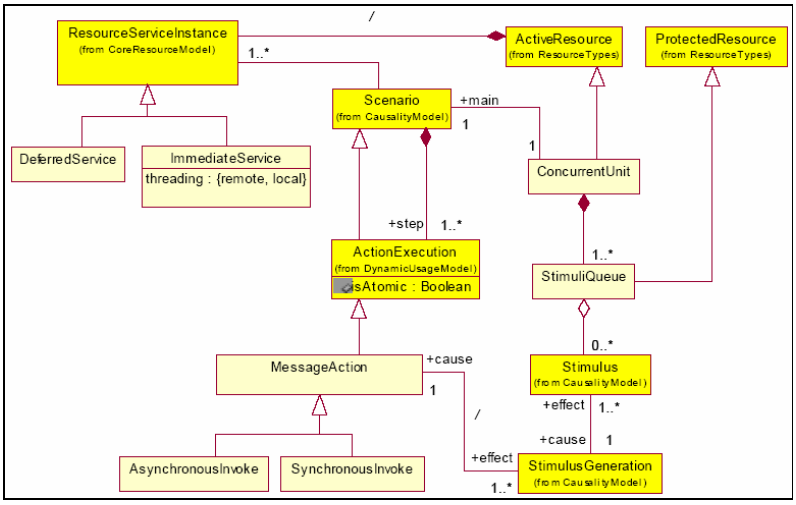

Figura 2.5: Base para modelagem da concorrência (OMG, 2004).

2.2.1.5 Modelos de Análise da Escalonabilidade

Análise de escalonabilidade se refere ao conjunto de técnicas que são usadas para determinar se uma dada combinação de software e hardware é "escalonável", ou seja, se atende os requisitos temporais impostos ou não. O perfil para análise de escalonabilidade, derivado do modelo de recursos, é mostrado na Figura 2.6. Ele define um conjunto de especializações das concepções do modelo de referência, que identifica explicitamente um conjunto de características de QoS que são usadas pelas técnicas de análise de escalonabilidade mais comuns. Os estereótipos definidos neste *framework* iniciam com o prefixo SA. Alguns exemplos são os estereótipos: «*SAengine*» que representa um recurso do tipo processador, o qual é alocado para executar recursos de escalonamento, «*SAscheduler*» que representa o escalonador do sistema.

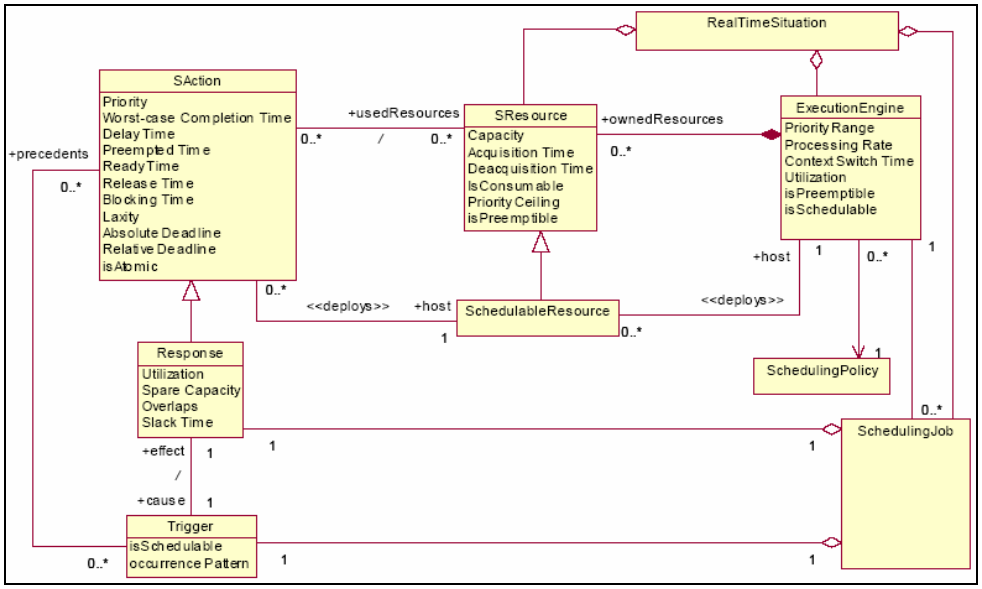

Figura 2.6: Núcleo do modelo de escalonabilidade (OMG, 2004).

2.2.1.6 Modelagem do Desempenho

O perfil para modelagem de desempenho pretende facilitar as seguintes tarefas:

- Captura dos requisitos de desempenho no contexto do projeto.
- Associação das características de QoS relacionadas a desempenho com elementos específicos do modelo UML.
- Especificação de parâmetros de execução que podem ser usados na modelagem de ferramentas para computar características de desempenho previsto.
- Apresentação de resultados de desempenho computados por ferramentas de modelagem ou obtidos por meio de testes.

Ferramentas típicas para este tipo de análise fornecem duas funções importantes. A primeira é estimar o desempenho de uma instância do sistema, usando algum tipo de modelo. A segunda função é apoio na determinação de como o sistema pode ser melhorado, identificando gargalos ou recursos críticos. Um típico projetista de sistema desejará analisar o sistema em alguns cenários, usando diferentes valores de parâmetros para cada cenário, enquanto mantém a mesma estrutura geral para o sistema.

O *framework* propõe um conjunto mínimo de conceitos para dar suporte às idéias centrais de uma análise de desempenho, tais como capacidade de recursos e taxa de carga admitida.

2.2.2 JAVA

Inicialmente criada para o desenvolvimento de aplicações embarcadas, Java (DIBBLE, 2002) teve grande penetração no mercado de aplicações voltadas para web, devido principalmente à simplicidade com a qual permite o desenvolvimento de aplicações distribuídas, portáveis e flexíveis.

Java constitui-se de uma tecnologia que além de conter a linguagem com o mesmo nome, também reúne todo o suporte para a execução dos programas em diferentes plataformas.

Enquanto linguagem, Java é orientada a objetos, apresentado as seguintes características: fortemente tipada, dinâmica (permite alocação dinâmica de memória), desaloca partes da memória não mais referenciadas (*garbage colector*), interpretada e independente de plataforma. Estas duas últimas características tornam Java altamente portável. Para se entender a portabilidade atribuída à Java, deve-se ter em mente o processo de desenvolvimento e execução de uma aplicação com esta linguagem. O programa é confeccionado e através de um compilador é gerado um *bytecode* Java para ser executado em uma máquina virtual Java (JVM). Esta máquina virtual Java fará a tradução do *bytecode* Java para os comandos da arquitetura alvo sobre a qual estiver sendo executada. Com isto, um *bytecode* Java pode ser executado em qualquer máquina, independente de sua arquitetura, desde que tenha uma JVM. Isto suporta a máxima que a SUN Microsystems prega sobre Java: “write once, run everywhere”. Além da execução sobre a JVM, um *bytecode* Java pode ser executado diretamente sobre um processador Java nativo, tal como o FemtoJava (ITO, 2001) e o PicoJava (SUN, 1999).

Quanto à plataforma, a tecnologia Java apresenta quatro plataformas base:

a) Java 2, Standard Edition (J2SE): constitui o cerne do Java, pois propicia o ambiente para o desenvolvimento de aplicações Java para desktops, e serve de base para toda a tecnologia;

b) Java 2, Enterprise Edition (J2EE): define padrões para o desenvolvimento de aplicações empresariais multicamada baseadas em componentes. Ela é baseada na J2SE e provê serviços, ferramentas e APIs adicionais para o desenvolvimento deste tipo de aplicação;

c) Java 2, Micro Edition (J2ME): consiste num conjunto de tecnologias e especificações destinados ao desenvolvimento de aplicações embarcadas, tais como PDAs e telefones celulares;

d) Java Card: busca adaptar a plataforma Java para que seja possível executar programas Java em dispositivos com recursos de memória e processamento extremamente limitados.

2.2.3 Real-time Specification for Java

A *Real-time Specification for Java* (RTSJ, 1999) busca adequar Java às necessidades de sistemas tempo-real, estendendo a API Java convencional de modo a cobrir os problemas que Java apresenta no tratamento tempo-real.

Embora Java seja provida de diversos recursos que possibilitam seu emprego em sistemas embarcados e distribuídos, a tecnologia não apresenta uma garantia de previsibilidade que é imperativa nos sistemas de tempo-real. O problema da adoção de Java para o desenvolvimento de sistemas TrED reside basicamente na não adequação desta tecnologia com relação ao tratamento de requisitos temporais, uma vez que como já mencionado, ela apresenta suporte para dispositivos embarcados e computação distribuída.

Os obstáculos ao tratamento tempo-real em Java são descritos a seguir:

a) Coletor de Lixo (*Garbage Colector*): Este é o principal obstáculo à previsibilidade em Java. Ele é um processo que é executado em três situações a saber: (1) a qualquer momento que o processador está livre; (2) quando há necessidade de se liberar memória; e (3) quando o usuário indica a necessidade de sua execução. O problema que a execução do Garbage Colector torna o programa não determinístico. Não é possível determinar-se com exatidão quando ocorrerá sua execução. A JVM decide se e quando o coletor será executado. A requisição do usuário é usada mais como uma indicação do que um critério absoluto de decisão. Isto resulta na possibilidade do coletor ser executado durante uma operação crítica, comprometendo o cumprimento de requisitos temporais.
b) Carga e Inicialização Dinâmica de Classes: A carga dinâmica de classes compromete o tratamento tempo-real em Java. Nos programas Java, a JVM adia o carregamento de classes até o momento em que realmente elas são referenciadas. Isto é um esforço para se acelerar a execução destes programas. Entretanto, isto introduz um grau de indeterminismo que não é admitido no domínio tempo-real. Mesmo que fosse possível introduzir determinismo no processo de carregamento, a inicialização só é efetivada no momento da primeira utilização, como estabelecido na especificação da linguagem Java (GOSLING; JOY; STEELE, 1996).

c) Construção e Inicialização de Objetos: Similar à questão tratada anteriormente com relação às classes, a inicialização de suas instâncias, os objetos, dispara um seqüência de chamadas de construção cujo tempo de execução depende da profundidade da hierarquia de classes e da natureza dos construtores, introduzindo, desta maneira, indeterminismo temporal.

d) Ligação Dinâmica de método (Dinamic Method Binding): A ligação dinâmica é um método que usa a herança para estabelecer polimorfismo. Ele pode degradar o desempenho da aplicação se não utilizado. Neste caso, a JVM vai ser obrigada a determinar qual o método que realmente deve ser chamado implicando novamente em imprevisibilidade e indeterminismo temporal.

e) Tratamento de Exceção: Não é possível determinar-se com precisão o tempo levado desde o primeiro lançamento de uma exceção até que seu tratador seja alcançado.

f) Distribuição: Embora Java possua um suporte que facilite a computação distribuída, a garantia temporal entre os nós ainda é um desafio.

g) Velocidade de Execução da JVM: Como uma linguagem que tem seus *bytecodes* interpretados pela JVM, Java ainda é mais lenta que outras linguagens. A velocidade de execução da JVM é dita 15 vezes em média mais lenta que a de um código C (HUNT, 1998). Na tentativa de se tornar Java mais rápido, algumas soluções tem sido propostas, tais como a já mencionada possibilidade da execução de Java nativo e compiladores *Just in Time* (HUNT, 1998).

Com o suporte previsto na *Real-time Specification for Java* (RTSJ, 1999), busca-se, portanto, cobrir os problemas anteriormente destacados introduzindo modificações que podem ser agrupadas em sete áreas, a saber:

a) Escalonamento e disparo de *threads*: adição de *real-time threads* que possuem atributos de escalonamento que proporcionam uma execução previsível. Isto representa um ganho se comparado à imprevisibilidade da execução de *threads* convencionais de Java;

b) Gerenciamento de memória: adição de ferramentas e mecanismos que ajudam os programadores a escrever código não afetado pela execução do *garbage colector*;

c) Sincronização e compartilhamento de recursos: adiciona propriedades de sincronização apropriadas às real-time *threads*;

d) Tratamento de eventos assíncronos: adiciona mecanismos que relacionam eventos assíncronos com acontecimentos fora da JVM;

e) Transferência assíncrona de controle: adiciona um mecanismo que permite a troca de fluxo de controle para outra *thread*. Isto é especialmente útil para que o tratamento de uma exceção seja feito por outra thread;

f) Finalização assíncrona de *threads*: adiciona um mecanismo que possibilita o término de uma *thread* de modo controlado e segura;

g) Acesso à memória física: adiciona mecanismos que permitem o programador controlar onde os objetos serão alocados e acessar endereços específicos da memória

2.3 Requisitos Não-Funcionais

2.3.1 Definição e Importância

De maneira resumida pode-se definir requisitos não-funcionais (RNF) como restrições nas funções oferecidas pelo sistema. Elas podem ser referentes às restrições de tempo, restrições no processo de desenvolvimento, padrões e qualidades globais de um software, como manutenibilidade, usabilidade, desempenho e custos. Os requisitos não-funcionais, ao contrário dos funcionais, não expressam nenhuma função específica a ser implementada em um sistema, mas sim condições de comportamento e restrições que devem prevalecer.

Incorporar RNFs em qualquer tipo de software é uma das tarefas mais complexas. Isso se deve ao fato destes requisitos geralmente se apresentarem muito vagos. Esse tipo de requisito pode ser tratado de maneira qualitativa ou quantitativa. No primeiro caso enquadram-se as abordagens orientadas a processo, ou seja, os RNFs afetam o processo de desenvolvimento (como por exemplo a adoção de uma determinada plataforma, determinação de custo máximo, integração com sistema legado, dentre outros). Já a abordagem quantitativa é orientada a produto, concentrando-se em requisitos que afetam o software no seu ciclo de desenvolvimento, como por exemplo, o sistema deve ser seguro, confiável, deve oferecer suporte a tolerância à falhas, dentre outros.

Os erros relacionados à omissão ou avaliação parcial de um RNF são apontados como erros caros e difíceis de serem solucionados (LINDSTRÖM, 1993). Pensando neste problema, novas metodologias têm sido propostas para se incorporar a análise de requisitos não-funcionais ao desenvolvimento de sistemas com abordagem orientada a produto, como a apresentada em (BERTAGNOLLI, 2004).

2.3.2 Classificação de Requisitos Não-Funcionais

A origem da preocupação com requisitos não funcionais remonta do final da década de 70, com a publicação de trabalhos que classificavam estes requisitos simplesmente como atributos de qualidade (BOEHM, 1978) (McCALL, 1977). O conceito de requisito não-funcional somente surgiu do modo como conhecemos hoje em 1984, com sua apresentação em (YEH, 1984). Desde então, muito se progrediu na inserção da consideração dos requisitos não-funcionais durante o desenvolvimento de sistemas computacionais.

Mas para que a utilização do conceito de RNFs fosse de fato adotado pelas metodologias de desenvolvimento de software era necessária a sua sistematização, ou seja, a proposição de um entendimento comum sobre o que são e o que tratam. Para isto foram propostas diversas classificações de requisitos não-funcionais. A classificação de mais alto nível, e que busca apenas diferenciar estes requisitos dos funcionais pode ser apresentada como segue:

- Requisitos Funcionais: são aqueles que definem as funções que o sistema ou seus componentes devem desempenhar;

- Requisitos Não-Funcionais: são também chamados de requisitos de qualidade. Referem-se às limitações no produto, (tratando de características como desempenho, interface com usuário, confiabilidade, segurança, e interoperabilidade) e no processo de

desenvolvimento (referentes à questões como metodologias de desenvolvimento, reutilização de componentes, atraso no desenvolvimento, custos, dentre outros).

Diversas outras propostas de classificações foram apresentadas, diferenciando-se no nível de detalhamento e enfoque mais voltado a uma determinada área de desenvolvimento de software. Um exemplo de classificação que apresenta um nível de detalhamento bastante rico é a apresentada em (BETAGNOLLI, 2004), cujo enfoque é voltado a sistemas tolerantes à falhas, como mostra a Figura 2.7.

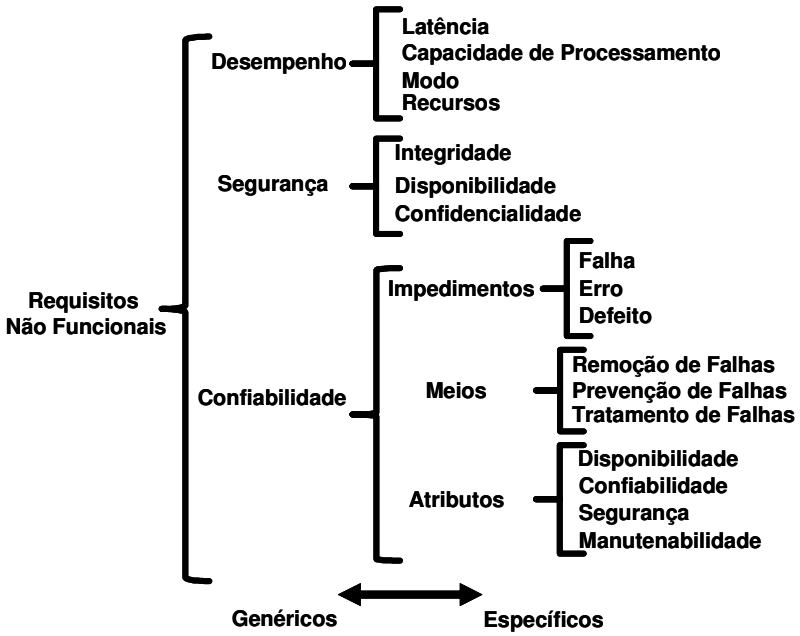

Figura 2.7: Classificação de RNFs proposta em (BETAGNOLLI, 2004).

2.3.3 Problemas Encontrados no Tratamento de RNFs

Ao se realizar a elicitação de um RNF o desenvolvedor encontra diversos problemas que dificultam a análise deste tipo de requisito do sistema. Podem-se listar alguns destes problemas baseando-se principalmente nos estudos apresentados por (KICZALES, 1997) e (CAZZOLA, 2000), como se segue:

1- Modelagem incompleta: os requisitos funcionais de um sistema são representados ou modelados como classes, objetos, atributos, operações e associações entre classes, numa metodologia orientada a objetos. Porém, os RNFs raramente são modelados. Uma vez que tais requisitos não representam entidades do mundo real, mas sim propriedades de objetos de software, estes requisitos acabam não fazendo parte dos modelos gerados no desenvolvimento de um sistema;
2- Representação dos RNFs: notações, meta-modelos, métodos e metodologias usadas para modelagem orientada a objetos convencional geralmente não são diretamente aplicáveis à modelagem de requisitos não-funcionais;
3- Transição da análise para o projeto: requisitos funcionais são relativamente fáceis de serem traduzidos para elementos do sistema na fase de projeto, sendo que o mesmo não ocorre com os requisitos não-funcionais;

4- Transparência: as abordagens tradicionais que tratam requisitos não-funcionais não costumam ser transparentes para o desenvolvedor da aplicação, o que acaba aumentando a complexidade do sistema;
5- Código altamente entrelaçado: sistemas desenvolvidos levando-se em conta requisitos não-funcionais geralmente apresentam o código que trata este tipo de requisito fortemente relacionado com aquele que trata a parte funcional do sistema. Isto acaba por dificultar a manutenção do código e reduzir a possibilidade de seu reuso e portabilidade;
6- Código disperso: o código responsável pelo atendimento de um requisito fica distribuído por mais de uma unidade funcional do sistema. Isto diminui a possibilidade de se reusar partes deste sistema, além de afetar negativamente a produtividade de quem sobre ele trabalha, uma vez que o código se torna difícil de se entender e portanto, ser modificado.

Conforme se pode observar, a maioria dos problemas está relacionada principalmente ao reuso, adaptabilidade, gerenciamento de complexidade, qualidade, rastreabilidade e desempenho.

2.3.4 Propostas para o Tratamento de RNFs

Abordagens diferentes têm sido utilizadas para solucionar ou amenizar os problemas citados na seção anterior. Elas incluem basicamente as seguintes linhas de ação:

1 – Projetar para mudança: uso de técnicas como padrões de projeto;

2 – Refatoração do código: realizar a refatoração periódica do código;

3 – Versionamento: usar múltiplas versões de componentes para representar diferentes conjuntos de características.

Estas soluções fornecem alguma flexibilidade ao código, de tal forma a prepará-lo para sofrer evolução, porém elas fracassam no que se refere ao fornecimento de um método para modularização de todos os tipos de responsabilidades (concerns).

Para suprir essa necessidade, novas soluções foram propostas, em sua maioria visando apenas algumas fases do processo de desenvolvimento. As mais adotadas atualmente são baseadas em técnicas de decomposição, ou seja, aquelas que realizam verdadeiramente uma separação de responsabilidades ao longo do desenvolvimento. As principais propostas são listadas a seguir:

1 – Protocolos de Meta-Objetos (Meta-Object Protocols) (KICZALES, 1991): Utilizada em diversas áreas da computação, o uso de meta-objetos possibilita uma separação implícita de responsabilidades e a modificação em tempo de execução. Esta técnica oferece uma arquitetura reflexiva dividida em dois níveis: o nível base e o meta-nível. A interface entre estes níveis se dá pelos protocolos de meta-objetos.

2 – Separação Multidimensional de Responsabilidade (Multimensional Separation of Concerns) (HARRISON, 1993): Tem como objetivo permitir o encapsulamento de diversos tipos de responsabilidade simultaneamente. Cada responsabilidade relevante é vista como uma dimensão. Com um conjunto definido de dimensões, realiza-se a decomposição do sistema segundo estas dimensões e realiza-se então o encapsulamento também de acordo com as dimensões definidas.

3 – Filtros de Composição (Composition Filters) (AKSIT, 1996): Busca expressar de forma mais clara a coordenação de mensagens no modelo orientado a objetos. Esta motivação foi gerada pela necessidade de se misturar código de sincronização (que é um requisito não-funcional) ao funcional para realizar esse tipo de coordenação no modelo convencional;

4 – Programação Adaptativa (Adaptative Programming) (LIEBERHERR, 1996): Objetiva fornecer uma melhor separação de responsabilidades entre o comportamento e a estrutura dos objetos nos programas orientados a objetos. Com isto, procura-se reduzir a complexidade do código, diminuindo o número de chamadas de métodos e tornando cada unidade de código mais especializada, por exemplo;

5 – Programação Orientada a Aspectos (Aspect-Oriented Programming) (KICZALES, 1997): Proposta que possibilita a estruturação do projeto e do código com a preocupação de representar e implementar os requisitos não-funcionais de forma separada da parte funcional do sistema. Ela oferece entidades chamadas aspectos, que são as sub-unidades que tratam os requisitos não-funcionais e permitem a separação de responsabilidades.

6 – Programação Orientada a Assuntos (Subject-Oriented Programming) (OSSLER; TARR, 1999): Constitui-se de uma extensão ao paradigma OO que busca manipular as diferentes perspectivas subjetivas dos objetos modelados. Fornece perspectivas diferentes dos objetos à pessoas diferentes que participam ou tem interesse de alguma forma no desenvolvimento do sistema.

2.4 Desenvolvimento Orientado a Aspectos

Assim como as metodologias orientadas a objetos, as orientadas a aspectos também tiveram seu início com propostas que objetivavam principalmente a codificação do sistema. O marco fundador desta idéia foi apresentado em (KICZALES, 1997). Neste artigo foi apresentada a linguagem AspectJ, que materializa os conceitos da orientação a aspectos através de uma extensão da linguagem Java orientada a aspectos.

Porém, o desenvolvimento orientado a aspectos não se limita à implementação do código fonte do sistema. Assim como a orientação a objetos, seus conceitos aos poucos vêm migrando para as fases iniciais do desenvolvimento. Com isto, os requisitos não-funcionais são considerados desde a fase de análise, idéia básica do conceito de *Early Aspects* (RASHID, 2002), além de serem representados também no projeto do sistema.

Os tópicos seguintes apresentam algumas considerações sobre o tratamento de requisitos não-funcionais nas fases iniciais do ciclo de vida do sistema, como o conceito de *Early Aspects*, e o uso de UML para modelar projetos orientados a aspectos. Um outro tópico será dedicado a um resumo sobre a linguagem AspectJ, e finalmente será ainda apresentado o uso de AspectJ em conjunto com a API RTSJ.

2.4.1 Early Aspects

A análise de requisitos não-funcionais nas fases iniciais do projeto de um sistema, ou seja, nas fases de engenharia (ou análise) de requisitos e design, representa um esforço para considerar o tratamento de tais requisitos de maneira mais independente do

código, ou seja, de modo a tratar tais requisitos não apenas na fase de implementação do sistema. Esse esforço de se identificar e tratar os requisitos não-funcionais no início do ciclo de vida de um sistema é conhecido como *early aspects* (RASHID et al, 2002).

A necessidade de se incluir a preocupação com a separação de o tratamento de requisitos não-funcionais em etapas da análise do problema tem basicamente duas motivações: (1) identificar e tratar conflitos entre restrições que surjam derivados de requisitos não-funcionais que afetem partes distintas do sistema; e (2) mapear a influência dos requisitos não-funcionais nos elementos de projeto e em decisões arquiteturais sobre o sistema (RASHID et al, 2002).

Esta nova perspectiva mudou a orientação da comunidade de desenvolvimento de software orientado a aspectos, que inicialmente se organizou principalmente focando a questão do software propriamente dito, com a sua estruturação e re-estruturação através do uso de aspectos. Esta utilização propiciou a escrita e re-escrita de códigos de forma a representar múltiplos conceitos e assim tornar menos complexo o emaranhado que resulta o código de um sistema de médio à grande porte. Mas isto vem mais de uma necessidade de se re-organizar o código para efeito de facilitar a manutenção, por exemplo. Porém, a possibilidade de se avaliar impactos e tratar requisitos não-funcionais desde o início do ciclo de vida do sistema abre um caminho que permite a realização de um projeto com melhor manutenibilidade e que ainda facilita o reuso (WAGELAAR, 2003).

Embora as fases de análise do problema e da solução sejam etapas distintas, em (NUSEIBEH, 2004) o autor defende a idéia de que o processo de compreensão correta do problema e a construção de uma solução estão fortemente ligados. Durante a fase de análise de requisitos e o processo de especificação, algumas decisões arquiteturais são implicitamente tomadas, o que dirige a uma identificação do que deveria ser um aspecto. Esta identificação antecipada, também conhecida como identificação de aspectos candidatos, pode levar a um design e implementação mais robustos. O que se sugere não é a identificação de aspectos durante a exploração do espaço de problema, mas sim durante o seu mapeamento para o espaço de solução, que é realizado durante a geração da especificação.

O estudo dos requisitos entrelaçados serve, portanto, para se evidenciar e descrever a sobreposição de requisitos. Esta descrição é o primeiro passo para o tratamento de possíveis inconsistências que podem surgir desta sobreposição. Uma outra utilidade é a sua utilização como entrada para o design e implementação orientados a aspectos. Desta forma, fica garantida a rastreabilidade dos requisitos não-funcionais em elementos do sistema.

2.4.2 Uso de UML para Modelar Sistemas Orientados a Aspectos

A UML é uma linguagem padrão de fato na indústria para se realizar a modelagem de sistemas computacionais. Como os conceitos de orientação a aspectos estão sendo trazidos às fases de análise e projeto, é natural que se deseje utilizar a UML para representar tais conceitos no desenvolvimento orientado a aspectos.

Dentre as diversas propostas existentes, destacam-se algumas abordagens tradicionais e não tradicionais. Esta classificação de abordagens, apresentada em (KHAN; JAFFAR-UR-REHMAN, 2005), divide os trabalhos de desenvolvimento de software orientado a aspectos de acordo com o paradigma utilizado na sua fase de

engenharia de requisitos. De acordo com o paradigma utilizado nesta fase, os diferentes trabalhos apresentam diferentes tratamentos e padrões de representação nas fases subseqüentes.

As abordagens tradicionais buscam estender a linguagem UML através de perfis, aumentando o poder semântico dos elementos já presentes na linguagem. Tais perfis especificam estereótipos a serem aplicados a elementos de modelo para representar os elementos não-funcionais nas fases de análise e projeto. Estas abordagens se baseiam em conceitos tradicionais de engenharia de requisitos, tais como casos de uso e cenários, por exemplo. Alguns exemplos deste tipo de abordagem são: AOUML (ARAÚJO, 2002), FRIDA (BERTAGNOLLI, 2004) e UCSoC (SOUZA et al, 2004). Devido ao interesse específico desta dissertação nesta linha de abordagem, alguns destes trabalhos serão mais profundamente analisados na apresentação do estado da arte. Em especial a metodologia FRIDA servirá de arcabouço para a proposta desta dissertação.

Uma outra vertente de trabalhos na área engloba abordagens não-tradicionais. Tais abordagens utilizam novos conceitos como desenvolvimento baseado em componentes, abordagem não-convencional e separação de interesses multidimensionais. Alguns exemplos deste tipo de trabalho são: ACCORD (TESANOVIC et al, 2004) e Theme/UML (CLARKE; WALKER, 2003). A linguagem Theme/UML apresenta o suporte para a representação UML de entidades chamadas "temas" na fase de projeto. Estas entidades na fase de projeto são mapeadas diretamente de visões do sistema descritas através de uma especificação chamada Theme/Doc.

É importante ressaltar que não existe um padrão definitivo para a modelagem de aspectos através da UML. Os trabalhos mencionados, sejam baseados em abordagens tradicionais ou não, apresentam propostas de como esta modelagem pode ser feita, porém nenhum deles representa consenso na comunidade desenvolvedora de sistemas orientados a aspectos.

2.4.3 AspectJ

AspectJ é a linguagem orientada a aspectos mais madura que se encontra em uso. Ela foi a linguagem que implementou o paradigma de programação orientada a aspectos proposto por Kiczales (KICZALES, 1997).

AspectJ constitui-se de uma extensão de Java orientada a aspectos. Ela adiciona apenas um novo conceito à linguagem Java, o conceito de ponto de junção (*join point*). Além desse conceito, algumas novas construções também são inseridas para suportar o novo paradigma, são elas: pontos de corte (*pointcuts*), *advice*, *introductions* e aspectos.

Um *join point* nada mais é do que um determinado ponto no fluxo do programa. Ele pode ser uma chamada de método, uma atribuição de valor, ou uma simples comparação entre variáveis. O *pointcut* é a construção responsável por capturar um *join point* (ou um conjunto de *join points*) quando este é atingido. Já o *advice* constitui o bloco de código determinado pelo *pointcut* que será executado quando se atingir o respectivo *join point*. AspectJ apresenta suporte a três tipos de *advices*: *before*, *around* e *after*, sendo que este último possui duas variantes, a *after returning* e *after throwing*. O *advice before* é executado imediatamente antes de se alcançar um *join point*. Já o *after* é executado após o *join point.* O *advice around* permite alterar o fluxo de execução, e até mesmo substituir a execução do *join point* por um outro método. Os *pointcuts* e os *advices* em conjunto interferem de maneira dinâmica no fluxo do programa.

Os *introductions* permitem o programador modificar a estrutura estática do programa, ou seja, permite a alteração da estrutura das classes, introduzindo novos atributos, métodos e construtores, por exemplo.

O aspecto é a unidade que permite modularizar a preocupação com os interesses ou conceitos entrelaçados. Ele se assemelha a uma classe Java, porém deve conter *pointcuts*, *advices* e *introductions.* No aspecto se concentram, portanto, todo tratamento não-funcional do sistema. Todo código não afeto a funcionalidade do sistema, mas sim a questões referentes a restrições, atributos de qualidades ou condições encontram-se encapsuladas nesta entidade.

Para se construir um programa usando AspectJ, deve-se construir normalmente as classes (que devem representar apenas os requisitos funcionais do sistema), e os aspectos (representando os requisitos não-funcionais do sistema). Em seguida os códigos das classes e dos aspectos devem passar por um pré-compilador, chamado de entrelaçador de aspectos (*aspect weaver*) e finalmente por um compilador Java, que irá gerar os *bytecodes* que podem ser executados em uma máquina virtual Java. A Figura 2.8 apresenta esse processo.

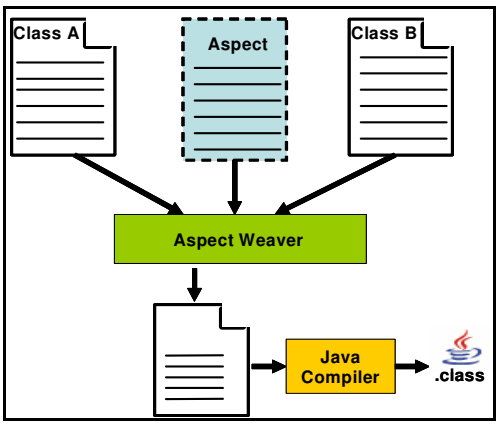

Figura 2.8: Compilação de um programa orientado a aspectos em AspectJ

2.4.4 Utilização de RTSJ em conjunto com AspectJ

A implementação de sistemas tempo-real ganhou uma forte contribuição com a especificação da RTSJ, como apresentado na seção 2.2.3. Porém, como relembrado em várias passagens deste trabalho, uma abordagem orientada a objetos não proporciona a melhor solução para o desenvolvimento de sistemas que apresentam características não-funcionais marcantes, seja nas fases de análise e projeto, seja na fase de implementação. Nesta seção analisa-se a utilização conjunta da API RTSJ e AspectJ para desenvolvimento de sistemas TrED.

Dois trabalhos se destacam nesta linha de pesquisa, um apresenta uma visão de encapsulamento de todo código RTSJ em aspectos (TSANG, 2004); e outro que apresenta uma utilização de aspectos voltada à concentração da preocupação do controle da administração da alocação e liberação de memória de modo a eliminar os efeitos da atuação do coletor de lixo (DETERS, 2001).

Em (TSANG, 2004), é apresentada a refatoração de um sistema originalmente desenvolvido orientado a objetos com o uso da RTSJ. O resultado desta refatoração foi um sistema orientado a aspectos que apresenta sete aspectos que entrecortam suas

classes. Estes aspectos se referem a cada uma das sete áreas de melhoria da RTSJ em relação à linguagem Java – (1) Escalonamento e disparo de *threads*; (2) Gerência de memória; (3) Sincronização e compartilhamento de recursos; (4) Tratamento de eventos assíncronos; (5) Transferência de controle assíncrona; (6) Término assíncrono de threads, e (7) Acesso à memória física. Realiza-se uma comparação entre os dois sistemas, o orientado a aspectos e o orientado a objetos, através de métricas baseadas na suíte C&K (CHILDAMBER; KEMERER, 1994). A avaliação aponta para os inconvenientes da utilização indiscriminada de aspectos na implementação de um sistema. Embora várias métricas tenham apresentado melhores resultados, como aquelas que definem a modularidade, outras têm seu resultado degradado, como as que definem a complexidade do sistema. Segundo conclusão apresentada no trabalho, isto se dá pelo fato de não ser necessária a utilização de aspectos em todos os casos apresentados. De fato, esta conclusão se apresenta bastante pertinente. Uma vez que o desenvolvimento do sistema não teve uma preocupação com os requisitos não-funcionais desde seu início, mas sim sua implementação orientada a objetos refatorada, não se pode garantir que o uso dos aspectos realmente atendeu de maneira coerente os requisitos do sistema.

Já o trabalho apresentado em (DETERS, 2001) estuda a utilização de aspectos para se obter as vantagens do uso de escopos de memória (definidos pela RTSJ) de forma automática e não estaticamente definidas. Esta automatização constitui uma contribuição ao reuso de partes de um sistema que se apresentam muito específicas para determinada aplicação quando os escopos de memória são estabelecidos estaticamente. O trabalho apresentado busca então trilhar como os objetos referenciam os outros num sistema de modo que se possa construir uma estrutura de escopos aninhados que não viole regras de referência de objetos em RTSJ. Para isto foi desenvolvido um aspecto chamado "reference-probing", que determina a perda de referência de um objeto. Define-se os *joinpoints* como sendo a instanciação dos objetos e a sua atribuição, com isto monta-se um grafo de referências entre objetos. Este grafo de referências consiste num grafo acíclico direto que provê mais informações para a definição dos escopos. Com a determinação dos *joinpoints* no programa original utiliza-se aspectos "reference-probing" e "liveness-probing", o primeiro determinando a perda de referência de um objeto e o outro se um objeto está vivo ou morto a partir da análise da execução dos métodos e do construtor do objeto. Combinando as informações reunidas por esses dois aspectos mais o que o grafo de referências apresenta, é possível montar-se a hierarquia de escopo que preserva a semântica da aplicação e respeita os requisitos da RTSJ. Também neste trabalho se parte de um sistema já desenvolvido com o uso de orientação a objetos. O uso de aspectos apresenta uma melhora na implementação do sistema, encapsulando o tratamento tempo-real, porém isto somente foi considerado já na fase de implementação.

Os dois trabalhos apresentados nesta seção referem-se essencialmente à fase de implementação de um sistema TrED, fase esta que não se encontra no escopo desta dissertação. Porém, o estudo de propostas como estas permite a compreensão da dificuldade do correto uso de aspectos no desenvolvimento de sistemas TrED, e reforça a idéia de que a separação das dimensões funcional e não-funcional deve ser abstraída para as fases inicias do ciclo de vida do sistema, de modo a se apresentar um claro mapeamento entre requisitos, projeto e implementação.

2.5 Métricas de Avaliação

Seja orientado a objetos ou orientado a aspectos, o desenvolvimento de sistemas computacionais devem ser submetidos a métricas que consigam medir atributos de qualidade, que por sua vez permitam que a qualidade do sistema seja inferida. A seguir serão apresentadas algumas métricas presentes na bibliografia consultada.

2.5.1 Childamber and Kemerer (C&K) Metrics Suíte

O conjunto de métricas C&K (CHILDAMBER; KEMERER, 1994) foi projetado para medir os principais fatores que afetam a qualidade de um software orientado a objetos: encapsulamento, abstração e herança. Este conjunto de métricas vem sendo usado em diversos trabalhos, inclusive para avaliação de sistemas aeroespaciais desenvolvidos pela NASA (ROSENBERG, 2003).

As métricas C&K consistem de seis medidas que são identificadas a seguir:

(1) Peso de Métodos por Classe (PMC): representa o número de métodos implementados em uma classe;
(2) Profundidade da Árvore de Herança (PAH): indica a maior distância de uma classe até a superclasse mais acima na hierarquia de herança;
(3) Número de Subclasses (NS): corresponde ao número de subclasses imediatamente abaixo de uma classe;
(4) Acoplamento Entre Objetos (AEO): representa o número de classes de onde uma classe utiliza algum elemento;
(5) Resposta de uma Classe (RC): indica o número de métodos que podem ser potencialmente executados em resposta a uma mensagem recebida por um objeto da classe;
(6) Falta de Coesão nos Métodos (FCM): mede falta de coesão entre os métodos de uma classe. Consiste no grau de relação entre métodos de uma classe quanto à utilização de variáveis compartilhadas.

Em conjunto, estas métricas são utilizadas para determinar atributos de qualidade do sistema. A Tabela 2.1 apresenta a relação entre as métricas e os atributos, onde as células marcadas indicam que a métrica da respectiva coluna influi no atributo daquela linha.

Tabela 2.1: Influência das métricas nos atributos de qualidade

	PMC	PAH	NS	AEO	RC	FCM
Compreensão	X	X		X	X	
Manutenibilidade	X			X	X	
Capacidade de Reuso	X	X	X	X		X
Facilidade de ser Testado		X	X	X	X	

Fonte: CHILDAMBER; KEMERER, 1994

Embora o objetivo geral seja minimizar o valor das métricas, deve-se observar que as métricas Profundidade da Árvore de Herança e Número de Subclasses, não seguem

esta lógica para todos os atributos de qualidade. Um alto PAH aumenta a complexidade, mas melhora o reuso. Da mesma forma que um alto NS aumentará o esforço para se realizar testes (diminuindo a facilidade de ser testado), porém aumentará a capacidade de reuso. Desta forma, não se deve olhar isoladamente as métricas nem os atributos de qualidade, mas sim ponderar qual deles é mais importante para determinado objetivo do projeto e adaptar o seu uso ao caso particular em questão.

2.5.2 Framework de Avaliação de Sistemas Orientados a Aspectos

O trabalho analisado nesta subseção apresenta um conjunto de métricas e um modelo de qualidade que utiliza conceitos amplamente aceitos, como métricas C&K, estendendo-os com objetivo de adequar a avaliação de sistemas desenvolvidos através do paradigma orientado a aspectos (SANT'ANNA et al 2003).

A seguir serão apresentadas as métricas e logo a frente como elas se relacionam para inferir a qualidade do sistema.

As métricas são agrupadas em quatro conjuntos, a saber:

(1) Métricas de Separação de Conceitos (Preocupações ou Interesses) (MSC): avalia a capacidade de se encapsular e manipular partes de um sistema que são relevantes a determinado conceito ou preocupação ou interesse. As seguintes métricas se enquadram neste grupo:
   (a) Difusão de Conceitos (preocupações ou interesses) por Componentes (DCC): conta o número de componentes que tem por objetivo principal contribuir na implementação de determinado conceito;
   (b) Difusão de Conceitos (preocupações ou interesses) por Operações (DCO): conta o número de operações que tem por objetivo principal contribuir com a implementação de determinado conceito;
   (c) Difusão de Conceitos (preocupações ou interesses) por Linhas de Código (DCLC): conta o número de pontos de transição para cada conceito através das linhas de código;

(2) Métricas de Acoplamento (MA): indicam a força das interconexões entre componentes no sistema. As seguintes métricas pertencem ao grupo:
   (a) Acoplamento Entre Componentes (AEC): definido para um componente, classe ou aspecto, como o número de componentes aos quais está acoplado. Uma classe está acoplada a outra quando seus métodos acessam atributos de outras classes. Já um aspecto é considerado acoplado quando afeta uma classe;
   (b) Profundidade da Árvore de Herança (PAH): corresponde ao número de ancestrais contados a partir do componente (classe ou aspecto) mais abaixo na hierarquia;

(3) Métrica de Coesão (MC): a coesão mede a relação de dependência entre elementos internos de um componente. Apenas uma métrica faz parte deste grupo:
   (a) Falta de Coesão nas Operações (FCO): mede a falta de coesão dos componentes do sistema. O cálculo desta métrica é feito da seguinte maneira: seja *n* o número operações (métodos ou *advices*) O_1,..,O_n de um componente C1 do sistema e I_j o conjunto de variáveis de instância

utilizadas pela operação O_j. Seja P o número de intercessões vazias entre os conjuntos I e Q o número de intercessões não-vazias entre estes conjuntos. A métrica FCO é definida como P – Q se P > Q ou 0 caso contrário.

(4) Métricas de Tamanho (MT): são aquelas que medem o tamanho do projeto e/ou código do sistema. As métricas componentes deste grupo são as seguintes:
   (a) Tamanho do Vocabulário (TV): conta o número de componentes do sistema – número de classes e aspectos que compõe o projeto;
   (b) Linhas de Código (LC): conta o número de linhas de código do sistema;
   (c) Número de Atributos (NA): é uma métrica que fornece o vocabulário interno de um componente, contando o número de atributos de uma classe ou aspecto;
   (d) Peso de Operações por Componente (POC): esta métrica mede a complexidade de um componente em termos de suas operações. Ela estende a métrica PMC definida no conjunto de métricas C&K.

Apresentadas as métricas, deve-se determinar como elas se relacionam para obter-se a medida de qualidade do sistema. Para isto, os autores consideraram cada conjunto de métricas acima apresentadas como atributos internos do sistema, a saber: (1) separação de conceitos; (2) acoplamento; (3) coesão; e (4) tamanho. A combinação direta destes atributos permite a aferição de fatores de qualidade que são a compreensão e flexibilidade do sistema. O primeiro é afetado por todos os quatro atributos internos do seguinte modo: (1) a separação de conceitos facilita o entendimento do sistema; (2) componentes muito acoplados dificultam a compreensão, uma vez que é necessário entender vários componentes para que se obtenha a compreensão de um componente; e (3) um baixo grau de coesão interna de componentes dispersa a atenção de um fator específico; (4) um sistema pequeno é mais fácil de ter seu projeto compreendido do que um sistema de maiores proporções. Já com relação à flexibilidade, um único atributo interno não a afeta, é o tamanho. Quanto aos outros, sua relação é a seguinte: uma alta separação de conceitos, um baixo acoplamento e uma alta coesão implicam em componentes altamente especializados, que representam partes com alto grau de independência no sistema. Componentes independentes podem ser substituídos ou alterados sem impacto nas demais partes do sistema.

O casamento dos fatores de qualidade compreensão e flexibilidade é o responsável pela aferição das qualidades de reuso e manutenibilidade do sistema. Quanto à compreensão, um sistema que seja mais facilmente compreendido do que outro tem maiores e melhores chances de ter seus componentes reusados com menor esforço de adaptação e maior facilidade de sofrer manutenção. Já a flexibilidade expressa a capacidade de se conseguir reusar um componente específico do sistema em outro projeto, bem como a facilidade de se realizar a manutenção em uma determinada parte do sistema que pode até mesmo ser substituída por outra sem maiores implicações para o restante do sistema. A Figura 2.9 apresenta o modelo de qualidade com a relação entre as métricas, atributos, fatores e qualidades.

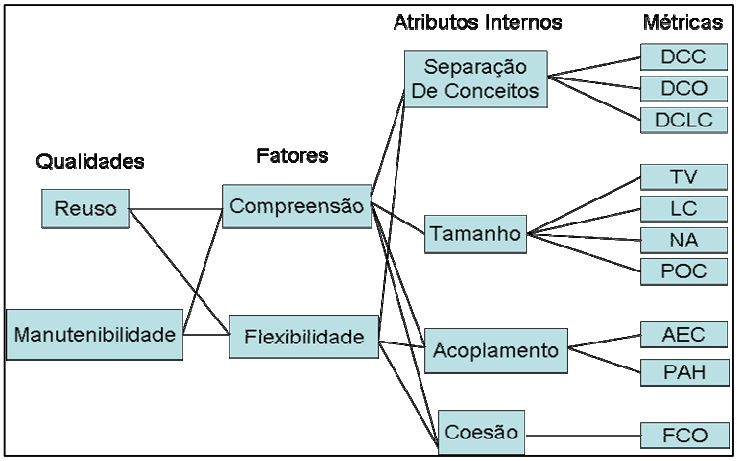

Figura 2.9: Modelo de Qualidade (SANT'ANNA et al 2003).

3 ESTADO DA ARTE

Este capítulo apresenta um estudo sobre o estado da arte na utilização de orientação a aspectos na análise de requisitos e projeto de sistemas computacionais, em especial sistemas TrED. Inicialmente serão analisadas duas propostas de análise de requisitos e num segundo momento serão apresentados dois trabalhos que se concentram na fase de projeto. No último item deste capítulo será apresentada uma metodologia completa que apresenta todas as fases do ciclo de desenvolvimento de um sistema orientado a aspectos. Uma análise mais abrangente pode ser encontrada em (FREITAS, 2006).

3.1 Análise de Requisitos Orientada a Aspectos

A análise dos requisitos envolvidos no projeto de um sistema TrED apresenta fatores que dificultam substancialmente tal tarefa. Um dos principais problemas se refere à representação dos requisitos não-funcionais de maneira clara e acessível. Metodologias que apresentem alguma contribuição no sentido de melhorar este processo de identificação e representação de tais requisitos certamente beneficiam os desenvolvedores da área. A seguir serão apresentadas duas importantes propostas encontradas na literatura.

3.1.1 Análise de Requisitos Orientada a Aspectos Utilizando UML

O objetivo do trabalho analisado nesta subseção é tratar da separação de conceitos e interesses entrelaçados (*crosscutting concerns*) no estágio de análise de requisitos utilizando UML (ARAÚJO, 2002). Para isto, identifica-se e especificam-se tais elementos em módulos separados, de tal forma que se garanta a manutenibilidade e o reuso. O mecanismo de engenharia de requisitos orientado a aspectos baseado em UML tem dois impactos: primeiro ele torna possível a identificação de conflitos entre propriedades que afetam o sistema como um todo logo nas primeiras fases do desenvolvimento; e por ser baseado em UML, um padrão de fato na indústria, permite a incorporação das práticas de engenharia de requisitos.

A proposta da engenharia de requisitos orientada a aspectos integrada a UML segue o modelo de processo de análise que é apresentado na Figura 3.1.

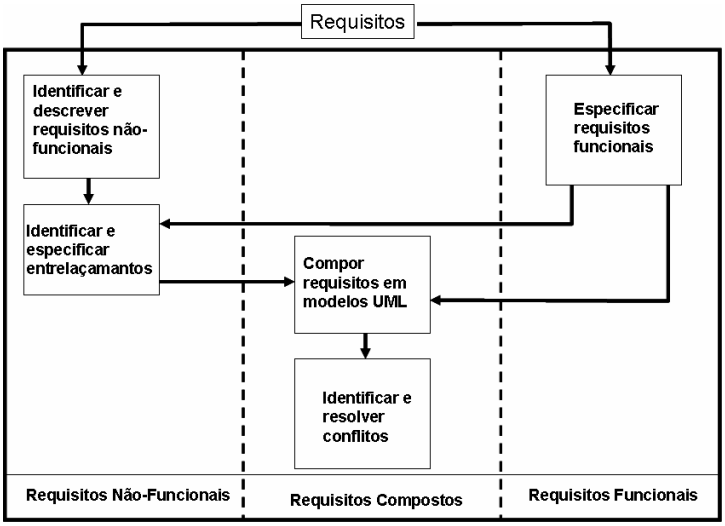

Figura 3.1: Modelo de Requisitos orientados a aspectos (ARAÚJO, 2002).

Os conceitos entrelaçados, parte esquerda da Figura 3.1, tratam os requisitos não-funcionais tentando identificar quais serão candidatos a aspectos. A especificação de um requisito não-funcional segue o *template* apresentado na Tabela 3.1:

Tabela 3.1: Especificação do interesses e conceitos entrelaçados

Conceito ou interesse entrelaçado	<Nome>
Descrição	<Descrição>
Prioridade	<Prioridade (Max, Med, Min)>
Lista de Requisitos	<Requisitos relacionados a este conceito>
Lista de Modelos	<Modelos UML influenciados pelo conceito>

Fonte: ARAÚJO, 2002.

A parte referente à análise funcional, parte direita da Figura 3.1, desempenha o tradicional papel da especificação de requisitos utilizando os modelos UML.

Finalmente, a parte que integra as duas primeiras, requisitos compostos, visualizada na parte central da Figura 3.1, realiza a composição dos requisitos funcionais com os aspectos procurando identificar e resolver possíveis conflitos. Para isto, utilizam-se os seguintes conceitos:

- sobreposição, define a modificação de um requisito funcional por um aspecto que o afeta;

- substituição, define a substituição do comportamento descrito pelo requisito funcional por aquele descrito no aspecto que o afeta; e

- encapsulamento, define que os requisitos de um aspecto encapsulam os requisitos funcionais atravessados pelo aspecto.

Para demonstrar a utilização do processo é apresentado um estudo de caso baseado em um sistema de tarifação por utilização de uma rodovia. O sistema prevê a instalação de um dispositivo no veículo do usuário da rodovia que permitirá a sua identificação ao

entrar ou sair da rodovia e ao passar por postos de controle ao longo da mesma. Ao passar por um dos postos de controle o sistema deve verificar se o automóvel está autorizado a trafegar na rodovia e realizar a tarifação. Deve ainda informar ao motorista quanto será cobrado e acender uma luz verde indicando que o veículo é autorizado. Caso o veículo não esteja autorizado a transitar na rodovia, o sistema deve fotografar a sua placa ao passar por um dos postos de controle e acender uma luz amarela. Ao final de cada mês o sistema deve informar a quantia que deve ser debitada da conta do usuário da rodovia.

Com base nesta descrição do sistema, inicia-se a fase de identificação e separação dos requisitos não-funcionais, uma vez que esses estão misturados com os requisitos funcionais de maneira obscura no texto.

Primeiramente, é importante decidir em que momento o sistema deve agir quando um veículo passa por um posto de controle. Isto depende de um cálculo de distância e velocidade do automóvel em relação a diversos componentes do sistema, tais como sensores de presença, leitores do dispositivo instalado nos carros, displays e câmeras. Com isto levanta-se a questão do posicionamento físico dos elementos do sistema. Outra questão importante refere-se à temporização dos eventos de acendimento das luzes indicativas de autorização (verde e amarela), bem como do valor que será cobrado pela utilização do trecho percorrido. Estes eventos devem ocorrer de tal forma que estas informações visuais sejam apresentadas ao motorista antes dele sair da área do posto de controle. Com isto tem-se levantados os requisitos de tempo de resposta do posto de controle. Esses requisitos podem ser resumidos da seguinte forma:

R1: quando um carro cruzar um posto de controle o sistema deve ler a sua identificação num tempo menor que t1;

R2: veículos não autorizados devem ser fotografados num tempo limite de t2;

R3: ao cruzar um posto de controle o sistema deve acender a luz respectiva num intervalo menor que t3;

R4: quando um veículo autorizado cruzar o posto de controle, o sistema deve informar o valor a ser debitado dentro de um intervalo menor que t4.

A especificação dos requisitos funcionais é feita com a identificação dos atores e casos de uso do sistema. A partir daí constrói-se ainda diagramas de seqüência para representar a dinâmica da utilização do sistema. Os atores identificados foram: proprietário do veículo; o banco; o relógio do sistema e o veículo. Os casos de uso identificados foram: registrar veículo; pagar conta; passar por um posto de controle na entrada da rodovia; passar num posto de controle na saída da rodovia; e passar por um posto de controle ao longo da rodovia.

Com base nas identificações e especificações realizadas, é possível então especificar-se um requisito não-funcional de acordo com o *template* apresentado na Tabela 3.1. A Tabela 3.2 apresenta o exemplo para o tempo de resposta do posto de controle.

Tabela 3.2: Exemplo de especificação de um requisito não-funcional

Conceito ou interesse entrelaçado	Tempo de resposta do posto de controle
Descrição	O posto de controle deve reagir antes da saída do veículo da área do posto
Prioridade	Max
Lista de Requisitos	R1, R2,R3, R4
Lista de Modelos	Casos de uso: 1) passar por um posto de controle na entrada da rodovia; 2) passar num posto de controle na saída da rodovia; e 3) passar por um posto de controle ao longo da rodovia

Fonte: ARAÚJO, 2002.

Os critérios para a integração dos requisitos funcionais e não funcionais são completude e suficiência. O primeiro garante que todos os requisitos que necessitam de suporte estão incluídos no aspecto. O último se refere a garantia de que todo requisito em um aspecto deve ter um impacto no processo de composição.

A representação nos modelos UML se dá através de estereótipos (e.g. wrappedBy), como apresentados no diagrama de casos de uso apresentado na Figura 3.2, e por expressões de restrição de tempo, como apresentado no diagrama de seqüência da Figura 3.3.

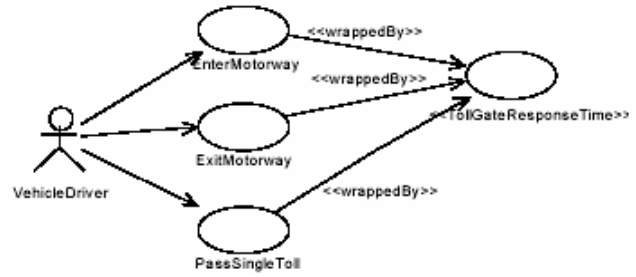

Figura 3.2: Diagrama de Casos de Uso com Aspecto (ARAÚJO, 2002).

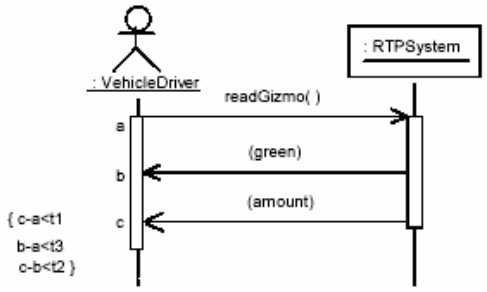

Figura 3.3: Diagrama de Seqüência Evidenciando Restrições Temporais (ARAÚJO, 2002).

Quanto à identificação de resolução de conflitos, pode-se exemplificar com base neste estudo de caso o conflito entre segurança e tempo de resposta do posto de

controle. Para se decidir qual dos dois priorizar, deve-se analisar como esses aspectos influenciam sobre os demais, se positiva ou negativamente, e qual a prioridade que lhes foi atribuída inicialmente. Com base nesta análise e na decisão dos *stakeholders* (partes interessadas no projeto, como por exemplo, futuros usuários, proprietários e mantenedores), prioriza-se um aspecto em relação ao outro conflitante.

3.1.2 Separação de Conceitos Entrelaçados da Análise ao Projeto – Uma Abordagem Dirigida por Casos de Uso

A proposta apresenta em (SOUZA et al, 2004) consiste na adaptação de algumas atividades orientadas a casos de uso do Unified Software Development Process (USDP), de modo a contemplar o estudo, descrição e tratamento de requisitos não-funcionais. Ela é caracterizada com uma proposta de propósito geral, não específica para STrED, mas que apresenta uma contribuição significante ao buscar adaptar ferramentas amplamente utilizadas no desenvolvimento orientado a objetos ao tratamento de requisitos não-funcionais. O trabalho proposto concentra-se nas fases de análise de requisitos e projeto, porém nesta seção será apresentada apenas a contribuição à primeira fase.

Os autores propõem a inserção do tratamento de requisitos não-funcionais no *framework* de atividades de USDP (JACOBSON, 1999), através de uma composição com o NFR *framework*, descrito em (MYNOPOULOS et al, 2001). O primeiro *framework* descreve atividades de estudo do domínio do problema, identificação dos atores e casos de uso, especificação dos casos de uso e estruturação dos mesmos. No NFR *framework*, as atividades consistem na decomposição dos conceitos e preocupações não-funcionais, identificação de possíveis maneiras de se operacionalizá-los, identificação de correlações e prioridades, e finalmente a escolha de como serão operacionalizados. A composição destes dois *frameworks* leva à proposta apresentada na Figura 3.4.

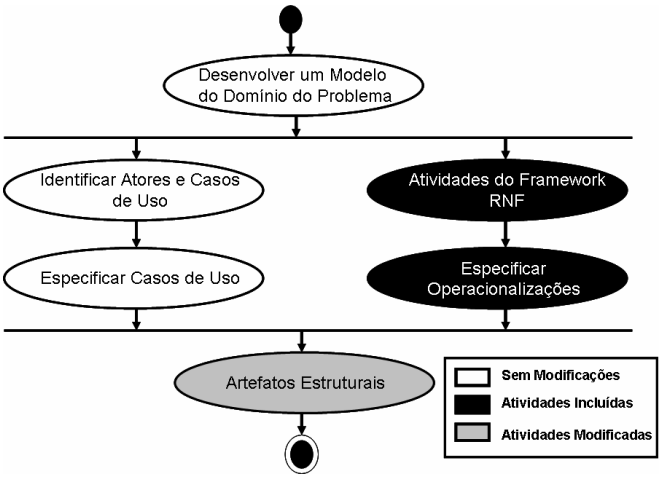

Figura 3.4: Atividades Propostas para a Fase de Análise de Requisitos (SOUZA et al, 2004).

Observando a Figura 3.4, verifica-se a introdução de atividades especificas para o tratamento de requisitos não-funcionais, que são destacadas em fundo preto. Já as atividades atinentes ao tratamento dos requisitos funcionais (fundo branco), continuam inalteradas. A última atividade que trata da estruturação é modificada devido a inclusão do tratamento não-funcional.

A representação do tratamento não-funcional no diagrama de casos de uso é feita através de casos de uso dedicados a tais requisitos, chamados *crosscutting*. Estes casos de uso especiais são então conectados aos casos de uso convencionais, dedicados aos requisitos funcionais, através de um relacionamento chamado *crosscuts*. As informações referentes à composição entre os casos de uso *crosscutting* e os casos de uso afetados são descritas em uma tabela, como apresentado na Tabela 3.3.

Tabela 3.3: Tabela de Composição para um Caso de Uso Crosscutting

CROSSCUTTING USE CASE: #N <NOME>			
Caso de Uso Afetado	Condição (Opcional)	Operador da Regra de Composição	Ponto Afetado
#N <nome>	Condição da extensão	{overlap.after | overlap.before | override | wrap}	Passo do Cenário

Fonte: SOUZA et al, 2004

A primeira coluna da tabela de composição lista todos os casos de uso afetados pelo caso de uso *crosscutting* identificado no topo da tabela. A segunda coluna descreve as condições da composição, enquanto a terceira determina como o comportamento do caso de uso *crosscutting* de ser aplicado ao caso de uso afetado. A última coluna especifica o ponto do caso de uso no qual o comportamento deve ser aplicado.

O estudo de caso de um sistema de Internet Banking exemplifica a utilização dos conceitos. Pelo sistema o usuário pode realizar operações de visualização do status da conta, transferência de valores e pagamento de contas. Um dos principais requisitos não-funcionais ligados ao sistema consiste na segurança para se realizar as transações. Após sucessivas decomposições, foram selecionadas as seguintes formas de operacionalização deste requisito: Limitação no valor de transações; Firewall; Criptografia de dados; Identificação; Conferência da senha de internet, Conferência dos dados do usuário; Duplicação de Servidores e Espelhamento da base de dados. Um exemplo de especificação de operacionalização para a conferência de senha de Internet é apresentado na Tabela 3.4.

Tabela 3.4: Especificação da operacionalização da Conferência de Senha de Internet

Operationalização #05 – Check Internet Password	
CENÁRIO PRINCIPAL	
Passo	Ação
1	O ator informa a senha de internet
2	O sistema compara a senha informada com a senha armazenada para a conta
3	O sistema retorna resultado da comparação

Fonte: SOUZA et al, 2004.

Especificados os requisitos funcionais e não-funcionais, constrói-se o diagrama de casos de uso utilizando-se os mecanismos de generalização, inclusão, extensão e *crosscuts* para se relacionar os casos de uso identificados. A Figura 3.5 apresenta o diagrama de casos de uso para o estudo de caso do Internet *Banking*.

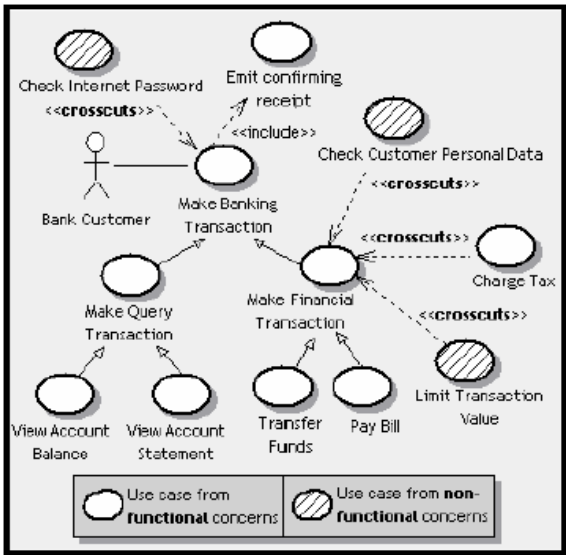

Figura 3.5: Diagrama de Casos de Uso Estruturado (SOUZA et al, 2004).

Para cada relacionamento *crosscuts* deve ser especificada a composição entre o caso de uso *crosscutting* e o afetado. Para isto foi criada a tabela de composição apresentada na Tabela 3.5. Na referida tabela é apresentado um exemplo de composição para a conferência de senhas de Internet.

Tabela 3.5: Tabela de Composição para Conferência de Senha de Internet

CROSSCUTTING USE CASE: #05 - Check Internet Password			
Caso de Uso Afetado	Condição (Opcional)	Operador da Regra de Composição	Ponto Afetado
#01 – Visualizar Balanço da Conta	-	overlap.before	Passo 1
#02 – Visaulaizar Estado da Conta	-	overlap.before	Passo 2
#03 – Transferência de Fundos	-	overlap.after	Passo 1
#04 – Pagamentos	-	overlap.after	Passo 1

Fonte: SOUZA et al, 2004.

3.2 Modelagem de STrED Orientada a Aspectos

Outro aspecto que preocupa a comunidade que trabalha no desenvolvimento de sistemas TrED é a questão da modelagem destes sistemas. Se os requisitos não-funcionais já apresentam dificuldade de serem identificados e descritos na fase de análise, o seu tratamento encontra igual ou maior dificuldade de ser expressa na fase da modelagem do sistema. Geralmente este tratamento encontra-se entrelaçado e obscuramente espalhado por diversas entidades do projeto, tornando sua mudança e evolução um verdadeiro desafio.

3.2.1 VEST: Uma Ferramenta de Composição para Sistemas de Tempo-Real Baseada em Aspectos

VEST (STANKOVIC et al, 2003) é um conjunto de ferramentas desenvolvido na Universidade de Virginia pelo grupo liderado por Stankovic, que tem como objetivo a checagem de dependências baseada no conceito de aspectos em sistemas embarcados tempo-real distribuídos.

Com foco no reuso, os autores propõe a utilização de componentes pré-escritos de uma biblioteca para se desenvolver um novo sistema embarcado tempo-real, ao invés de implementar tudo desde o início. Porém, a realização da composição de um sistema embarcado apresenta dificuldades muito especificas relativas às dependências sobrepostas, ou entrelaçadas (*crosscutting*) entre os componentes utilizados. Desta forma, é necessário que se tenha um mecanismo de composição eficiente, identificando as dependências e tornando possível a análise dos requisitos não-funcionais envolvidos no projeto. Os autores estendem a noção de aspectos criando dois tipos de aspectos independentes de linguagem chamados "*aspect checks*" e "*prescriptive aspects*". Com eles é possível utilizar as potencialidades dos aspectos no processo de composição ao invés de utilizá-los somente na implementação.

O conjunto de ferramentas inclui quatro bibliotecas de componentes. Nestas bibliotecas podem-se encontrar componentes de software, descrições de hardware e redes. Os componentes podem ser abstratos ou reais. Os primeiros são projetos que representam requisitos, enquanto os últimos são implementações ou descrições que podem ser reutilizadas. Existe um conjunto de informações reflexivas para cada tipo de componente. Essas informações são utilizadas para a análise de alguma dependência que possa existir. Existe ainda uma biblioteca de *prescriptive aspects* que contem um conjunto de *advices* independente de linguagem que pode ser utilizado no projeto. Outro recurso constitui-se do conjunto de *aspect checks* que representam aspectos intra e inter-componentes que podem ser utilizados para se descobrir erros de dependência entre componentes.

Prescriptive aspects contém *advices* que podem ser utilizados na adaptação do projeto. Eles podem ajustar propriedades tais como prioridades de tarefas ou o nível de replicação de um componente de software.

Já a verificação de aspectos (*Aspect checking)* realiza a conferência de dependências entre os componentes de forma a evitar erros na composição de um sistema embarcado. Algumas checagens são simples de se realizar, porém existem outras que apresentam uma grande dificuldade dado o grande número de componentes afetados e a profundidade em que são afetados. Desta forma, realiza-se a conferência através dos componentes em busca de erros relativos às dependências geradas por requisitos não-funcionais de maneira automática.

Uma verificação importante realizada pela ferramenta, corresponde à análise de escalonabilidade para garantir que todas as tarefas cumpram seu *deadline*. Enquanto se realiza o projeto e a implementação do projeto, isto pode alterar propriedades do sistema como um todo. Desta forma, esta análise ocupa-se de uma dependência global do sistema.

Dois estudos de caso são apresentados para demonstrar a utilização da ferramenta. Ambos se referem à composição de uma parte de um sistema aviônico distribuído baseado em um *middleware*. O primeiro se atem às vantagens da análise e composição

utilizando a ferramenta. São analisadas questões referentes à utilização de memória e escalonabilidade de tarefas. Já o segundo traz informações sobre tempo gasto para composição do sistema utilizando a ferramenta, comparado com o mesmo desenvolvimento sem o uso de VEST. Os resultados apresentados mostram um sensível ganho de produtividade através do uso da metodologia e do conjunto de ferramentas VEST.

3.2.2 Modelagem OA de Sistemas de Tempo-Real Baseado em UML

A proposta do trabalho dos autores (ZHANG; LIU, 2005) é separar todo tratamento não-funcional do sistema em aspectos específicos para cada tipo de requisito. Com foco em sistemas de tempo-real, a descrição das características, condições e restrições temporais é concentrada em um aspecto exclusivo destas propriedades. A contribuição proposta é utilizar UML para expressar esta separação do tratamento de requisitos em elementos de modelo.

Para representar o elemento que concentra os requisitos de tempo-real no modelo de classes UML, estendeu-se a linguagem através de um estereotipo <<aspect>> que representa um aspecto. Um aspecto se relaciona com um componente funcional (classe) do sistema, chamado de *core*, através de um relacionamento estereotipado <<crosscut>> que identifica que aquele aspecto afeta aquela classe. A Figura 3.6 mostra a relação entre classe e aspecto.

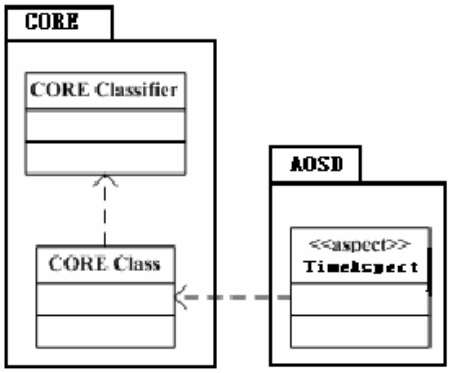

Figura 3.6: Relacionamento entre Classe e Aspecto (ZHANG; LIU, 2005).

A separação do tratamento dos requisitos funcionais dos não-funcionais torna o projeto do sistema mais simples de ser reusado, uma vez que toda a parte funcional encontra-se modularizada e livre de entrelaçamentos. Desta forma, os autores advogam o argumento de que o uso por eles proposto de um aspecto que retire todo tratamento não-funcional de uma classe acaba tornando a parte funcional mais fácil de ser reusada.

Todos os requisitos temporais são capturados e reunidos no aspecto *TimeAspect*. A Figura 3.7 apresenta o diagrama de classes de um sistema de controle de elevador modelado de acordo com a proposta.

Deve-se ressaltar que todas as restrições temporais, como tempo máximo para abertura da porta, encontram-se anotadas em uma *tag* associada ao aspecto *TimeAspect*. Na Figura 3.7 pode-se ainda observar o relacionamento daquele aspecto com a classe *ControlSystem*, mostrando que este elemento funcional do sistema é afetado pelo aspecto de tempo.

Além do modelo estrutural, o trabalho ainda apresenta a representação das restrições temporais em uma descrição comportamental do sistema, que corresponde a um diagrama de estados, onde tais restrições são anotadas nas transições entre estados. Este diagrama busca refinar a descrição estrutural apresentada no diagrama de classes. Neste diagrama, assim como no diagrama de classes, as restrições temporais também são apresentadas em uma *tag* associada a elementos do modelo.

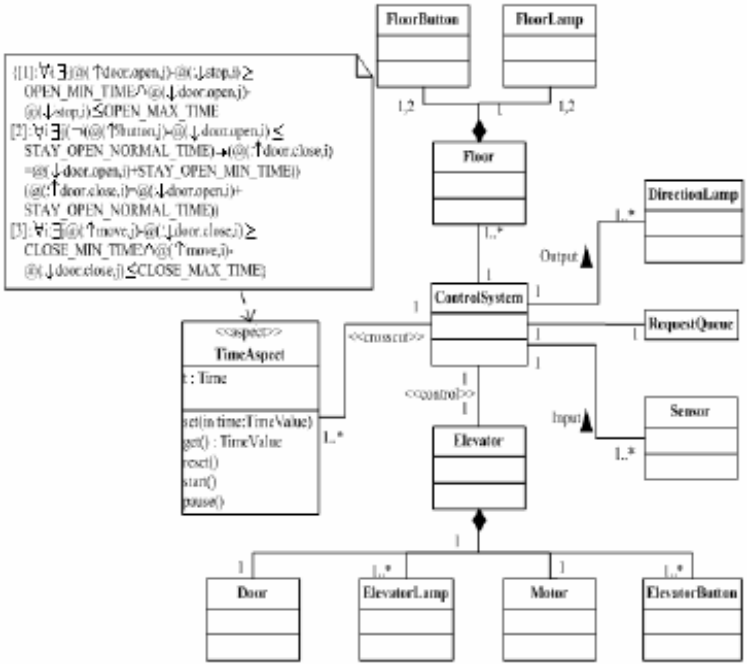

Figura 3.7: Diagrama de Classes do Sistema de Controle de Elevador (ZHANG; LIU, 2005).

De fato o trabalho apresenta uma contribuição no sentido de isolar o tratamento de requisitos não-funcionais em um elemento distinto do projeto. No entanto, vê-se que não houve preocupação com a organização do tratamento não-funcional em si, pois este foi representado por uma *tag* que se associa ao elemento que representa a dimensão não-funcional. Os requisitos não-funcionais não se encontram representados em elementos do modelo, mas sim anotados como um comentário.

3.3 Metodologia de Desenvolvimento Orientada a Aspectos

Nesta seção será apresentada uma metodologia orientada a aspectos que trabalha a separação da preocupação com requisitos não-funcionais desde a fase de análise até o projeto e início da implementação do sistema.

3.3.1 A Metodologia FRIDA

FRIDA (From RequIrements to Design using Aspects) é uma metodologia que tem como objetivo balizar o processo de desenvolvimento de software utilizando os conceitos da orientação a aspectos (BERTAGNOLLI, 2004). Ela é uma metodologia originalmente aplicada a sistemas tolerantes a falhas, porém a flexibilidade de seu

conjunto de ferramentas torna possível a sua utilização em outros domínios. A metodologia de FRIDA se concentra em como a modelagem separada dos requisitos não-funcionais pode auxiliar na análise dos requisitos funcionais e componentes. A metodologia segue alguns passos para alcançar seus objetivos, começando pela identificação dos requisitos, como apresentado na descrição a seguir.

A primeira fase constitui-se da análise do problema. Baseado nesta análise se constrói o diagrama de casos de uso. Para cada ocorrência de caso de uso é associado um *template* que o descreve em detalhes, apresentado informações tais como: objetivo, pré-condições, pós-condições, prioridade, atores envolvidos, exceção, variações, dentre outros.

A segunda fase constitui-se justamente da identificação dos requisitos não-funcionais. Para realizar esta tarefa, utiliza-se um artefato chamado *checklist*. Esta *checklist* é desenvolvida para cada tipo de requisito não-funcional, como por exemplo, desempenho. Outra tarefa importante desta fase constitui-se na ordenação em prioridades de atendimento dos requisitos não-funcionais para facilitar uma futura resolução de possíveis conflitos entre eles.

A terceira fase tem por objetivo esclarecer alguns requisitos não-funcionais que permaneceram obscuros após a identificação ocorrida na segunda fase. Para isto utiliza-se um processo léxico descrito em (LEITE, 1995) e a Backus Naur Form (BACKUS; NAUR, 1969) para defini-la. O léxico é similar a um glossário, com palavras-chave relacionadas ao domínio do problema no qual o requisito não-funcional se insere. O casamento de informações contidas no léxico com as *checklists* permite identificar os requisitos não-funcionais candidatos e daí decidir se eles são ou não RNFs a serem tratados.

A quarta etapa consiste na identificação e resolução de conflitos entre RNFs. Esta identificação pode ser feita através de um conjunto de regras que definem possíveis alterações e impactos de um requisito em outro. Com base nestes resultados determina-se o possível conflito. Pode-se a partir daí montar-se uma matriz que exibe quais requisitos conflitam. Para se resolver o problema de conflitos utiliza-se a prioridade estabelecida previamente para cada requisito. Caso dois requisitos conflitantes tenham a mesma prioridade, os *stakeholders* envolvidos devem ser questionados para se solucionar o impasse. Um exemplo claro de conflito seria, por exemplo, o contraste entre um requisito de desempenho e segurança, uma vez que esta gera um overhead no sistema, impactando negativamente o seu desempenho.

A fase subseqüente, a quinta, consiste no projeto da parte funcional do sistema. Nela, os conceitos estabelecidos na especificação e análise dos requisitos são transformados em classes. Para se realizar as tarefas desta etapa, utiliza-se a linguagem UML.

É muito importante que se mantenham elos entre os diagramas gerados nesta fase de tal forma a se manter a rastreabilidade com requisitos, sejam eles funcionais ou não. A criação do elo entre requisitos funcionais e classes consiste na sexta fase do método. Cada classe do modelo deve estar associada à pelo menos um caso de uso do sistema. Ela só pode existir se estiver ligada a um requisito funcional. Para cada classe gera-se uma descrição sintática que segue um *template* com informações relativas ao(s) caso(s) de uso e ator(es) associados a ela.

A sétima fase trata da definição, ou extração dos aspectos. Com base em todas as informações geradas nas fases anteriores, principalmente na segunda, terceira e quarta, realiza-se a definição dos aspectos. A representação visual dos aspectos corresponde à oitava fase, na qual se utilizam estereótipos que estendem a linguagem UML. Estes estereótipos definem o aspecto e seus campos, bem como seus métodos, *pointcuts* e *advices*. Nesta fase também se realiza a descrição textual dos *joinpoints*.

Terminado mapeamento dos aspectos e seus elementos, faz-se necessário estabelecer as relações entre as classes e os respectivos aspectos, o que caracteriza a nona fase. Estes elementos estão desconexos e precisam de um elo que os relacione. Para realizar esta conexão, utilizam-se as ligações entre os diagramas descritos na sexta etapa, que permitem a rastreabilidade dos requisitos. Com base nestas informações e nos *templates* que integram aspectos funcionais e não-funcionais gerados na quinta fase, é possível se identificar as correlações entre classes e aspectos e dessa maneira realizar a sua associação. Desta forma, cada classe ficará associada a um ou mais aspectos completando, portanto a modelagem do sistema.

E finalmente como fechamento da metodologia, existe a geração de código do sistema, que consiste basicamente na geração da estrutura básica dos elementos de projeto (classes e aspectos).

4 METODOLOGIA ORIENTADA A ASPECTOS PARA A ESPECIFICAÇÃO DE STREDS

Buscando responder a necessidade de se tratar os requisitos não-funcionais dos STrED de maneira a gerar um projeto que corresponda às necessidades levantadas, é necessário que a preocupação com os requisitos não-funcionais seja expressa desde o princípio do ciclo de vida do sistema. Esta necessidade motivou a adaptação de uma metodologia orientada a aspectos contextualizada ao domínio dos STrED. A metodologia adaptada, batizada de RT-FRIDA (Real-Time FRIDA), tem como base a FRIDA (From RequIrements to Design using Aspects) (BERTAGNOLLI, 2004), que é uma metodologia de propósito geral preocupada com requisitos não-funcionais desde a fase de análise e que fornece um ferramental que facilita o mapeamento destes requisitos em aspectos.

A justificativa para a escolha de FRIDA se resume basicamente no fato desta metodologia apresentar flexibilidade na utilização de suas ferramentas a outros domínios e ainda permitir o mapeamento de requisitos não-funcionais em elementos de projeto. Portanto, escolheu-se esta metodologia como base para a proposta do presente trabalho. Outro motivo que sustenta a escolha de FRIDA foi a decisão de se utilizar uma metodologia tradicional de desenvolvimento, segundo a classificação apresentada em (KHAN; JAFFAR-UR-REHMAN, 2005) já discutida na seção 2.4.2. No mencionado trabalho foi apresentada uma divisão entre metodologias de desenvolvimento orientadas a aspectos baseadas em abordagens tradicionais e não-tradicionais de engenharia requisitos. FRIDA se enquadra no grupo das abordagens tradicionais e se baseia em cenários e casos de uso, o que facilita a sua utilização por aqueles já acostumados com metodologias tradicionais orientadas a objetos, como é o caso da comunidade desenvolvedora de sistemas TrED.

A adaptação de FRIDA foi feita através da contextualização de suas ferramentas aos conceitos envolvidos no projeto de sistemas TrED; de uma reestruturação dos passos originais da metodologia; e da definição de uma biblioteca extensível de aspectos de alto nível (DERAF High-level Aspects) a serem utilizados para o tratamento dos requisitos não-funcionais referentes aos sistemas TrED. O objetivo final é que a metodologia adaptada, a RT-FRIDA, consiga elicitar claramente os requisitos não-funcionais referentes a questões de tempo, distribuição e embarcados mapeando o seu tratamento em aspectos da biblioteca proposta e por fim sendo integrada ao *framework* do projeto SEEP. A primeira tarefa foi realizada através da identificação de requisitos pertinentes ao domínio de interesse do trabalho, buscando adaptar a semântica das ferramentas disponíveis na metodologia original a estes requisitos. Já a reestruturação dos passos buscou simplificar a utilização da metodologia, agregando algumas idéias encontradas em trabalhos consultados na literatura (ARAUJO, 2002), (STEIN et al,

2002) e (STEIN et al, 2006). A definição da biblioteca extensível de aspectos de alto nível, bem como a integração ao framework do projeto SEEP, foram resultado de um trabalho em conjunto com o doutorando Marco Aurélio Wehrmeister.

4.1 Classificação de Requisitos Não-funcionais para Sistemas TrED

O primeiro passo para se adaptar FRIDA ao contexto dos sistemas TrED, constitui-se na identificação de um núcleo de requisitos relevantes correlatos ao domínio. Com isto, alguns dos principais requisitos envolvidos no projeto de sistemas TrED foram levantados e organizados como apresentado na Figura 4.1. O levantamento destes requisitos, bem como sua descrição se baseia nos Glossário da IEEE (IEEE, 2006), no Glossário da SEI (SEI, 2005), no estudo apresentado em (BURNS, 1997) e (BURNS, 2000).

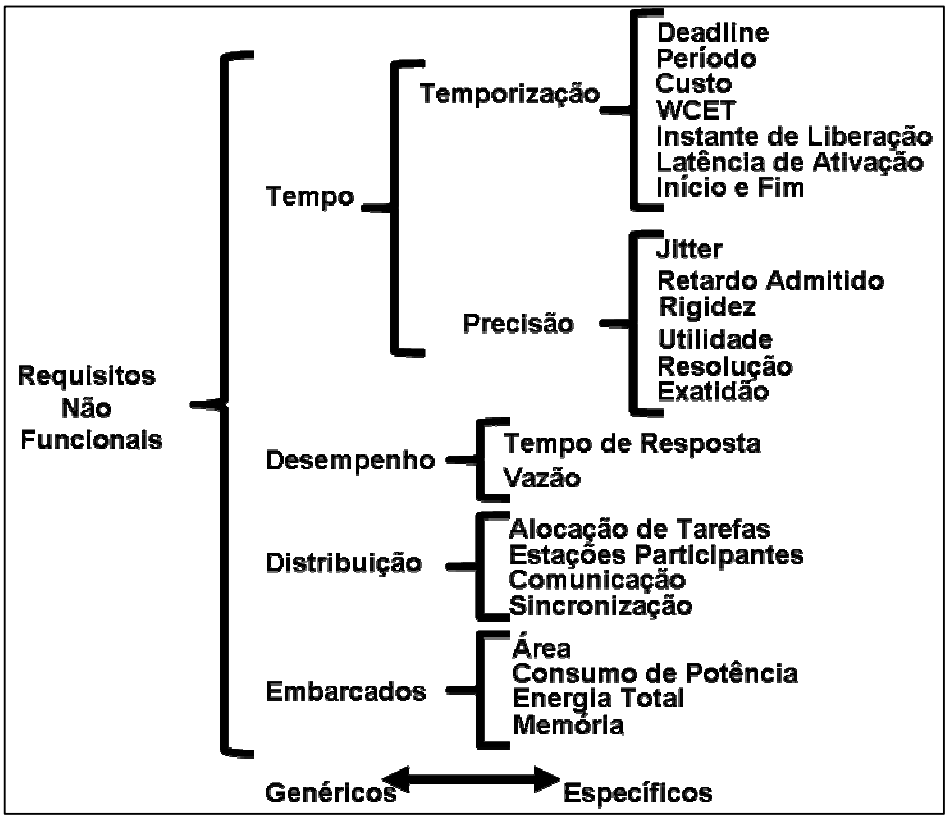

Figura 4.1: Classificação de RNFs proposta para o domínio TrED.

Antes de passar a descrição de cada requisito, é importante ressaltar a existência de interação direta entre eles. Esta relação de influência será alvo de estudo quando se tratar do conflito de requisitos na descrição da metodologia. Outra observação importante se refere ao escopo dos requisitos. Alguns se aplicam ao sistema, ou seja, tem um escopo mais amplo, como a latência de uma resposta final do sistema, enquanto outros se referem aos elementos do sistema de forma individualizada, como o *deadline* da execução de uma tarefa específica, por exemplo. Este escopo deve ser detalhado na especificação de cada requisito. Optou-se por manter o nome de alguns destes requisitos em língua inglesa, como *deadline* e *jitter*, devido ao seu corrente uso desta forma mesmo em documentação escrita em outro idioma.

4.1.1 Requisitos de Tempo

Temporização

Os RNFs levantados durante o estudo da problemática de sistemas TrED e com base na bibliografia apresentada, revelam alguns requisitos de fácil compreensão, como os listados na subclassificação Temporização. Estes requisitos são diretamente mapeados em propriedades que regem a execução de atividades num sistema TrED.

Segue uma descrição sucinta de cada requisito de temporização:

a) **Deadline**: limite temporal para a execução de uma determinada atividade do sistema;
b) **Período**: intervalo entre duas ativações sucessivas de uma atividade;
c) **Custo (Tempo de Execução)**: tempo médio necessário para se executar uma atividade no sistema;
d) **WCET**: este conceito segue seu entendimento convencional, sendo o tempo de execução do pior caso;
e) **Instante de Liberação**: instante em que o sistema deve estar pronto para executar determinada atividade;
f) **Latência de Ativação**: corresponde ao tempo entre o instante de liberação e o instante no qual uma atividade começa a ser executada;
g) **Início e Fim**: instantes de início e fim da execução de uma atividade;

Precisão

Os requisitos classificados como de precisão necessitam de uma maior reflexão para sua compreensão, uma vez que apesar de também refletirem propriedades de execução das atividades de um sistema, eles podem ser mais abrangentes, envolvendo (entrelaçando) diversas propriedades. Quando se trata de *jitter*, por exemplo, pode-se restringir ao *jitter* especifico associado a uma determinada tarefa ou ao *jitter* de uma resposta final do sistema.

Descrição dos requisitos:

a) **Jitter**: variação na medida de determinado requisito temporal. Quanto maior a variação do *jitter*, menor a previsibilidade do sistema;
b) **Retardo Admitido**: tempo excedente máximo admitido para o início da execução de uma atividade do sistema;
c) **Rigidez**: ela define a precisão de outros conceitos, como o de deadline por exemplo, que pode ser *hard* ou *soft*. Indica, portanto, as conseqüências do não atendimento do requisito temporal. Quanto à utilidade de um dado, a questão do prazo de sua validade pode seguir uma gradação entre os extremos *soft* e *hard*, significando a perda gradativa da validade de utilização (utilidade) do dado;
d) **Utilidade (Prazo de Validade)**: é um aspecto que pode influenciar diversos elementos do sistema. Ele é associado à idéia de lapso temporal no qual o valor de um dado/variável é considerado válido. Um exemplo seria o dado lido por um sensor. Este dado pode ter sua validade restrita a um lapso temporal, de tal forma que expirando este prazo, o dado se torna inutilizável.
e) **Resolução**: identifica a granularidade de tempo mais fina na qual um elemento de temporização do sistema pode trabalhar.

f) **Exatidão**: refere-se ao desvio máximo entre o tempo físico e o tempo lógico utilizado efetivamente pelo sistema computacional.

4.1.2 Desempenho

Analisados os requisitos de aspecto temporal, deve-se ter atenção a alguns outros requisitos com os quais tem interação direta; tais requisitos são especificados na classificação de desempenho. As restrições de vazão e latência, além de influenciarem os requisitos temporais, vão se relacionar de maneira determinante com a questão da distribuição no sistema. Por apresentarem este relacionamento entre aspectos distintos de um sistema TrED, optou-se por criar uma classificação separada para critérios de desempenho e não inserí-los nos macro-requisitos de tempo ou distribuição.

Descrição dos requisitos:

a) **Vazão**: taxa na qual um recurso deve executar sua função, ou quantas chamadas por unidade de tempo um recurso deve ser capaz de tratar;
b) **Tempo de Resposta**: representa o tempo necessário para que o sistema retorne uma resposta final que dependa da execução de atividades locais ou remotas;

4.1.3 Distribuição

A especificação relacionada à distribuição é mais complexa do que o apresentado nesta proposta. Porém, para os propósitos deste estudo, os requisitos apresentados na Figura 4.1 apresentam maior relevância.

Segue a descrição dos requisitos de distribuição levantados:

a) **Alocação de Tarefas**: este requisito se relaciona à questão da distribuição e escalonabilidade das tarefas nas diferentes unidades de processamento (“hosts” ou estações participantes). Este requisito deve ser levado em consideração, principalmente quando associado a outros requisitos não-funcionais, como segurança, por exemplo. Quando entrelaçado à segurança, pode-se ter a necessidade de se exigir que determinadas atividades ou tarefas críticas do sistema sejam executadas em determinadas estações preferencialmente.
b) **Estações Participantes**: constitui os requisitos de monitoramento de estações participantes do sistema. Este critério encontra-se relacionado às questões que envolvem a alocação de tarefas. Outro ponto importante a se tratar é a possibilidade de se realizar auditorias e verificações do estado momentâneo dos diversos elementos do sistema.
c) **Comunicação**: este requisito engloba as restrições relativas à comunicação no sistema, tais como exigência de utilização de determinado barramento ou tecnologia especifica, qual tipo de comunicação desejada, se confiável (com garantia de entrega de mensagem) ou não, ou ainda a vazão associada à troca de mensagens ou transmissão de dados;
d) **Sincronização**: definição das políticas e restrições de acesso a recursos do sistema.

4.1.4 Embarcados

Quanto aos requisitos relativos às restrições do sistema ser embarcado, buscou-se focar os aspectos mais relevantes no escopo do trabalho ora desenvolvido. Com isso,

quatro requisitos foram levantados, como apresentado na Figura 4.1. Sua descrição é a que segue:

a) **Área**: corresponde à restrição de área máxima (em silício ou placa de circuito impresso) que pode ser ocupada por determinado componente do sistema;
b) **Consumo de Potência**: corresponde à restrição de consumo de potência associada a componentes do sistema;
c) **Energia Total**: quanto de energia seria disponibilizado para o sistema. Este requisito trata ainda de quanto deve durar a energia disponibilizada para o sistema. Esta medida de duração pode ser temporal, por número de operações que o sistema deve realizar, ou em distância que deve percorrer, por exemplo;
d) **Memória**: corresponde às restrições quanto ao uso de memória no sistema.

É possível verificar um eminente foco de conflito entre requisitos desta classe, uma vez que dada à energia total disponibilizada e a duração desejada de seu uso, seu consumo será naturalmente afetado. Porém, pode haver casos em que para se atender determinados requisitos de outras classes, como desempenho, seja necessário manter o consumo de componentes do sistema até certo limite. Com isso pode-se escolher qual requisito ligado à questão de energia expressa melhor a necessidade que está sendo analisada. Neste ponto identifica-se o conflito entre requisitos, assunto que será tratado oportunamente na descrição da metodologia adaptada, na fase de identificação e especificação de requisitos.

4.2 Descrição da Metodologia Adaptada

Devido à adoção de *templates* também para a descrição de requisitos não-funcionais e visando ainda facilitar a utilização da metodologia RT-FRIDA, a presente proposta apresenta-se divida em três fases principais que mantém correspondência com as fases da metodologia original, como descrito a seguir:

a) Primeira fase: Identificação e Especificação de Requisitos – corresponde as fases de 1 a 4 da metodologia original;
b) Segunda fase: Mapeamento de Requisitos em Elementos de Projeto – corresponde as fases 6 e 7 da metodologia original;
c) Terceira fase: Projeto do Sistema – corresponde as fases 5, 8 e 9 da metodologia original.

Naturalmente estas fases são subdivididas em etapas ou passos que devem ser seguidos de modo a se atingir o objetivo a que se propõe a respectiva fase.

4.2.1 Primeira Fase: Identificação e Especificação de Requisitos

Esta primeira fase pode ser subdividida basicamente em duas etapas: (1) identificação e especificação de requisitos funcionais, e (2) identificação e especificação de requisitos não-funcionais.

Na primeira etapa, o primeiro passo é a identificação das funcionalidades desejadas para o sistema e a construção do diagrama de casos de uso. Após a construção do diagrama, deve-se preencher um *template* de requisitos funcionais que detalha a informação apresentada graficamente no diagrama de casos de uso. A Figura 4.2 apresenta o referido *template*.

Após a descrição dos requisitos funcionais através do *template*, passa-se à identificação e resolução de conflitos entre requisitos desta natureza. Para isto monta-se uma matriz relacionando todos os requisitos funcionais levantados, assinalando a célula no encontro de dois requisitos conflitantes. Um exemplo desta tabela destacando o conflito entre dois requisitos pode ser observado na Tabela 5.1 do estudo de caso apresentado na Seção 5.1. A decisão de qual requisito deve prevalecer é baseada na prioridade atribuída a cada requisito. Caso os requisitos conflitantes tenham a mesma prioridade, deve-se consultar os *stakeholders* para a solução do conflito. Para se identificar o conflito entre dois requisitos, utiliza-se o conjunto de regras matematicamente escritas que definem quando o impacto da mudança de um requisito se traduz em conflito. Estas regras foram definidas em (BERTAGNOLLI, 2004) e encontram-se listadas no Anexo A.

		Item	Descrição
Geral	**Identificação**	**Identificador**	Identificação que permitirá a rastreabilidade do requisito por todo o projeto.
		Nome	Nome do caso de uso associado ao requisito.
		Objetivo	Descrição dos objetivos do caso de uso.
		Autor	Pessoa responsável pela definição.
	Contexto	**Pré-condição**	Estado em que o sistema deve se encontrar antes que este caso de uso possa ser executado.
		Pós-condição	Estado no qual o sistema deve se encontrar após o cenário primário ter sido finalizado.
		Ator primário	Ator considerado a fonte dos eventos que estimulam a execução do cenário primário.
		Ator secundário	Um ator que interage passivamente com o caso de uso, mas não executa nenhuma ação sobre o mesmo.
	Decisão e Evolução	**Prioridade**	Usada para decidir a importância relativa entre os casos de uso. Ela assume um dos seguintes valores: sem prioridade, baixa, média, alta ou crítica.
		Situação	Um requisito pode estar em uma das seguintes situações: 0 - identificado; 1 - analisado; 2 - especificado; 3 - aprovado; 4 - cancelado; 5 - finalizado;
Caminhos		**Primário (Normal)**	Descreve o fluxo principal do caso de uso, sem condições de erro, apenas com resultados positivos.
		Alternativo	Apresenta o fluxo de andamento alternativo para a funcionalidade do caso de uso.
		Excepcional	Apresenta a descrição do fluxo de andamento do caso de uso em uma situação atípica.
Cenários		**Principal**	Descrição dos passos principais que envolvem o cenário onde se insere o caso de uso.
		Variações	Os passos descritos como variações são aqueles que modificam um ou mais passos do cenário principal.

Figura 4.2: *Template* para Requisitos Funcionais

A segunda etapa, que se dedica aos requisitos não-funcionais, apresenta um maior número de passos.

Primeiramente é necessário que seja feita a identificação dos requisitos não-funcionais envolvidos no sistema. Para isto utiliza-se um conjunto de questões

especificamente elaboradas para o domínio de interesse, que constituem o artefato chamado *checklist*. Neste trabalho foram desenvolvidas quatro *checklists*, uma para cada classe de requisitos não-funcionais apresentadas na Figura 4.1. Cada *checklist* apresenta questões que dizem respeito aos requisitos elencados em cada uma das quatro classes. O uso deste artefato permite o levantamento de informações como, relevância, prioridade, condições, descrição e restrições envolvendo o requisito. Um modelo genérico de organização de uma *checklist* é apresentado na Figura 4.3, na qual se pode observar na primeira coluna o espaço reservado às questões de inferência, divididas segundo as classificações genéricas e específicas descritas na classificação de requisitos apresentada na Figura 4.1. O uso deste artefato é feito através da resposta aos questionamentos apresentados nesta primeira coluna. As respostas a estas questões definirão se os requisitos em questão estão ou não presentes no sistema. Na seqüência, deve-se ponderar sobre a relevância ou não de tais requisitos, caso sejam relevantes, deve-se marcar a quadrícula referente à relevância na segunda coluna. Sendo o requisito relevante, deve-se então definir a prioridade do seu atendimento no respectivo espaço para isto reservado na terceira coluna. Esta prioridade pode ser baixa, média ou alta. Finalmente preenche-se a quarta coluna com detalhes sobre condições, restrições ou uma breve descrição do requisito.

	Relevante	**Prioridade**	**Restrição/Condição/Descrição**
Classificação Genérica			
Classificação Específica			
Pergunta de Inferência			

Figura 4.3: Modelo de Organização de uma *Checklist*

Como mencionado anteriormente, foram desenvolvidas quatro *checklists* de modo que cada uma atenda as demandas de uma das classes de requisitos não-funcionas de sistemas TrED. A *checklist* dedicada ao levantamento de requisitos temporais é dividida em duas *sub-checklists*, uma específica sobre questões relativas à temporização do sistema e outra específica sobre a precisão no atendimento de requisitos temporais. A *checklist* dedicada aos requisitos temporais pode ser observada na Figura 4.4 e 4.5.

	Relevante	**Prioridade**	**Restrições Condições Descrição**
Tempo			
Temporização			
Existem atividades ou amostragens periódicas?			
Existem atividades esporádicas?			
Existem atividades aperiódicas?			
Existe restrição quanto à latência para se efetivar o início da execução de alguma atividade no sistema?			
Existem instantes específicos de início/fim para execução de atividades do sistema?			
Foi especificado algum tempo de pior caso para a execução de atividades do sistema? (ou ao menos existe preocupação com relação a esta propriedade?)			

Figura 4.4: *Checklist* para Requisitos Temporais - Temporização

	Relevante	Prioridade	Restrições Condições Descrição
Tempo			
Precisão			
Existem atividades com flexibilidade no atendimento de seus requisitos temporais?			
Caso exista flexibilidade no atendimento de requisitos temporais, o sistema suporta retardo em alguma atividade temporizada?			
O sistema suporta variações no atendimento de requisitos temporais?			
Em uma situação de uso degradado do sistema, existe a possibilidade de se utilizar dados antigos?			
Existe a necessidade de algum tipo de controle sobre a validade dos dados utilizados em algum processo do sistema?			
Existe limite quanto à diferença entre o tempo lógico utilizado pelo sistema e o tempo físico gerado por componentes do sistema?			

Figura 4.5: *Checklist* para Requisitos Temporais - Precisão

Os requisitos de desempenho mereceram destaque na classificação de requisitos apresentada na seção 4.1, apesar de envolverem questões ligadas tanto ao tempo quanto à distribuição, e, portanto também devem ser tratados em um *checklist* específico. A Figura 4.6 apresenta este artefato que se encontra dividido em duas *sub-checklists.*

Desempenho	Relevante	Prioridade	Restrições Condições Descrição
Vazão			
Existe limite quanto ao número de atividades que o sistema pode executar?			
Existe alguma restrição quanto à captura ou armazenamento de dados?			
Existem pontos de convergência de dados?			
Existem restrições importantes quanto à utilização da banda no sistema?			
Tempo de Resposta			
Existem limitações temporais para o retorno de respostas finais do sistema?			
Em caso de desempenho degradado (sobrecarga do sistema), existem respostas finais do sistema que podem ser penalizadas em detrimento de outras?			
Existe a possibilidade de se distribuir tarefas para garantir o tempo de resposta de uma atividade do sistema?			

Figura 4.6: *Checklist* para Requisitos de Desempenho

A *checklist* para tratar requisitos ligados à distribuição do sistema apresenta quatro *sub-checklists* dedicadas aos conjuntos de requisitos especificados na classificação proposta neste trabalho. Sua organização pode ser observada na Figura 4.7.

Distribuição	**Relevante**	**Prioridade**	**Restrições Condições Descrição**
Alocação de Tarefas			
Existem critérios espaciais que determinem a distribuição das tarefas do sistema?			
Existindo pontos de convergência de dados, há possibilidade de redistribuição do tratamento destes dados?			
Estações Participantes			
As Estações Participantes tem processamento dedicado a uma determinada atividade?			
Em caso de sobrecarga do sistema, alguma estação pode substituir ou auxiliar no processamento de outra?			
Existe restrição quanto à capacidade de processamento das Estações Participantes?			
Existe restrição espacial que impeça a instalação de alguma estação em algum ponto de atuação ou leitura do sistema?			
Comunicação			
Existe restrição quanto ao uso de alguma tecnologia de comunicação?			
A comunicação exige garantia de entrega de mensagem?			
Existe restrição quanto ao tamanho dos dados transmitidos ou recebidos?			
Existe diferença (tamanho, prioridade) entre tráfego de controle e de dados?			
Sincronização			
Existem recursos ou dados compartilhados?			
É necessário controle de oncorrência sobre os dados compartilhados?			
Existe alguma política pré-estabelecida para acesso de recursos ou dados compartilhados?			
Existe hierarquia quanto ao acesso aos recursos ou dados compartilhados?			

Figura 4.7: Checklist para Requisitos de Distribuição

Os requisitos não-funcionais ligados à característica de o sistema ser embarcado apresentam grande dificuldade de serem levantadas e expressas nas fases inicias do desenvolvimento do sistema. A *checklist* preparada para estes requisitos busca tornar mais clara as necessidades do sistema com relação aos quatro grupos de requisitos elencados na subseção 4.1.4. A *checklist* apresentada na Figura 4.8 expressa o tratamento de análise prestado a tais requisitos.

Embarcados	Relevante	Prioridade	Restrições Condições Descrição
Área			
Existe restrição quanto à área (em silício ou placa de circuito integrado) ocupada por algum componente do sistema?			
Consumo de Potência			
Existe restrição quanto ao consumo de potência de algum elemento do sistema?			
Quanto ao consumo do sistema, existe necessidade de se inserir monitoramento ou controle?			
Energia total			
Existe restrição quanto à energia disponibilizada ao sistema?			
Existe estimativa de quanto deve durar a energia disponibilizada ao sistema?			
Existem fontes alternativas de energia no sistema?			
Existe restrição referente à geração de calor pelo sistema (silhueta térmica)?			
Memória			
Existe restrição quanto ao uso da memória de armazenamento do sistema? (seja para dados ou programas)			
Existe limitação quanto ao uso da memória de execução no sistema?			

Figura 4.8: Checklist para Requisitos de Embarcados

O segundo passo após o preenchimento das *checklists* consiste em verificar se algum requisito não-funcional foi esquecido ou foi mal especificado. Para isto utiliza-se um outro artefato, o processo léxico. Este processo descrito em (LEITE, 1995) consiste em um glossário povoado com termos do contexto específico do domínio do problema, que é utilizado para se identificar a presença de alguma informação que detalhe um requisito já levantado ou apresente um novo. Este glossário é definido sob a forma Backus Naur Form (BACKUS; NAUR, 1969). Esta forma consiste num conjunto de regras que permite uma análise estruturada das sentenças do texto descritivo do sistema e dos *templates* de requisitos funcionais, em busca de conceitos não-funcionais considerados importantes para o projeto, e que não tenham sido devidamente tratados. Assim como para as *checklists*, criou-se um léxico para cada uma das quatro classes de requisitos não-funcionais apresentadas na seção 4.1. Além destes léxicos, existe ainda um quinto que representa a entrada do léxico, este apenas direciona um requisito não-funcional a um dos quatro léxicos específicos. As Figuras 4.9 a 4.13 apresentam os referidos léxicos.

```
<RNF_generico> ::= <tempo> | <desempenho> | <distribuição> | <embarcados>
<tempo> ::= <temporização>  | <precisão>
<desempenho> ::= <tempo_de_resposta>  | <vazão>
<distribuição> ::= <alocação_de_tarefas>  | <estações_participantes> |
                    <comunicação> | <sincronização>
<embarcados> ::= <area>  | <consumo_potencia> | <energia_total> | <memoria>
```

Figura 4.9: Entrada do Léxico

```
<tempo> ::= <temporização>  | <precisão>
<temporização> ::= <deadline>  | <periodo> | <custo> | <wcet> |
          <instante_de_liberacao> |      <latencia_de_ativacao> | <inicio_fim>
          <deadline> ::= a execução deve ser feita até <n> <unidade_tempo>
          <periodo> ::=  a cada <n> <unidade_tempo> | por <unidade_tempo> |
                    periodicamente | ciclicamente
          <custo> ::=  consome <n> <unidade_tempo>
          <wcet> ::=  no pior caso o tempo de execução deve ser de <n>
          <unidade_tempo>
          <instante_de_liberacao> ::= a atividade deve estar pronta para executar
                              em <n> <unidade_tempo>
          <latencia_de_ativacao> ::= após a liberação a atividade deve ser iniciada
                              em <n> <unidade_tempo>
          <inicio_fim> ::= atividade inicia em <n> <unidade_tempo> |  termina em
                    <n> <unidade_tempo>
<precisao> ::= <jitter>  | <retardo_admitido> | <rigidez> | <utilidade> |
          <resolucao>| <exatidao>
          <jitter> ::= variação menor que <n> <unidade_tempo>
          <retardo_admitido> ::=  tempo excedente admitido para a execução de
                              uma atividade é <n> <unidade_tempo>
          <rigidez> ::=  hard | medium | soft
          <utilidade> ::= válido até <n> <unidade_tempo> | válido por <n>
                    <unidade_tempo>
          <resolucao> ::=  registra até <unidade_tempo>
          <exatidao> ::=  desvio de  <n> <unidade_tempo>
          <unidade_tempo> ::= h | min | s | ms | µs | ns | hora | minuto | segundo |
                    milisegundo | microsegundo | nanosegundo | dia | semana | mês | ano
          <n> ::= <n> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |,
```

Figura 4.10: Léxico para o Contexto Tempo

```
<desempenho> ::= <tempo_de_resposta>  | <vazão>
<tempo_de_resposta> ::= o tempo para o sistema retornar uma resposta final
                    é de <n> <unidade_tempo>
<vazão> ::=  chamadas por <unidade_tempo> | execuções por
                              <unidade_tempo>
<unidade_tempo> ::= h | min | s | ms | µs | ns | hora | minuto | segundo |
          milisegundo | microsegundo | nanosegundo | dia | semana | mês | ano
<n> ::= <n> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |,
```

Figura 4.11: Léxico para o Contexto Desempenho

<distribuição> ::= <alocação_de_tarefas> | <estações_participantes> |
<comunicação> | <sincronização>
<alocação_de_tarefas> ::= <n> tarefas devem ser alocadas em determinada
estação | determinada tarefa deve ser alocada em uma estação específico
<estações_participantes> ::= a estação suporta executar <n> tarefas
simultaneamente | a estação deve operar com ocupação de <n>% de sua
capacidade de processamento
<comunicação> ::= com garantia de entrega | sem garantia de entrega | deve usar
<tecnologia_comunicacao> | taxa de <n> <unidade_dados> por <unidade_tempo>
<sincronização> ::= <n> processo(s) pode(m) ler um dado ao mesmo tempo | um
recurso pode ser compartilhado por <n> processo(s)
<tecnologia_comunicacao> ::= CAN | TT-CAN | FTT-CAN | CAN-Aero | Ethernet |
FieldBus | ProfiBus | IEEE802.11
<unidade_dados> ::= bit | byte | Kbit | Kbyte | Megabit | Megabyte | Gigabit |
Gigabyte | Terabyte
<unidade_tempo> ::= h | min | s | ms | µs | ns | hora | minuto | segundo | milisegundo
| microsegundo | nanosegundo | dia | semana | mês | ano
<n> ::= <n> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |,

Figura 4.12: Léxico para o Contexto Distribuição

<embarcados> ::= <area> | <consumo_potencia> | <energia_total> | <memoria>
<area> ::= <n> de células lógicas | <n> <unidade_comprimento>²
<consumo_potencia> ::= <n> Watts | <n> Joules por operação
<energia_total> ::= <n> Joules | <n> Joules por componente | <autonomia >
<autonomia> ::= <n> de operações | <n> de operações por
<unidade_tempo> | <n> <unidade_comprimento> percorrida |
<n> <unidade_tempo>
<memoria>::= <n> <unidade_dados> de armazenamento | <n>
<unidade_dados> de memória física | <n> <unidade_dados> de
memória volátil | ocupa <n> <unidade_dados>
<unidade_comprimento> ::= km | m | dm | cm | mm | kilometro | metro |
decímetro | centímetro | milímetro
<n> ::= <n> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |,
<unidade_tempo> ::= h | min | s | ms | µs | ns | hora | minuto | segundo |
milisegundo | microsegundo | nanosegundo | dia | semana |
mês | ano
<unidade_dados> ::= bit | byte | Kbit | Kbyte | Megabit | Megabyte | Gigabit |
Gigabyte | Terabyte

Figura 4.13: Léxico para o Contexto Embarcado

Com as informações levantadas pelas *checklists* e léxicos, é possível preencher-se um *template* para cada requisito não-funcional identificado. As informações até então levantadas vão sendo dispostas de forma estruturada no *template*, sendo então especificadas em maior detalhe. Durante o preenchimento do *template* deve-se refinar a informação sobre a prioridade do requisito. Esta informação, que foi levantada em alto nível (identificada como alta, média ou baixa) num dos campos de uma *checklist*, será

então refinada para um dos seguintes valores: sem prioridade, baixa, média, alta ou crítica. A Figura 4.14 apresenta a organização do *template* utilizado para a especificação de requisitos não-funcionais.

	Item	Descrição
Identificação	**Identificador**	Identificação que permitirá a rastreabilidade do requisito por todo o projeto.
	Nome	Nome do requisito não-funcional
	Autor	Pessoa responsável pela identificação e definição.
Especificação	**Classificação**	Classificação a qual pertence o entrelaçamento.
	Descrição	Descrição de como afeta funcionalidades do sistema.
	Casos de Uso Afetados	Lista de casos de uso afetados.
	Contexto	Determina em que momento o entrelaçamento afeta um caso de uso.
	Escopo	(Global/Parcial) O requisito é global se afeta o sistema como um todo, e parcial se afeta apenas uma parte do sistema.
Decisão e Evolução	**Prioridade**	Usada para decidir a importância relativa entre os casos de uso. Ela é assume um dos seguintes valores: sem prioridade, baixa, média, alta ou crítica.
	Situação	Um requisito pode estar em uma das seguintes situações: 0 - identificado; 1 - analisado; 2 - especificado; 3 - aprovado; 4 - cancelado; 5- finalizado;

Figura 4.14: *Template* para Requisitos Não-Funcionais

Preenchidos os *templates* para os requisitos não-funcionais, passa-se a resolução de possíveis conflitos entre requisitos desta natureza. Para se realizar esta tarefa, utiliza-se a mesma técnica descrita para a resolução de conflitos entre requisitos funcionais. Monta-se uma tabela com todos os requisitos dispostos na primeira linha e coluna e assinala-se a quadrícula que relaciona os requisitos em conflito (de acordo com as regras de identificação de conflitos descritas no Anexo A). Um exemplo da tabela de resolução de conflitos pode ser observada no estudo de caso apresentado na Seção 5.1, através da Tabela 5.1. Baseando-se nas informações de prioridades refinadas nos *templates*, decide-se sobre o atendimento ou não dos requisitos conflitantes. Esta decisão pode alterar as prioridades dos requisitos envolvidos no conflito (isto pode ocorrer quando dois requisitos têm a mesma prioridade), ou mesmo cancelar o tratamento de um deles. Nos casos em que os requisitos apresentem prioridades distintas, a sua ordem é obedecida, caso eles tenham a mesma prioridade, tenta-se buscar alguma informação adicional nos *checklists* que ajude a solucionar o impasse. Caso isto não seja possível, os *stakeholders* devem ser consultados.

Resolvidos os conflitos entre requisitos não-funcionais, atualiza-se o *template* de todos os requisitos envolvidos em conflitos. Esta atualização mantém a primeira versão da especificação e adiciona uma nova coluna com o conteúdo da nova especificação no requisito que por ventura tenha sido mantido, porém alterado. Os requisitos que após a

resolução de conflitos forem finalizados e mantidos serão considerados aspectos candidatos na fase seguinte.

O último passo corresponde ao povoamento do diagrama de casos de uso com os requisitos não-funcionais levantados. Como não existe um padrão para a representação gráfica destes requisitos no diagrama de casos de uso, optou-se por uma notação similar à sugerida em (ARAÚJO et al, 2002), que é uma das propostas consideradas em (JACOBSON; NG, 2004). Com isto, utiliza-se uma elipse com o estereótipo <<non-functional>> para representar o requisito não-funcional e um relacionamento ligando o requisito não-funcional ao caso de uso afetado. Este relacionamento é modificado com o estereótipo <<crosscut>> e navegável do requisito não-funcional para o caso de uso que representa o requisito funcional afetado. A Figura 4.15 apresenta a notação. Deve-se ressaltar que por questão de clareza do diagrama, pode-se optar por representar cada requisito não-funcional específico (como por exemplo, *jitter* e prazo de validade), ou o macro requisito que engloba os requisitos específicos em questão (como o de precisão que engloba os dois anteriormente apresentados como exemplo).

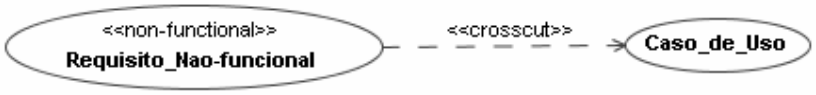

Figura 4.15: Notação para representação de requisitos não-funcionais em diagrama de casos de uso

4.2.2 Segunda Fase: Mapeamento de Requisitos em Elementos de Projeto

Nesta fase será realizado o mapeamento dos requisitos levantados e especificados na primeira fase em elementos de projeto que serão detalhados na fase de projeto. Para isto, três passos devem ser executados:

a) Extrair do diagrama de casos de uso e dos *templates* de requisitos funcionais os conceitos e atributos que serão responsáveis por compor a parte funcional do sistema. Esta informação servirá de subsídio para a definição das classes responsáveis pelo tratamento de cada requisito funcional;
b) Realizar a extração de aspectos, que consiste na análise dos aspectos candidatos e decisão de que aspectos serão utilizados na fase de projeto para tratar cada requisito não-funcional levantado;
c) Composição da informação levantada nos passos anteriores em uma tabela de mapeamento que relaciona os requisitos aos elementos de projeto (classes e aspectos) especificados para tratá-los. Esta tabela tem grande importância na manutenção da rastreabilidade entre requisitos e elementos de projeto e vice-versa. Além disto, as células de encontro entre requisitos funcionais e não-funcionais servem para explicitar que requisitos não-funcionais afetam determinado requisito funcional, bem como que aspectos afetam determinada classe. O modelo de organização da tabela de mapeamento é apresentado na Figura 4.16.

Durante a extração de aspectos (passo b), o projetista tem a sua disposição uma biblioteca extensível de aspectos descritos em alto nível (DERAF High-level Aspects) para realizar o tratamento de requisitos não-funcionais referentes a sistemas tempo-real distribuídos e embarcados. Caso algum requisito levantado não encontre tratamento adequado em nenhum aspecto presente na biblioteca, deve-se estender o seu conteúdo

através da definição de um novo aspecto ou através da inclusão de algum tratamento específico (comportamental ou estrutural) a algum dos aspectos já especificados. Outros aspectos não afetos especificamente ao domínio TrED podem ser especificados e utilizados conforme a necessidade do projeto.

Caso ocorra a inclusão de algum novo componente durante a fase de projeto, deve-se atualizar a tabela de modo que ela expresse com precisão a que requisito o novo componente está associado.

		Requisitos Não-Funcionais				
		Identificador do Requisito Não-Funcional (NRF 1)	NFR 2	...	NFR n	**Classe(s) Responsável(eis) pelo tratamento do Requisito Funcional**
Requisitos Funcionais	Identificador do Requisito Funcional (FR 1)	A quadrícula que apresenta o encontro de um RF afetado pelo RNF da coluna é marcada.				Nome da(s) Classe(s) Responsável(eis) pelo tratamento do RF
	FR 2					Classe 1, ..., classe j.
	...					...
	FR n					Classe k, ..., classe n.

Aspecto(s) Responsável(eis) pelo tratamento do Requisito Não-Funcional	Nome do(s) Aspecto(s) Responsável(eis) pelo tratamento do RNF	Aspect 1, ..., Aspecto j	...	Aspect k, ..., Aspecto n

Figura 4.16: Tabela de Mapeamento de Requisitos em Elementos de Projeto

4.2.3 Terceira fase: Projeto do Sistema

O projeto do sistema pode ser realizado com as informações colhidas e organizadas nas fases anteriores. Para isto, utiliza-se primeiramente a tabela de mapeamento construída na segunda fase para povoar o diagrama de classes. Constrói-se então a hierarquia de classes do sistema. As classes são estereotipadas com elementos do perfil UML de escalonabilidade, desempenho e tempo – RT-UML (OMG, 2004), conforme representem recursos escalonáveis, objetos compartilhados ou qualquer outro elemento de destaque tratado pelo perfil (mais detalhes sobre elementos do perfil na subseção 2.2.1).

Em seguida, constrói-se um novo diagrama composto apenas pelas classes afetadas por requisitos não-funcionais e os respectivos aspectos que as afetam. Este diagrama, chamado ACOD (Aspect Crosscutting Overview Diagram), apresenta uma visão do entrelaçamento de aspectos em classes do sistema, através de um relacionamento que liga os aspectos com as classes por eles afetadas. A notação utilizada para representar aspectos, seus elementos internos e a associação entre aspectos e classes, segue a sugestão apresentada em (STEIN et al, 2002). Uma vez que não existe uma notação padronizada, procurou-se um trabalho reconhecido pela comunidade internacional que estuda a utilização de orientação a aspectos, através de pesquisa no *Survey of Analyses*

and Design Approaches publicado pela AOSD-Europe (AOSD-Europe, 2005). Seguindo o objetivo de utilizar uma notação que seja compatível com as abordagens tradicionais, de modo a facilitar a assimilação do uso da metodologia, escolheu-se seguir a sugestão de Stein. Outro motivo que levou a escolha desta notação foi a facilidade de se proporcionar suporte à rastreabilidade entre o projeto e a implementação em linguagem Java/AspectJ, uma vez que é possível um mapeamento direto entre os elementos internos de um aspecto adotado neste trabalho (*StructuralAdaptation* e *BehavioralAdaptation*), segundo o meta modelo apresentado em (SCHAUERHUBER et al, 2006), e os elementos internos de um aspecto em AspectJ (respectivamente *introduction* e *advices*). Esta característica permite a integração da metodologia a um *Framework* que possua bibliotecas de componentes escritos em Java, como o do projeto SEEP.

A representação dos aspectos é feita através de uma classe com o estereótipo <<aspect>>. Nele estão declarados todos os elementos necessários para a inserção da dimensão não-funcional nas classes que representam a parte funcional do sistema. Estes elementos que fazem parte do aspecto são: o *StructuralAdaptation*, que representam uma abstração do *introduction* encontrado em AspectJ, *BehavioralAdaptation*, que representa uma abstração do *advice* de AspectJ, e os *pointcuts*, que determinam os *joinpoints* a serem interceptados. A Figura 4.17 apresenta um aspecto, com seus componentes, e seu relacionamento com um elemento base do sistema (uma classe).

A determinação de onde serão inseridas as adaptações, sejam elas estruturais (*StructuralAdaptation*) ou comportamentais (*BehavioralAdaptations*), fica a cargo do elemento chamado *pointcut*. Eles são identificados nos aspectos através do estereótipo <<pointcut>>, e tem como parâmetros o *joinpoint* e a respectiva adaptação a ser inserida. O parâmetro *joinpoint* é representado por um diagrama chamado JPPD, que será apresentado em seguida. No caso de adaptações comportamentais, além dos dois parâmetros mencionados, o *pointcut* ainda possui um terceiro que identifica a posição na qual deve ser inserida a adaptação (*before*, *after* ou *around*), de acordo com o local onde será inserido o comportamento não-funcional, antes, depois ou em substituição ao comportamento capturado. A opção de se indicar a posição da adaptação no *pointcut* e não na adaptação comportamental (*advice* em AspectJ), se dá pelo fato de se proporcionar um maior grau de reuso das adaptações, uma vez que pode haver a necessidade de se utilizar tais adaptações em posições diferentes em projetos distintos (SCHAUERHUBER et al, 2006). Vale ressaltar que tal escolha em nada invalida o mapeamento do par (*pointcut*, *BehavioralAdaptation*), do modelo em alto nível proposto, em (*pointcut*, *advice*) da linguagem AspectJ, uma vez que as informações necessárias à codificação de um entrelaçamento comportamental em AspectJ estão presentes no par que especifica o entrelaçamento comportamental no modelo de alto nível.

Os *StructuralAdaptation* são responsáveis pelas mudanças estáticas nas classes-base do sistema, sejam elas introduções de atributos, ou operações (estas podem ser operações como métodos *get* ou *set*, mas também podem ser métodos construtores). A declaração de um *StructuralAdaptation* consiste basicamente do nome do elemento estrutural a ser introduzido, podendo ser completado ainda com seu tipo (no caso de um atributo) ou tipo de retorno (no caso de um método). A notação adotada ressalta este elemento com o estereotipo <<StructuralAdaptation>>.

Já o tratamento dinâmico fica a cargo dos *BehavioralAdaptations*. Este tipo de tratamento não-funcional representado pelo *BehavioralAdaptation* é identificado pelo

estereótipo <<BehavioralAdaptation>>. A sua declaração consiste basicamente do nome da adaptação a ser inserida estereotipada conforme mencionado.

A conexão entre os aspectos e as classes é feita através de um relacionamento estereotipado <<crosscut>>, que pode ainda apresentar alguma informação adicional sobre como o aspecto está afetando uma determinada classe, como por exemplo, o valor de algum novo atributo inserido pelo aspecto. Este relacionamento é navegável no sentido do aspecto para a classe.

Deve-se lembrar que além dos elementos apresentados, um aspecto pode naturalmente apresentar atributos e métodos para sua utilização interna. Estes elementos são representados de maneira convencional como os atributos e métodos de uma classe.

Uma observação importante a ser feita é que assim como um diagrama de classes, um ACOD pode apresentar as informações internas dos aspectos (adaptações estruturais e comportamentais, *pointcuts*, métodos e atributos) visíveis ou não, dependendo do nível de detalhe que se deseja observar.

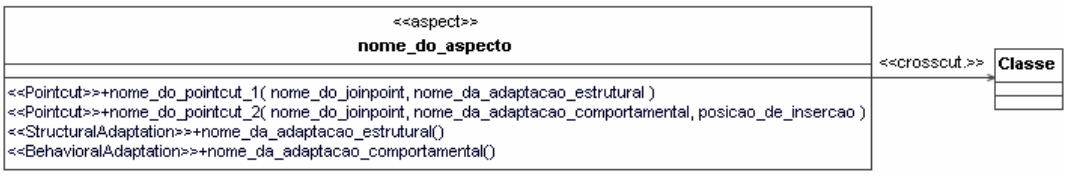

Figura 4.17: Notação para representação de aspectos em modelos UML

Para completar o modelo do sistema, deve-se representar graficamente os pontos onde serão inseridos os tratamentos não-funcionais especificados pelos aspectos (*joinpoints*). Para se realizar esta especificação, a proposta apresentada nesta dissertação também segue o trabalho de Stein, que por sua vez apresenta um conjunto de diagramas para representar *joinpoints* em diversas situações (STEIN et al, 2006). Os modelos de representação de *joinpoints* apresentados por Stein (JPDD – JoinPoint Designation Diagram) permitem a descrição destes pontos em fluxos de controle, fluxos de dados e mudanças de estado. A proposta desta dissertação apresenta especial interesse na descrição de fluxos de controle, de tal forma que apenas esta parte do referido trabalho será utilizada. Porém, como na proposta de Stein aparecem elementos não padronizados pela UML, será adotada sua descrição de *joinpoints* de fluxo de controle com pequenas adaptações. Estas adaptações têm por objetivo adequar o diagrama de designação de *joinpoints* (JPDD) de modo a utilizar apenas elementos presentes no padrão UML. No entanto, deve-se ressaltar que a adaptação do referido diagrama não muda a semântica apresentada pela proposta original de Stein. O diagrama de representação de *joinpoints* adaptado é apresentado na Figura 4.18. Para realizar a especificação dos *joinpoints*, o desenvolvedor deve utilizar informações provenientes dos campos “contexto” do *template* do respectivo requisito não-funcional analisado, mais as informações colhidas nos campos “caminhos” e “cenários” dos requisitos funcionais afetados.

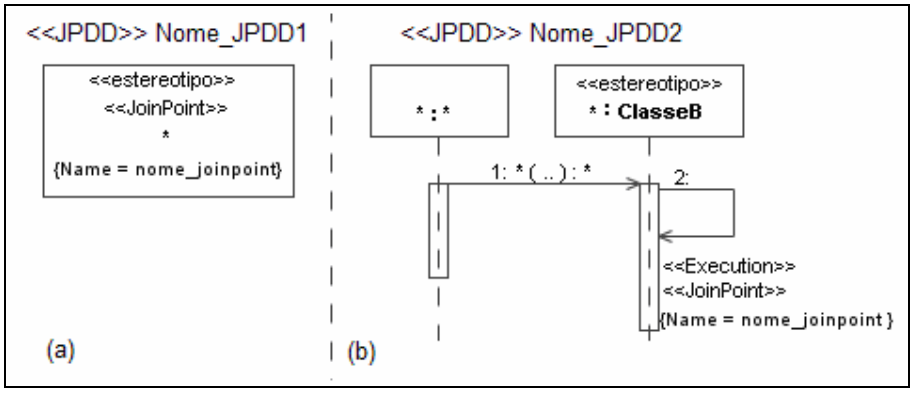

Figura 4.18: Diagrama de Representação de Joinpoints (JPDD) adaptado: (a) seleção de classe (b) seleção de método

Na Figura 4.18(a), pode-se observar o JPDD cujo nome é "Nome_JPDD1", que representa um *joinpoint* que captura qualquer classe (representado pelo asterisco "*") anotada com o estereótipo <<estereótipo>>. Este tipo de JPDD é usado em *pointcuts* que determinam adaptações estruturais. Já na Figura 4.18(b), observa-se o JPDD de nome "Nome_JPDD2", que captura a execução de qualquer método com quaisquer parâmetros e com qualquer retorno (representado por " *(..):* ") de qualquer objeto da classe de nome "ClasseB", chamado por qualquer objeto de qualquer classe.

Adicionalmente, pode-se utilizar um diagrama de seqüência para se observar a interação das classes e aspectos do sistema, como apresentado em (STEIN et al, 2002). Na verdade esta é uma representação primitiva do que o mesmo autor mais tarde propôs através dos JPDDs, porém a título de propor um maior detalhamento e talvez enriquecer a documentação, o uso de diagramas de seqüência com este propósito pode ser feito, mesmo não sendo indispensável. Uma outra utilidade do diagrama de seqüências seria seu uso convencional para demonstrar a interação entre instâncias de classes do sistema.

4.3 Biblioteca Extensível de Aspectos de Alto Nível para Sistemas TrED

Durante a utilização da RT-FRIDA foi mencionado o uso de aspectos definidos em uma biblioteca de aspectos extensível de alto nível para tratar os requisitos não-funcionais levantados do sistema. Esta biblioteca faz parte de um *framework* de aspectos chamado DERAF (Distributed Embedded Real-time Aspects Framework). Este *framework*, integrado ao do projeto SEEP, provê suporte ao uso de aspectos nas fases de projeto e implementação (WEHRMEISTER, 2006). Esta seção apresenta a biblioteca de alto nível utilizada na fase de projeto.

A biblioteca é dividida em pacotes de acordo com a sua aplicação ao tratamento específico de requisitos não-funcionais relativos à: tempo, distribuição e embarcado. A Figura 4.19 apresenta uma visão geral da biblioteca.

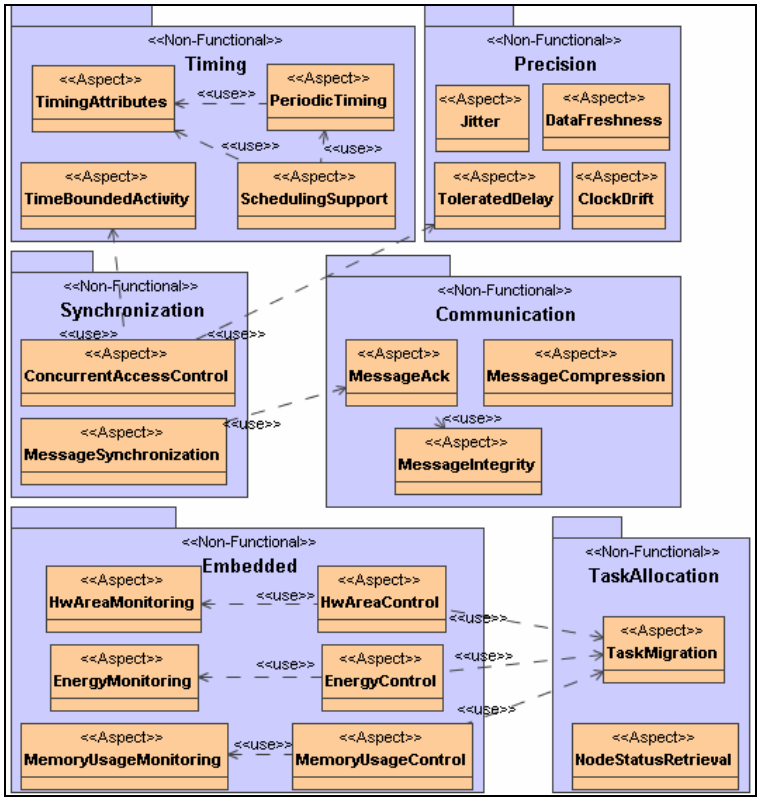

Figura 4.19: Biblioteca de Extensível de Aspectos de Alto Nível para Sistemas TrED

Cada um dos pacotes contém aspectos que tratam requisitos bem específicos de modo que um requisito não-funcional levantado na fase de análise pode vir a ser tratado por mais de um aspecto da biblioteca. Alguns aspectos utilizam comportamentos de outros, o que é representado pela relação <<use>> que pode ser observada na Figura 4.19.

A seguir será apresentada uma descrição de cada aspecto previsto na biblioteca.

a) Pacote “Timing”: reúne os aspectos que tratam os requisitos de temporização do sistema. A Figura 4.20 apresenta os aspectos componentes deste pacote com maiores detalhes.

a.1) TimingAtributes: introduz os atributos temporais aos objetos ativos do sistema (deadline, prioridade, WCET, tempo de início/fim, dentre outros). Ele também é responsável pela inicialização dos atributos que introduz.

a.2) PeriodicTiming: adiciona o mecanismo de ativação periódica em objetos ativos. Para isto é preciso introduzir um novo atributo temporal (período), além do mecanismo de controle de execução periódica.

a.3) SchedulingSupport: insere o mecanismo de escalonamento no sistema que dá suporte a execução de objetos ativos. Além disto, este aspecto é responsável pela inclusão de objetos ativos à lista de escalonamento, e ainda de realizar o teste de verificação da escalonabilidade da lista.

a.4) TimeBoundedActivity: limita temporalmente a execução de uma atividade. Para isto, inclui um mecanismo que restringe o tempo máximo de execução de uma atividade (aquisição de um “lock” por um recurso compartilhado, por exemplo). O tempo começa

a ser contado imediatamente antes do início da atividade, caso o tempo máximo seja atingido, deve prover uma maneira de interromper a execução da atividade.

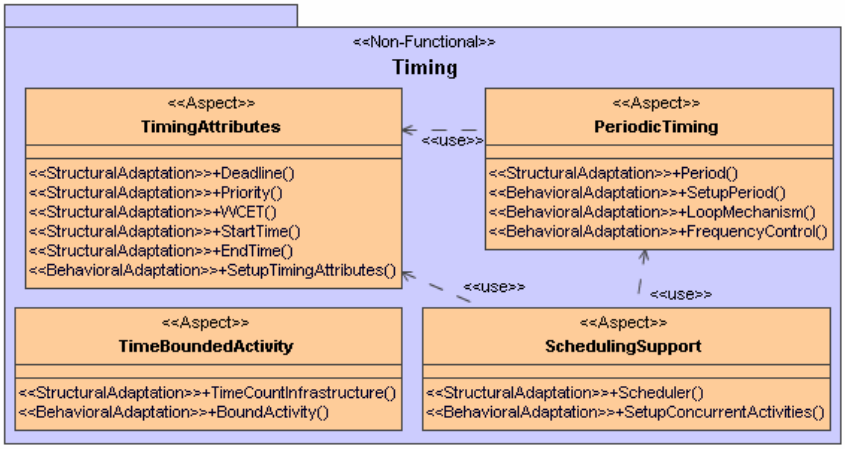

Figura 4.20: Pacote “Timing”

b) Pacote “Precision”: reúne os aspectos que tratam requisitos de precisão no atendimento de requisitos temporais. Uma visão mais detalhada do pacote é apresentada na Figura 4.21.

b.1) Jitter: mede o tempo de início e fim de execução de uma determinada atividade do sistema. Com base nestas medidas, calcula a variação e caso o limite tolerado seja atingido, executa medidas corretivas.

b.2) ToleratedDelay: limita temporalmente o início da execução de uma atividade (por exemplo o tempo máximo de espera para a aquisição de um “lock” de um recurso compartilhado). Ele adiciona um mecanismo de contagem de tempo que é iniciado imediatamente antes do início da execução da atividade e caso o tempo máximo tolerado seja atingido, executa ações corretivas.

b.3) DataFreshness: associa *timestamps* a dados a fim de verificar a sua validade antes de utilizá-los. Depois que um dado com validade controlada for escrito, seu *timestamp* deve ser atualizado. De forma análoga, a validade de um dado controlado deve ser verificada antes de se realizar uma leitura para utilização de seu valor. Caso a validade do dado tenha expirado, ações corretivas devem ser adotadas.

b.4) ClockDrift: mede o tempo no qual uma atividade inicia e compara com o tempo esperado para seu início. Se a diferença acumulada exceder um valor máximo tolerado, uma medida corretiva é adotada.

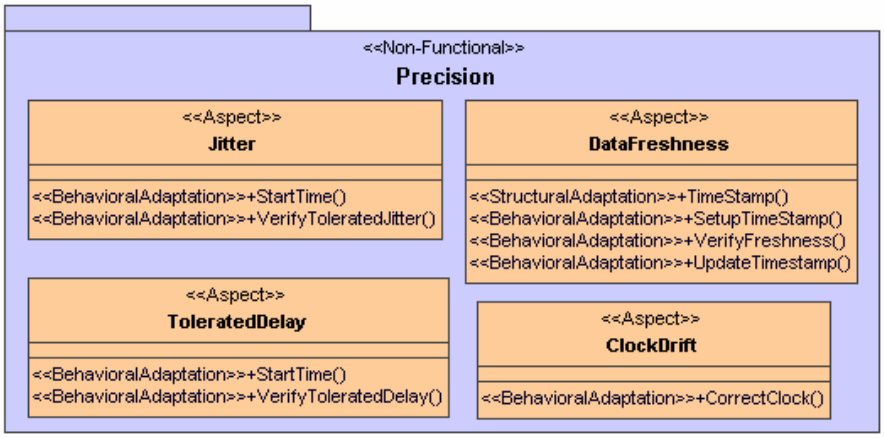

Figura 4.21: Pacote “Precision”

c) Pacote “Synchronization”: este pacote contém os aspectos que tratam os requisitos de sincronização e acesso concorrente do sistema. Maiores detalhes dos aspectos integrantes deste pacote podem ser observados na Figura 4.22.

c.1) ConcurrentAccessControl: adiciona um mecanismo de controle de concorrência para o acesso de recursos compartilhados. Toda vez que um objeto ativo precisar acessar um recurso compartilhado ele pede um “lock” ao mecanismo de controle, e após utilizá-lo, o objeto ativo deve notificar o mecanismo para que o “lock” seja liberado.

c.2) MessageSynchronization: adiciona um mecanismo de espera que retém a execução de uma atividade até a chegada de uma confirmação da mensagem enviada.

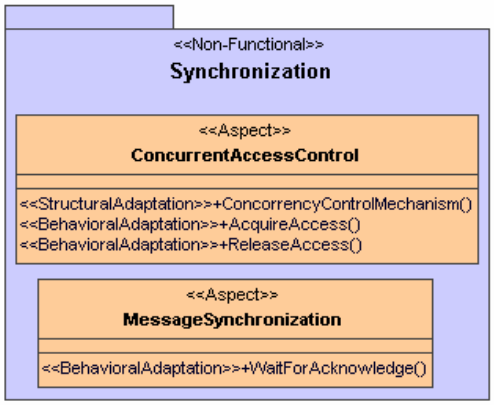

Figura 4.22: Pacote “Synchronization”

d) Pacote “Communication”: reúne os aspectos que tratam requisitos que afetam a comunicação entre nodos do sistema. Os detalhes dos aspectos que se encontram neste pacote podem ser observados na Figura 4.23.

d.1) MessageAck: adiciona um mecanismo de garantia de entrega de mensagem. No lado do emissor, após o envio da mensagem o mecanismo deve ser notificado que uma mensagem de confirmação deve chegar. No lado do receptor, após a entrega de uma mensagem, uma mensagem de confirmação deve ser enviada ao emissor.

d.2) MessageIntegrity: verifica a integridade de uma mensagem. No lado do emissor, antes e se enviar uma mensagem, deve-se calcular um código de checagem (e.g. CRC ou paridade), e anexar esta informação à mensagem. No lado do receptor,

após o recebimento de uma mensagem, um algoritmo deve gerar um código de checagem e compará-lo ao código anexado à mensagem.

d.3) MessageCompression: adiciona um mecanismo de compressão para otimizar a utilização do canal de comunicação. No lado do emissor, uma mensagem deve ser comprimida antes de ser enviada, enquanto no lado do receptor ela deve ser expandida antes de sua entrega.

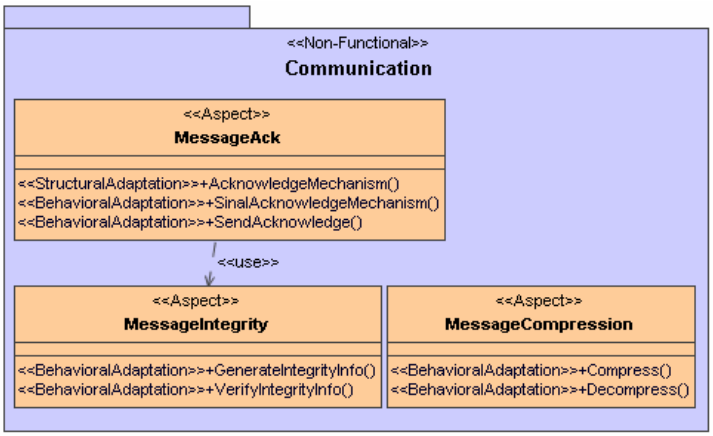

Figura 4.23: Pacote “Communication”

e) Pacote “TaskAllocation”: este pacote contém os aspectos responsáveis pelo tratamento dos requisitos ligados à distribuição das tarefas pelos diversos nodos do sistema. A Figura 4.24 apresenta maiores detalhes deste pacote.

e.1) TaskMigration: fornece um mecanismo que permite a migração de tarefas de um nodo a outro do sistema ou de software para hardware, ou o contrário. Ele é utilizado pelos aspectos de controle definidos no pacote Embedded (EnergyControl, Memorycontrol e HwAreaControl), que são responsáveis pela decisão sobre a migração.

e.2) NodeStatusRetrieval: insere um mecanismo capaz de prover informações sobre a carga de processamento, taxas de recebimento/envio de mensagem, e disponibilidade do nodo (mensagem: *I’m alive*). Antes ou depois do início ou do fim da execução de um objeto ativo a carga de processamento deve ser calculada. Antes ou depois do envio/recebimento de uma mensagem a taxa de comunicação deve ser calculada. A mensagem de disponibilidade do nodo pode ser enviada periodicamente ou após um número arbitrário de mensagens enviadas na rede.

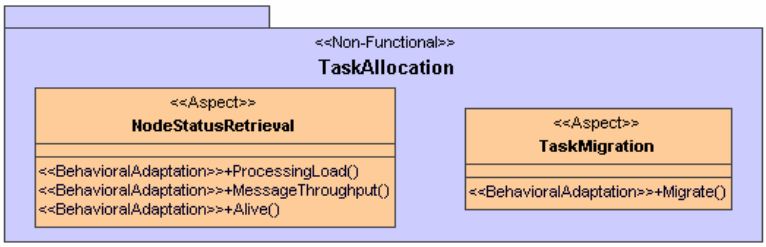

Figura 4.24: Pacote TaskAllocation

f) Pacote “Embedded”: os aspectos que tratam os requisitos associados ao fato do sistema ser embarcado se encontram neste pacote. A Figura 4.25 apresenta os aspectos deste pacote com maiores detalhes.

f.1) EnergyMonitoring: insere um mecanismo de monitoramento de energia capaz de medir o consumo de uma atividade. Antes do início e depois do fim da execução de uma atividade, o nível de energia é medido. A diferença entre as medidas é calculada e armazenada.

f.2) EnergyControl: adiciona um mecanismo que implementa uma política de controle de uso de energia que realiza ações de acordo com o nível restante de energia. As ações podem ser: a migração de tarefas; a desativação de tarefas menos prioritárias; diminuição da freqüência do sistema; dentre outras.

f.3) MemoryUsageMonitoring: insere um mecanismo de monitoramento do total de memória utilizada pelo sistema. Antes da alocação de memória a quantidade de memória demanda é somada ao total utilizado pelo sistema, assim como após a liberação de memória, a quantidade liberada é subtraída do total utilizado.

f.4) MemoryUsageControl: adiciona mecanismo capaz de realizar o controle do uso de memória segundo uma política determinada, como compressão da memória, migração de tarefas, liberação de memória, dentre outras.

f.5) HwAreaMonitoring: fornece um mecanismo de monitoramento do uso de área em um FPGA. Caso o sistema utilize técnicas de reconfiguração de hardware, e dependendo da política de reconfiguração, a área total utilizada deve ser atualizada antes de cada reconfiguração do sistema.

f.6) HwAreaControl: verifica se a reconfiguração do hardware requerida é possível de ser realizada. Caso isto seja possível, a reconfiguração é permitida.

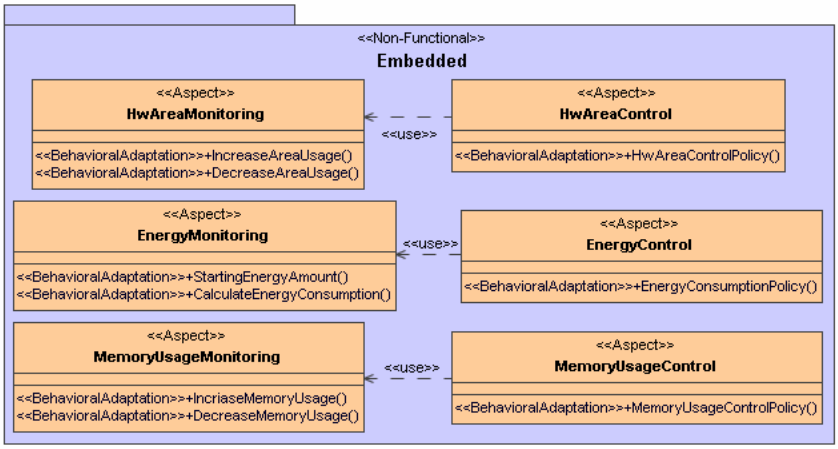

Figura 4.25: Pacote “Embedded”

4.4 Inserção da Metodologia ao *Framework* do Projeto SEEP

Nesta seção serão apresentadas as contribuições do presente trabalho propostas ao *Framework* do projeto SEEP. A Figura 4.26 apresenta o ciclo de projeto do SEEP já com as modificações propostas nesta dissertação bem como aquelas propostas em (WEHRMEISTER, 2006). As contribuições específicas do presente trabalho encontram-se destacadas na referida figura.

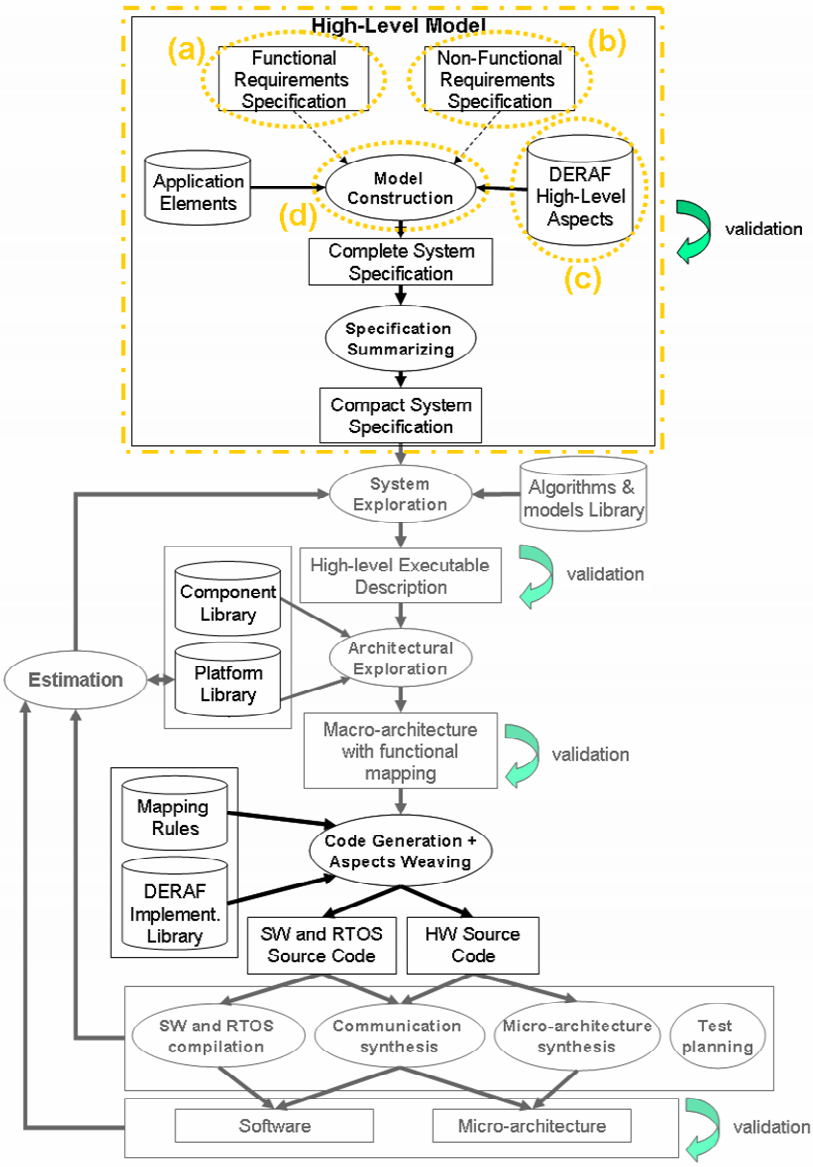

Figura 4.26: Contextualização do trabalho no âmbito do projeto SEEP

A proposta se concentra, portanto, na fase de modelagem de alto nível do sistema. Como destacado na Figura 4.26, a contribuição proposta pode ser encontrada em quatro partes da modelagem de alto nível. A primeira - destacada pela letra (a) – consiste na adoção de *templates* que auxiliam a especificação dos requisitos funcionais do sistema. A utilização deste *template* encontra-se descrita no início da subseção 4.2.1. A incorporação da modelagem não-funcional à especificação de requisitos, que é efetivado pela fase de identificação e especificação de requisitos, também apresentada na subseção 4.2.1, constitui a contribuição destacada pela letra (b) na Figura 4.26. Nela pode-se destacar a utilização de ferramentas de especificação e elicitação presentes na metodologia RT-FRIDA.

A especificação de uma biblioteca de aspectos abstratos (trabalho realizado em conjunto com o doutorando Marco Aurélio Wehrmeister) para o tratamento de requisitos não-funcionais caracteriza a terceira contribuição do trabalho, destacada na

Figura 4.26 pela letra (c). Esta biblioteca, que faz parte de um *framework* orientado a aspectos chamado DERAF, apresenta um conjunto de aspectos abstratos divididos em pacotes que tratam os requisitos não-funcionais específicos de STrED (conforme classificação apresentada em 4.1), como descrito na subseção 4.3. Estes aspectos são então utilizados na construção do modelo do sistema, contribuição destacada na Figura 4.26 pela letra (d). Para a construção do modelo é necessário primeiramente realizar o mapeamento de que elementos de projeto serão responsáveis pelo tratamento dos requisitos do sistema. Este mapeamento, responsável pela rastreabilidade entre requisitos e elementos de projeto, é feito através da utilização de uma tabela de mapeamento, segundo a descrição contida na subseção 4.2.2. Por fim a modelagem utiliza as informações desta tabela para povoar os diagramas do modelo, como descrito na subseção 4.2.3.

Uma vez que o projeto SEEP utiliza componentes escritos em linguagem Java que são executados sobre um processador Java nativo, o FemtoJava (ITO, 2000), buscou-se propiciar o suporte a rastreabilidade entre a especificação funcional e não-funcional do sistema com a sua implementação. Para isto, escolheu-se uma notação para modelagem que fosse facilmente mapeada em elementos da linguagem Java/AspectJ e que ao mesmo tempo pudesse se integrada aos elementos do perfil RT-UML, utilizado no projeto SEEP. Esta notação, baseada nos trabalhos (STEIN et al, 2002), (STEIN et al, 2006) e (SCHAUERHUBER et al, 2006), é apresentada na subseção 4.2.3.

Não houve necessidade de nenhuma mudança ou adequação a nenhuma outra fase do *framework* do projeto SEEP, uma vez que o código a ser entregue a ferramenta SASHIMI (ITO, 2000) para geração do software da aplicação, RTOS e para a síntese da micro-arquitetura é similar àquele utilizado em (WEHRMEISTER, 2005). Para que isto seja possível, existem duas possibilidades: (1) se codifica o sistema com classes e aspectos na linguagem Java/AspectJ, e utiliza-se uma ferramenta de *weaving* (entrelaçamento), i.e. ajc (aspectj compiler), que gerará classes com as inclusões e modificações especificadas nos aspectos, estas classes são então entregues à ferramenta SASHIMI que prosseguirá o processo; (2) utiliza-se uma ferramenta de geração de código a partir de modelos que já faça o entrelaçamento dos aspectos com as informações extraídas diretas do modelo (WEHRMEISTER, 2006).

5 ESTUDOS DE CASO

Este capítulo apresenta dois estudos de caso que demonstram a utilização da metodologia RT-FRIDA. O primeiro consiste no projeto do sistema de controle de movimento de uma cadeira de rodas automatizada e o segundo no projeto do sistema de controle de movimento de um veículo aéreo não-tripulado. Como uma das motivações do uso da metodologia é destacar a viabilização do reuso de componentes, a confecção dos dois estudos de caso utilizam aspectos da biblioteca de aspectos descrita na subseção 4.3, bem como é destacado o reuso de componentes funcionais do projeto da cadeira de rodas no projeto da aeronave não-tripulada.

5.1 Cadeira de Rodas Automatizada

O estudo de caso da automação de uma cadeira de rodas se baseia num projeto que está sendo desenvolvido no Laboratório de Sistemas Embarcados, no âmbito do SEEP (LSE, 2004). Este estudo de caso tem como objetivo a validação da metodologia desenvolvida no projeto, tendo sido alvo de diversos trabalhos. A presente utilização do estudo de caso da automação da cadeira de rodas tem por objetivo apresentar uma comparação entre a modelagem deste mesmo sistema orientado a objetos, apresentado em (WEHRMEISTER, 2005), em contraste com a modelagem orientada a aspectos apresentada nesta dissertação. Nesta seção será apresentado o modelo orientado a aspectos da cadeira enquanto a comparação será alvo de análise no capítulo 6, reservado à apreciação dos resultados.

A automação da cadeira de rodas proposta no projeto busca auxiliar pessoas com necessidades especiais a se locomoverem com segurança, proporcionando o máximo de bem estar além de auxiliar no controle dos sinais vitais do seu usuário. Maiores detalhes sobre a descrição original do sistema podem ser obtidos em (GREEF et al, 2004). No presente trabalho o foco recai sobre a modelagem do controle de movimento da cadeira através da metodologia proposta.

Uma breve descrição do controle de movimento da cadeira de rodas automatizada é apresentada como se segue:

A cadeira possui três modos de operação do controle de movimento, a saber: (1) modo silencioso, que não sinaliza os eventos que ocorrem durante o controle do movimento; (2) modo sinalizar, que oferece ao usuário todas as informações sobre o controle de movimento; (3) modo sinalizar com segurança, que continua sinalizando, porém freia a cadeira em caso de sinalização de um problema grave no controle do movimento. Deve haver um seletor que possibilita o usuário a troca entre os modos de operação. A cadeira deve realizar amostragem de valores referentes ao seu movimento atual (valor da velocidade e ângulo de deslocamento de suas rodas) e posição do *joystick*

(que irá fornecer novos valores de direção e velocidade) em intervalo de tempo que permita uma taxa de atualização de 1000 amostras por segundo. Estes dados devem ser suficientes para que o sistema de controle de movimento da cadeira seja capaz de fornecer novos parâmetros de movimento (novo valor de velocidade e nova angulação para as rodas) na taxa de 500 novos valores por segundo. Estes parâmetros devem ser transmitidos aos atuadores para que estes por sua vez possam ajustar o torque do motor e alterar a angulação das rodas. A execução das atividades do sistema que realizam as leituras de dados e o controle de movimento devem ter tempos de execução 3 e 7 milisegundos, respectivamente. Esta necessidade visa manter uma folga no processador dedicado a cada uma das respectivas funções. Por questões de adequação do funcionamento seguro da cadeira, não se deve admitir a execução de ciclos de processamento atrasado em janelas de tempo reservadas aos novos ciclos de execução. O controle de movimento como uma atividade crítica do sistema, deve possuir um módulo de tratamento de erros, bem como um alarme que informe a ocorrência dos eventuais erros. Estes elementos implementam os três modos de operação da cadeira anteriormente descritos. O alarme deve responder aos comandos de acionamento e desligamento oriundos de demandas do sistema num tempo menor que 2 milisegundos, bem como não deve demorar mais que 2 milisegundos para realizar sua função. Os dados amostrados pelo sistema devem ter uma validade temporal para sua utilização. Extrapolado este tempo de validade (estipulado em 15 milisegundos) os dados devem ser descartados pelo sistema e devem ser tornar inutilizáveis. Por questões de disposição espacial dos sensores, o sistema deve ter processamento distribuído, ficando a cargo de um nodo do sistema a leitura de valores de movimento amostrados e noutro a leitura de valores do *joystick* e o processamento do controle. Deve-se garantir o controle de acesso sobre todos os dados do sistema, seja para leitura ou atualização. A comunicação entre os nodos do sistema deve garantir a integridade dos dados trocados, bem como a sua entrega aos destinatários. Uma questão importante do sistema é a economia de energia. Uma vez que o sistema é alimentado apenas por uma fonte (bateria), deve-se manter um controle sobre a utilização deste recurso, bem como prover meios de reduzir o seu consumo. Outra questão importante, porém mais crítica no sistema, é tempo de resposta de comandos de controle executados pelo usuário. Este tempo de resposta não pode ser superior a 0,5 segundo, caso contrário à interatividade do sistema fica comprometida.

A seguir será apresentado o desenvolvimento do estudo de caso de acordo com as fases de identificação e especificação de requisitos; mapeamento de requisitos; e projeto.

5.1.1 Identificação e Especificação de Requisitos

A descrição completa da cadeira de rodas automatizada é mais complexa do que o apresentado anteriormente, porém é possível perceber pelo extrato apresentado diversas características funcionais e não-funcionais descritas em conjunto, entrelaçando-se. A fase de identificação e especificação de requisitos busca justamente realizar a separação de tais requisitos utilizando para isto ferramentas que possibilitam sua elicitação e estruturação, como descrito na subseção 4.2.1.

Após a leitura da descrição do sistema realiza-se a análise dos requisitos funcionais do mesmo. Tenta-se delimitar o escopo do sistema, buscando entender o que dele se deseja. Realiza-se então o levantamento dos casos de uso do sistema. Estes casos de uso são especificados por *templates* descritivos de requisitos funcionais. Como resultado desta etapa tem-se o diagrama de casos de uso, apresentado na Figura 5.1, e os

templates referentes a cada caso de uso (o conjunto de *templates* de requisitos funcionais pode ser consultado no Apêndice A). Este processo de especificação dos requisitos funcionais deve passar por uma avaliação que determine a possível presença de conflitos entre tais requisitos. A análise baseada nas regras apresentadas no Anexo A não detectou a presença de nenhum conflito, de forma que se omitem maiores detalhes desta questão.

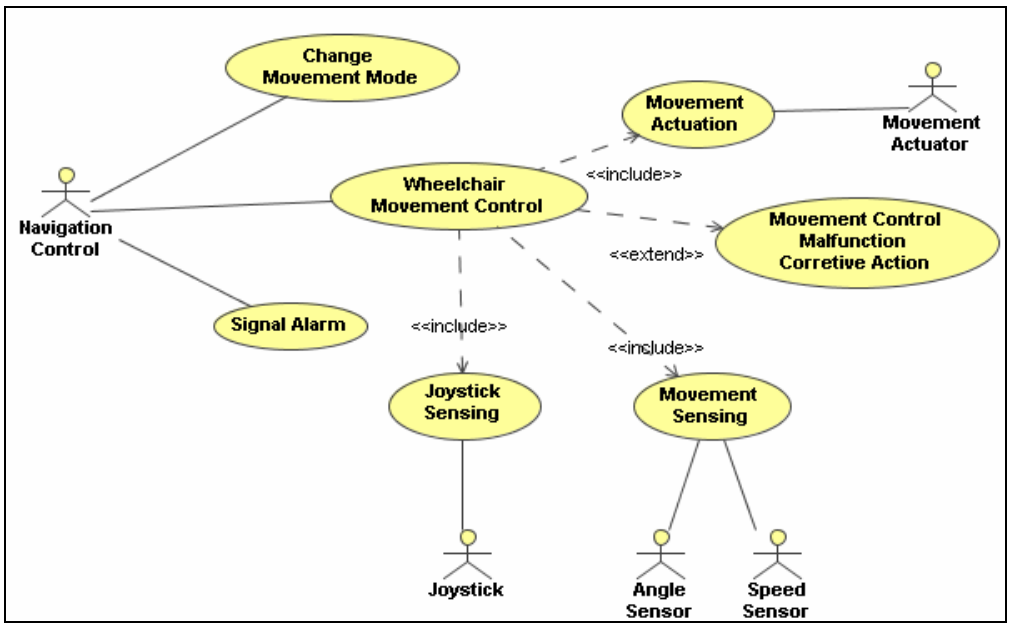

Figura 5.1: Diagrama de Casos de Uso do Controle de Movimento da Cadeira – somente requisitos funcionais

A segunda etapa se dedica ao estudo dos requisitos não-funcionais presentes na descrição do sistema. Como mencionado, estes requisitos encontram-se espalhados pela descrição envolvendo os requisitos funcionais já identificados. Para que seja possível o seu reconhecimento, utiliza-se o conjunto de *checklists* apresentado na subseção 4.2.1. As *checklists* preenchidas com as informações extraídas da descrição do sistema podem ser consultadas no Apêndice B. Para completar a identificação dos requisitos não-funcionais, utiliza-se ainda o léxico desenvolvido para o domínio dos sistemas TrED também apresentado na subseção 4.2.1 para se buscar algum requisito que não tenha sido identificado durante o preenchimento das *checklists*, ou então para se completar as *checklists* com alguma informação que tenha sido omitida. Como resultado deste processo de uso do léxico, atualizam-se as *checklists*.

Identificados os requisitos não-funcionais, passa-se então a sua especificação. Este procedimento é realizado através do preenchimento de *templates* de requisitos não-funcionais. O conjunto de *templates* de requisitos não-funcionais especificados pode ser consultado no Apêndice C. Especificados os requisitos não-funcionais, passa-se a análise e resolução de conflitos. Para se desempenhar esta tarefa deve-se montar a tabela de relacionamento entre requisitos não-funcionais utilizando as regras de identificação de conflitos para preencher a tabela. O resultado deste procedimento pode ser observado na Tabela 5.1. A área cinza da tabela não é utilizada, pois basta identificar o conflito entre dois requisitos uma única vez.

Tabela 5.1: Tabela de visualização de conflitos entre Requisitos Não-funcionais envolvidos no projeto da cadeira de rodas

	NFR-1	NFR-2	NFR-3	NFR-4	NFR-5	NFR-6	NFR-7	NFR-8	NFR-9	NFR-10	NFR-11
NFR-1											
NFR-2											
NFR-3											
NFR-4											
NFR-5											
NFR-6											
NFR-7											
NFR-8											
NFR-9											
NFR-10											X
NFR-11											

Pode-se perceber, portanto, a identificação de um conflito entre os requisitos NFR-10 (Consumo de Energia) e NFR-11 (Tempo de Resposta Final). Como o NFR-11 tem maior prioridade, ele é o que prevalece. Analisando este conflito, vê-se que o controle de consumo que eventualmente seria aplicado ao sistema poderia impactar negativamente o desempenho na questão específica do tempo final de resposta. Desta maneira, o controle de energia fica suspenso, permanecendo apenas o monitoramento do consumo (conforme pode ser observado na última coluna do *template* do NFR-10, que expressa a especificação do requisito pós-resolução do conflito, e que também pode ser consultado no Apêndice C). No referido apêndice, pode-se observar que na versão inicial da especificação tanto a descrição do requisito quanto o seu contexto faziam referência ao controle além do monitoramento do consumo de energia, pois o requisito levantado exigia estes dois tratamentos. Já na versão posterior à resolução do conflito com o requisito de tempo de resposta final, observa-se a retirada da questão envolvendo o controle e a manutenção apenas do monitoramento do consumo. Deve-se ainda destacar que os demais campos do *template* permanecem inalterados, salvo aquele reservado à situação do requisito, que após esta última especificação passa ser considerado finalizado.

Terminada a especificação dos requisitos não-funcionais, deve-se então relacionar graficamente tais requisitos aos funcionais no diagrama de casos de uso. Este relacionamento pode ser feito adicionando-se uma elipse para cada requisito não-funcional especificado, o que resultaria em 11 elipses para o presente estudo de caso, ou agrupando-se requisitos de acordo com a sua classificação e representando uma elipse para cada agrupamento. Para manter a clareza da figura, decidiu-se apresentar o diagrama de casos de uso com os requisitos não-funcionais agrupados em suas respectivas classes, como se pode observar na Figura 5.2.

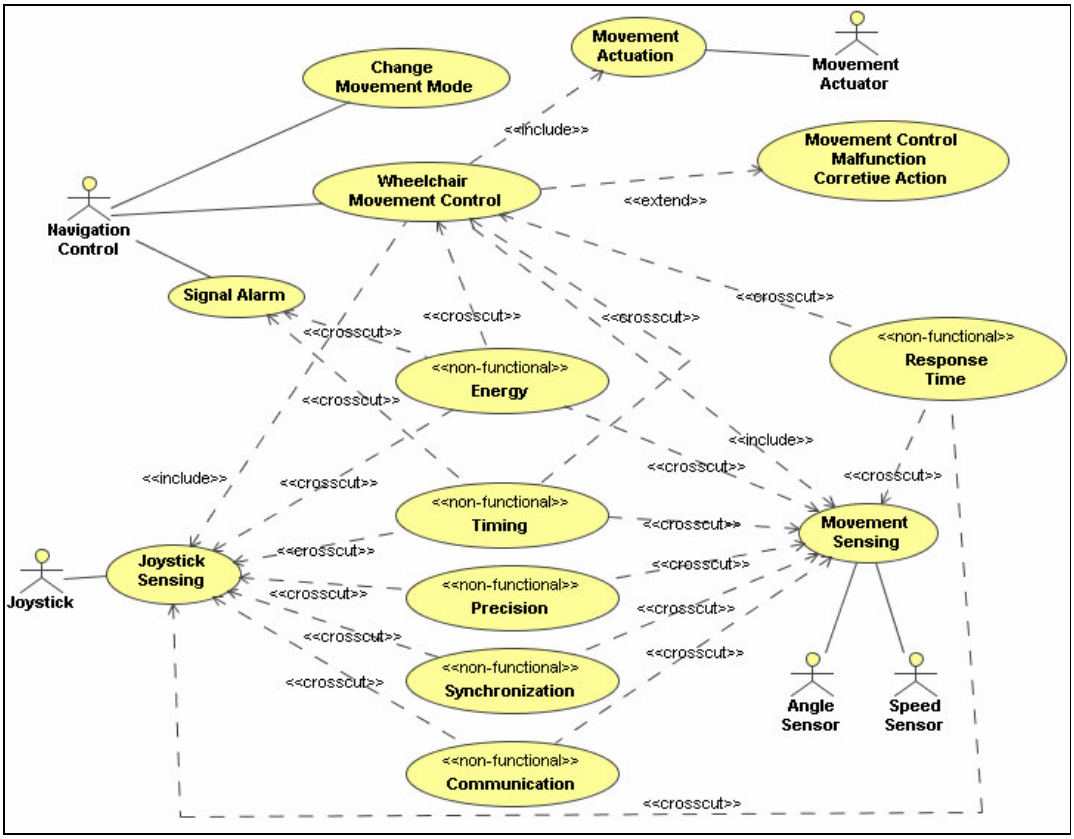

Figura 5.2: Diagrama de Casos de Uso do Controle de Movimento da Cadeira – representação de requisitos funcionais, não-funcionais e sua interação.

5.1.2 Mapeamento de Requisitos em Elementos de Projeto

Após a identificação e especificação dos requisitos envolvidos no sistema, passa-se à fase de especificação do tratamento que atenda tais requisitos. Um problema recorrente neste momento do desenvolvimento do sistema é a questão da documentação. Geralmente as soluções são propostas sem uma preocupação clara em se manter um elo que associe os requisitos levantados aos elementos de projeto que acabam por atendê-los. Este problema somente é percebido no momento em que se necessita realizar alguma alteração no sistema, seja para manutenção ou evolução, ou ainda quando se deseja reutilizar algum componente constituinte do projeto. O propósito então desta fase é realizar este elo entre os requisitos e projeto de modo a evitar o problema da identificação de que elemento de projeto trata determinado requisito.

Para cumprir o objetivo desta fase, monta-se uma tabela de mapeamento da seguinte forma: dispõem-se todos os requisitos não-funcionais na primeira linha e todos os requisitos funcionais na primeira coluna. O desenvolvedor, com o suporte da documentação gerada na primeira fase (identificação e especificação de requisitos), pondera sobre classes que possam atender os requisitos funcionais listados. Da mesma forma realiza-se a análise os requisitos não-funcionais e se determina aspectos que os atendam. Durante este processo o desenvolvedor busca reutilizar componentes (classes e aspectos) já desenvolvidos em outros projetos. No caso específico dos aspectos, existe ainda a biblioteca de aspectos de alto nível (definidos pelo DERAF), que já possui um conjunto substancial de aspectos a serem reutilizados. As classes são listadas na última

coluna com cada célula correspondendo ao requisito que atende, e os aspectos na última linha, com cada célula desta linha correspondendo ao requisito não-funcional listado na primeira linha que esteja sendo atendido. A caracterização do entrelaçamento entre requisitos é obtida marcando-se com um "X" as quadrículas que representam o encontro entre um requisito funcional afetado por um não-funcional. A Tabela 5.2 apresenta o mapeamento dos requisitos em elementos de projeto para o presente estudo de caso da cadeira de rodas.

Tabela 5.2: Tabela de Mapeamento de Requisitos em Elementos de Projeto para o Sistema de Controle de Movimento da Cadeira de Rodas

		Requisitos Não-Funcionais											
		NFR-1	NFR-2	NFR-3	NFR-4	NFR-5	NFR-6	NFR-7	NFR-8	NFR-9	NFR-10	NFR-11	Classes Responsável pelo tratamento do Requisito Funcional
Requisitos Funcionais	RF-1												**MovementController**
	RF-2		X		X	X					X	X	ControlSubSystem MovementController MovementActuator
	RF-3												**MovementActuator**
	RF-4												ControlSubSystem MovementController
	RF-5	X		X		X	X		X	X	X	X	**JoystickDriver** **JoystickInformation** **SensingSubSystem**
	RF-6	X		X		X	X		X	X	X	X	MovementEncoder MovementSensorDriver SpeedSensorDriver AngleSensorDriver SensingSubSystem
	RF-7							X			X		**ControlSubSystem** **Alarm**
	Aspectos Responsáveis pelo tratamento do Requisito Não-funcional	Periodic Timing Scheduling Support	Periodic Timing Scheduling Support	Periodic Timing Scheduling Support	Timing Atributes	Timing Atributes	Data Freshness	Timing Atributes	Concurrency Control Time Bonded Activity Tolered Delay	Message Ack Message Integrity Message Synchronization	Energy Monitoring	Timing Atributes Scheduling Support	

Deve-se ressaltar que esta tabela não é estática. Caso ocorra a necessidade de se criar uma nova classe ou aspecto na fase de projeto, deve-se retornar a esta tabela e atualiza-la de acordo com a alteração ocorrida. O importante é que esta tabela mapeie de fato quais requisitos estão sendo tratados por quais componentes presentes no projeto do sistema.

5.1.3 Projeto

O projeto do sistema de controle de movimento da cadeira de rodas segue as definições realizadas na fase de mapeamento. Com base nas informações contidas na documentação gerada naquela fase mais os *templates* gerados na primeira fase, constroem-se os diagramas que constituem o modelo do sistema.

Primeiramente constrói-se o diagrama de classes com a hierarquia proposta para o sistema. Para esta tarefa, utiliza-se a especificação dos estereótipos da RT-UML para caracterizar as classes do sistema, conforme a necessidade. O diagrama de classes proposto para o estudo de caso do sistema de controle da cadeira de rodas é apresentado na Figura 5.3. Como se pode observar na referida figura, o sistema é composto de quatro objetos ativos (caracterizados pelo estereótipo SAschedRes), por recursos de acesso concorrente (caracterizado pelo estereótipo SAresource), além das demais classes sem nenhum estereótipo específico sobre elas aplicado.

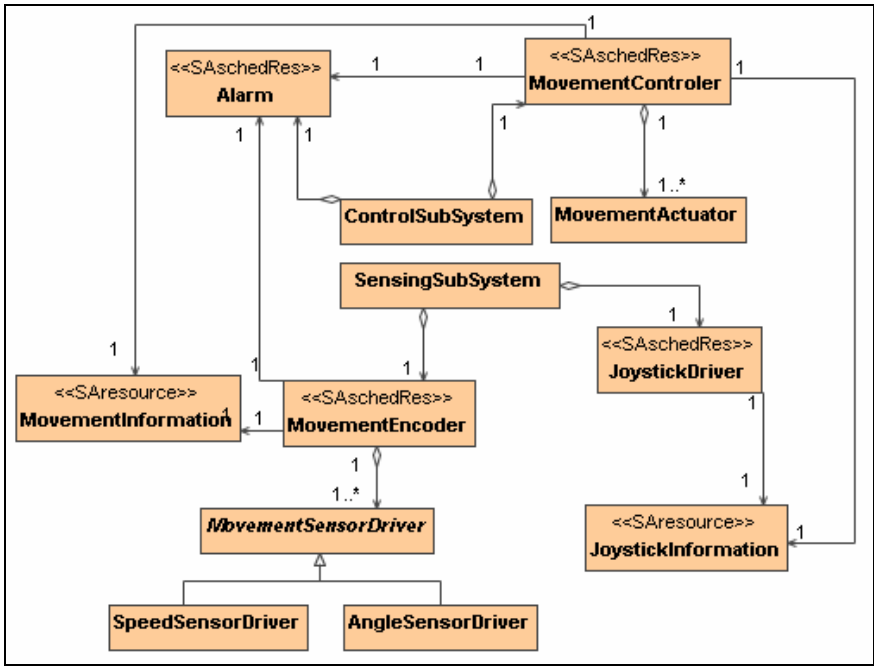

Figura 5.3: Diagrama de Classes

Durante a construção do diagrama de classe o desenvolvedor detalha cada classe com seus atributos e métodos, conforme a necessidade. Esta informação pode ser apresentada ou não no diagrama, de acordo com o nível de detalhe que se deseja observar. Na Figura 5.3 não foram apresentados os métodos e atributos, por uma questão de legibilidade e compreensão do diagrama. Porém, a ausência destes elementos em nada prejudica a compreensão do modelo.

Em seguida é feita a construção do diagrama ACOD (Aspect Crosscutting Overview Diagrama). Este diagrama apresenta somente as classes que tem suas funcionalidades afetadas por requisitos não-funcionais, e os aspectos utilizados para tratar estes requisitos. O entrelaçamento fica caracterizado através de um relacionamento navegável do aspecto para a classe afetada, estereotipado como “crosscut”. A Figura 5.4 apresenta uma versão compacta do diagrama (ocultando os elementos internos das classes e aspectos). Nesta versão compacta do ACOD ocultou-se também o estereótipo “crosscut” dos relacionamentos, para se manter a clareza da figura.

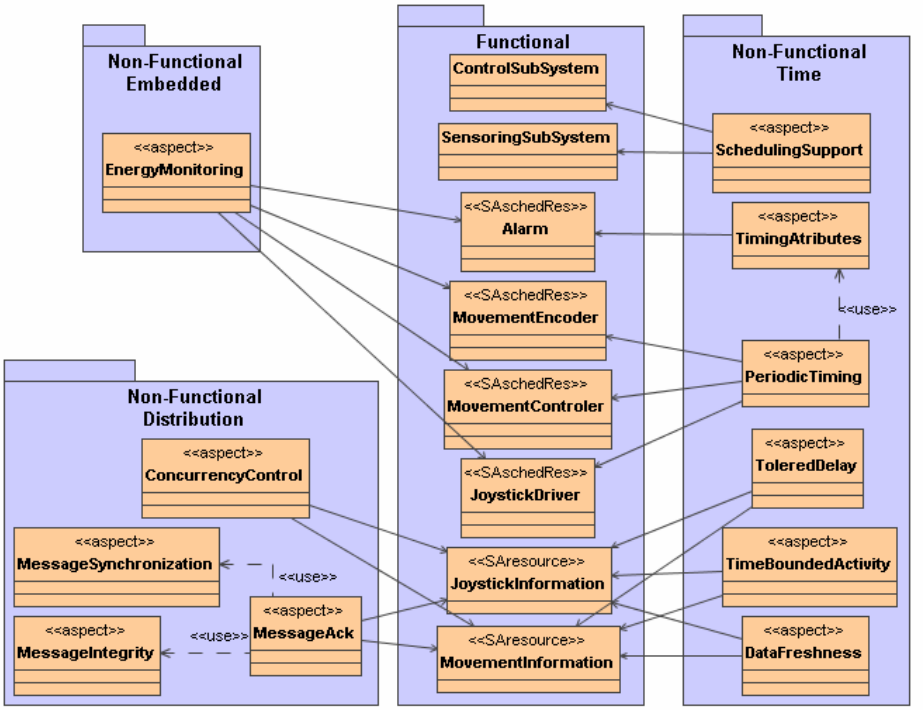

Figura 5.4: ACOD do Sistema de Controle de Movimento da Cadeira de Rodas

Eventualmente pode aparecer algum valor associado ao relacionamento *crosscut*, o que caracteriza a passagem de algum parâmetro na adaptação realizada pelo aspecto. Isto pode ser observado nos valores temporais presentes nos relacionamentos *crosscut* entre os objetos ativos (estereotipados como SAschedRes) e os aspectos TimingAtributes e PeriodicTiming apresentado na Figura 5.5, que apresenta uma visão parcial do ACOD detalhado. Observa-se, na referida figura, o detalhamento dos dois aspectos presentes. No aspecto PeriodicTiming existem quatro *pointcuts*, uma adaptação estrutural e três comportamentais. O primeiro *pointcut* tem seu ponto de entrelaçamento definido pelo *joinpoint* activeclass, e utiliza a *StructuralAdaptation* chamada Period definida neste aspecto. Já o segundo *pointcut* define um entrelaçamento que constitui numa adaptação comportamental. Ele tem seu ponto de entrelaçamento definido pelo *joinpoint* constructor, utiliza a adaptação comportamental SetupPeriod, que é inserida após o *joinpoint* (informação identificada pelo parâmetro AFTER). Os demais *pointcuts* têm semântica análoga a este último. Quanto às adaptações presentes no aspecto PeriodicTiming, a primeira se caracteriza de uma adaptação estrutural que insere o parâmetro "period" na classe entrelaçada. Já a adaptação SetupPeriod realiza a atribuição de valor ao atributo "period". A adaptação comportamental LoopMechanism insere o suporte necessário a tornar cíclica a execução da parte afetada. E por fim, a adaptação FrequencyControl insere o mecanismo de controle de freqüência de execução da parte afetada pela adaptação.

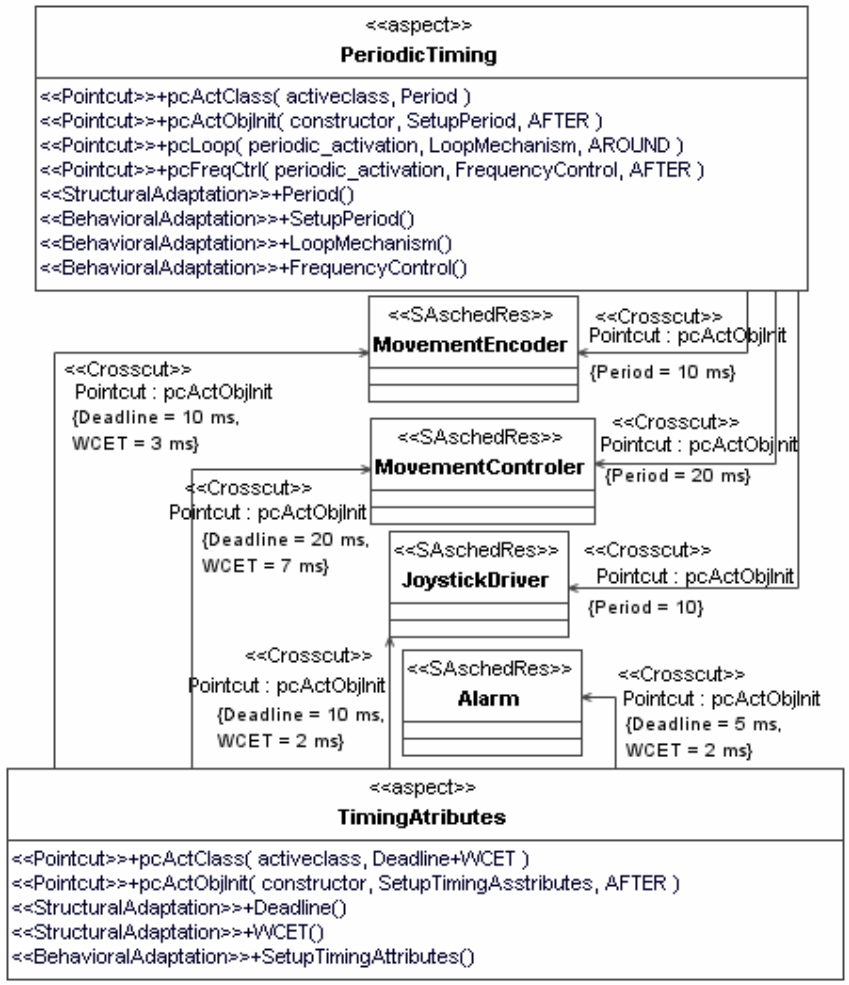

Figura 5.5: Visão parcial do ACOD detalhando entrelaçamentos de tempo.

É importante ressaltar no estudo de caso a utilidade do uso dos aspectos como elementos que concentram o tratamento de requisitos não-funcionais. Deve-se observar na Figura 5.5 que todos os elementos que caracterizam preocupação com requisitos temporais, como limites de tempo para execução ou períodos encontram-se especificados em um único elemento do projeto, dedicado apenas a este tipo de requisito não-funcional. Evita-se com isto o espalhamento deste requisito não-funcional por diversas entidades distintas do sistema. O tratamento não-funcional afeta aquelas diversas entidades, porém a sua especificação encontra-se concentrada em uma entidade dedicada. Outro ponto que merece destaque é o uso dos estereótipos do perfil RT-UML. Uma vez que determinados estereótipos possuem elementos intrínsecos, como o período em classes estereotipadas como SAschedRes, a aplicação dos estereótipos já identifica as classes não quais determinados atributos e mecanismos devem ser introduzidos através dos aspectos.

Para representar a dinâmica da interação entre os aspectos e as classes é necessário apresentar os pontos nos quais os aspectos as afetam. Estes pontos, chamados de *joinpoints*, se caracterizam pela determinação do local exato onde ocorre o entrelaçamento. Para realizar esta representação, este trabalho adota a utilização do diagrama JPDD. A seguir são apresentados os JPDDs que representam os *joinpoints*

utilizados pelos aspectos apresentados na Figura 5.5. Para se especificar os *joinpoints* a seguir apresentados, utiliza-se como subsídio as informações descritas no campo "contexto" do *template* de requisitos não-funcionais, no caso de uma adaptação comportamental, ou simplesmente a definição das classes afetadas (informação extraída da tabela de mapeamento) no caso de adaptações estruturais.

A Figura 5.6(a) representa o JPDD ActiveClass, que tem um *joinpoint* de mesmo nome que seleciona todas as classes estereotipadas como SAschedRes. Já a Figura 5.6(b) representa o JPDD ActiveObjectCreation, que tem o *joinpoint* chamado "constructor" que consiste na construção de um objeto de qualquer classe estereotipada como SAschedRes, no escopo de execução de qualquer objeto de qualquer classe.

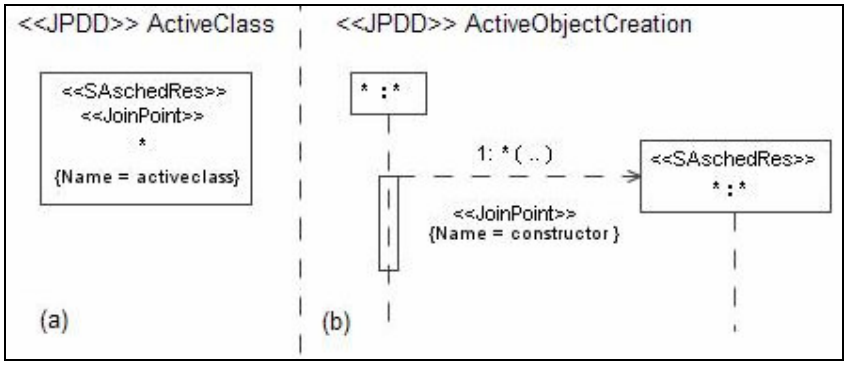

Figura 5.6: JPDDs: (a) Activeclass (b) ActiveObjectCreation

Na Figura 5.7 pode-se observar o *joinpoint* periodic_activation utilizado pelos *pointcuts* pcLoop e pcFreqCtrl. Ele seleciona a chamada de qualquer método estereotipado como SAtrigger de qualquer instância de qualquer classe estereotipada como SAschedRes por um objeto da classe Scheduler estereotipada como SAscheduler.

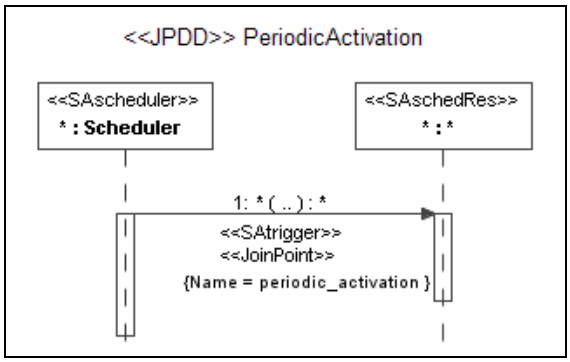

Figura 5.7: JPDD PeriodicActivation

A utilização dos diagramas JPDD apresentados nas Figuras 5.6 e 5.7 evidenciam os pontos de entrelaçamento onde os aspectos afetam as classes. Isto facilita a compreensão do projeto durante um processo de manutenção ou reuso.

5.2 Veículo Aéreo Não-Tripulado

Veículos aéreos não-tripulados (VANTs – em inglês UAV, unmanned air vehicle) estão sendo a cada mais utilizados em diversas atividades nas quais se evita a presença humana devido aos riscos inerentes ou até mesmo ao custo de prover segurança àqueles

que desempenham tais atividades. Alguns exemplos práticos de aplicação de VANTs são: (1) o monitoramento de áreas afetadas por catástrofes ambientais ou acidentes com produtos tóxicos ou radioativos, atuando no auxílio do resgate das vítimas; (2) aplicações militares de patrulhamento de áreas desmilitarizadas ou utilização em exercícios de reconhecimento; (3) monitoramento ambiental, com o objetivo de identificar áreas de ocorrência de algum incidente; (4) monitoramento de manutenção de linhas de transmissão (telefônicas ou elétricas) localizadas em lugares inóspitos; dentre diversas outras possibilidades de aplicação.

Um sistema VANT consiste de um complexo de subsistemas que devem atuar de modo a cumprir a missão para o qual foi destinado. Baseado em (DoD, 2005), uma descrição do sistema pode ser feita nos seguintes termos:

Um VANT tem, em sua maioria, missões relativas a monitoramento de áreas geográficas. Para isto, este aparelho necessita estar equipado de componentes como câmeras e suportes de gravação de vídeo, além de apresentar capacidade de transmissão de dados a uma estação base. A comunicação com esta estação deve ainda possibilitar a transmissão de mensagens de controle, que consistem basicamente no envio de dados sobre o vôo e status do aparelho, e a recepção de dados de gerência da missão, tais como modo de operação, plano de vôo e definição de objetivos. Para realizar sua finalidade, um VANT necessita de um sistema de navegação "inteligente", ou seja, este sistema deve ser autônomo, não necessitando de interação humana para definir nem atingir os pontos intermediários (*waypoints*) até o alvo da missão. Para isto, o sistema de navegação além de realizar o controle direto sobre a aeronave, deve ser capaz de calcular rotas, perseguir alvos e evitar colisões. Como um sistema autônomo, deve ser capaz de realizar diagnósticos de teste, bem como a gerência de energia e temperatura do sistema, emitindo alarmes em caso de detecção de problemas. Estes alarmes fornecerão subsídio para decisão de outras partes do sistema de que atividades devem ser priorizadas, dependendo do modo de operação e doutrina da missão. Quanto às doutrinas, pode-se ter uma grande diversidade, dependendo da aplicação do aparelho, porém existem duas doutrinas básicas que servem como referência. A primeira consiste no cumprimento incondicional da missão que se caracteriza pela necessidade de se atingir o objetivo (ou ao menos tentar atingi-lo), independentemente de se sacrificar o aparelho. A segunda doutrina encontra-se no extremo oposto, consistindo na preservação do aparelho. Segundo esta doutrina, caso a missão atinja um nível muito crítico, aborta-se o objetivo principal e passa-se a determinar o retorno do aparelho a estação base, a sua condução a um local seguro ou a adoção de alguma outra atitude que procure preservar sua integridade.

A descrição apresentada permite construir um diagrama de casos de uso que expressa as funcionalidades de um VANT. A Figura 5.9 mostra este diagrama.

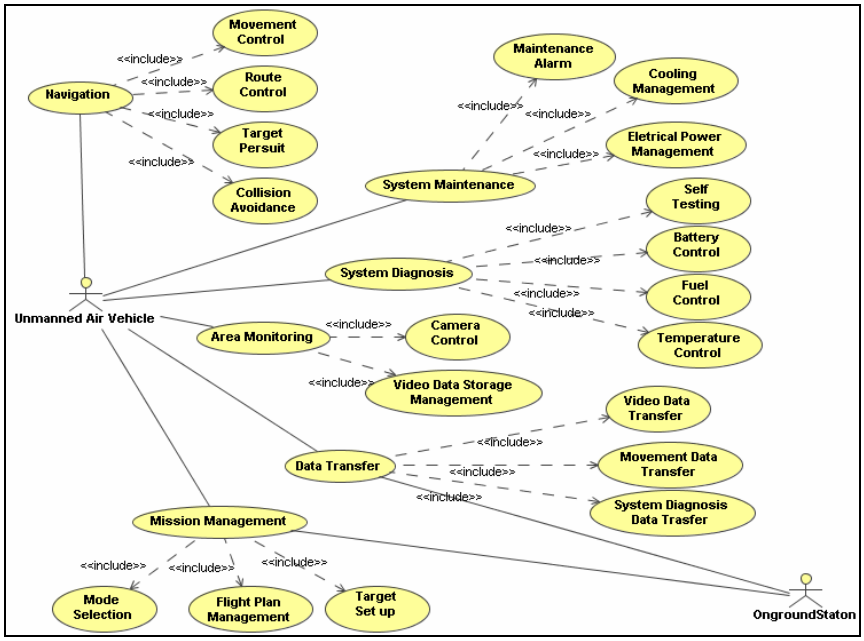

Figura 5.8: Diagrama de Casos de Uso para o projeto de um VANT

O diagrama da Figura 5.8 mostra a complexidade de um sistema VANT. O estudo de caso proposto nesta dissertação se concentra no tratamento do controle de movimento do VANT que consiste num veículo aéreo de asas rotativas (helicóptero), e se baseia em informações obtidas na descrição do projeto HELIX (GYRON, 1998) e no método de controle de missões de aeronaves não-tripuladas especificado em (SEIBEL, 2000). Com base nas referidas fontes, é apresentado na seqüência a descrição do sistema de controle de um helicóptero não-tripulado.

O sistema de controle do helicóptero é composto por módulos de sensoriamento, controle e atuação. O sensoriamento se divide em dois grupos, sendo o primeiro responsável por adquirir dados sobre o movimento da aeronave, que consistem basicamente no estado dos rotores principal e de cauda, além do valor de passo dos respectivos rotores. O segundo grupo de dados se refere ao monitoramento das condições climáticas, uma vez que variáveis como temperatura, umidade, direção e força do vento influem no controle e estabilidade do aparelho. Os dados de movimento do primeiro grupo devem ser fornecidos ao sistema de controle a uma taxa de 1000 amostras por segundo, enquanto os dados de monitoramento do ambiente devem ser fornecidos a taxa de 500 amostras por segundo para os dados de direção e velocidade do vento e 10 amostras por segundo para os dados de umidade e temperatura. Estes dados têm uma vida útil de até 25 milisegundos em condições normais de funcionamento e 30 milisegundos em condições especiais de operação que não priorizem o controle da aeronave. Vencido o prazo de validade, os dados não têm utilidade para o sistema. O controle deve fornecer dados novos de atuação sobre a rotação e passo dos rotores a cada 15 milisegundos. A atividade de controle não deve demorar mais que 2 milisegundos para iniciar sua execução a partir do início de um novo ciclo de controle. O sistema de controle deve então fornecer os dados para os atuadores realizarem a aplicação dos novos parâmetros de movimento nos rotores, controlando o guiamento (ou guiagem), que corresponde ao controle do centro de massa da aeronave em relação a um sistema de coordenadas, e a pilotagem, que consiste no controle da aeronave em torno do seu centro de massa. Para realizar a guiamento, deve-se atuar sobre o passo e

torque do rotor principal, enquanto que para se realizar a pilotagem deve-se atuar sobre o passo do rotor de cauda. As atividades de leitura de dados e controle devem respeitar os intervalos de tempo reservados para sua execução, tendo como prazo limite o final da respectiva janela de tempo reservada a sua execução.

O projeto da aeronave exige que o processamento da amostragem dos dados de movimento seja feito espacialmente junto aos sensores de origem da informação, e que somente os dados já digitalizados e normalizados sejam transmitidos para a unidade que realiza o processamento do controle. Uma vez que o rotor de cauda se encontra distante desta unidade de processamento, existe a necessidade de se aplicar um controle sobre a comunicação das mensagens que enviam os dados. Este controle consiste basicamente na garantia de entrega e integridade dos dados transmitidos. Caso ocorra algum problema com a unidade de processamento local dos dados obtidos pelos sensores, pode-se migrar a atividade relativa ao seu tratamento para outra unidade que não esteja sobrecarregada.

O acesso aos dados de controle e monitoramento de ambiente é protegido. Como existe concorrência no acesso destes dados, o sistema deve prover proteção que garanta acesso exclusivo. Esta proteção deve contar ainda com um limitante temporal para aquisição do recurso.

Problemas em qualquer atividade de processamento de dados dos sensores ou no controle da aeronave devem gerar um alarme que transmitirá este problema para o sistema de manutenção, além de requerer a transmissão da informação sobre o incidente para a estação base. As características temporais deste alarme são as seguintes: limite de execução de 6 milisegundos e deve ter um custo máximo de processamento de 3 milisegundos. Ele deve ainda obedecer a condição de entrar em execução 1 milisegundo após a identificação de um problema.

O controle de consumo de energia da aeronave é prioritário em todos os subsistemas. Mesmo no sistema de controle de movimento é possível reduzir a atividade com objetivo de economia de recurso energético em determinada situações (e.g. aeronave pairando no ar). Desta forma o monitoramento e controle do consumo deve se aplicar também ao controle de movimento.

Realizando uma missão que siga a doutrina do cumprimento incondicional, o sistema de controle da aeronave pode ser penalizado caso outras atividades, como utilização da câmera de vídeo e transmissão de dados para estação base, demandem processamento extra. Neste caso, deve-se relaxar os tempos de execução das atividades de controle além de diminuir o número de amostras requeridas dos sensores. Porém, caso a missão siga a doutrina de preservar a aeronave, o sistema de controle deve ser priorizado sobre os demais e estando a aeronave em perigo, os controles e sensoriamentos envolvidos com o controle devem respeitar rigorosamente as restrições temporais. O laço de controle deve ser mais freqüente passando a ser realizado a cada 10 milisegundos.

5.2.1 Identificação e Especificação de Requisitos

A descrição do sistema de controle do helicóptero não-tripulado apresenta outras particularidades que fogem ao escopo do objetivo desta dissertação, porém o texto descritivo apresentado consegue resumir as principais atividades envolvidas neste processo, bem como suas restrições e condições de funcionamento. Como na descrição

da cadeira de rodas apresentada no primeiro estudo de caso, também é possível perceber o entrelaçamento entre requisitos funcionais e não-funcionais na descrição do sistema de controle da aeronave. Realizando as atividades descritas na subseção 4.2.1, busca-se, portanto separar a apresentação de tais requisitos. Com esta separação é possível uma melhor análise que proporciona um posterior benefício em seu tratamento na fase de projeto.

O primeiro passo da análise baseada na descrição do sistema busca apresentar o levantamento dos requisitos funcionais. Estes requisitos estão resumidos no diagrama de casos de uso apresentado na Figura 5.9. A especificação de cada caso de uso pode ser consultada no Apêndice D.

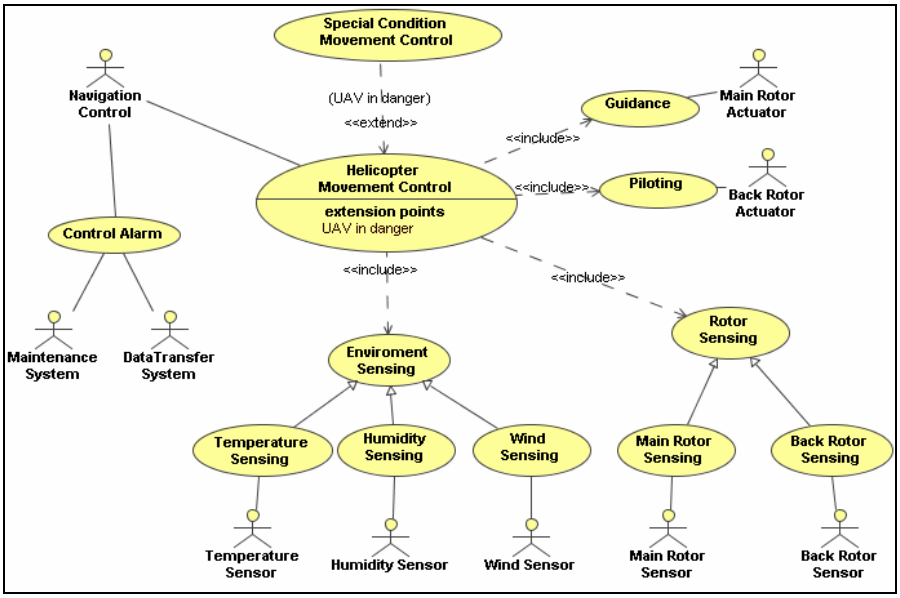

Figura 5.9: Diagrama de Casos de Uso do Sistema de Controle de Movimento do Helicóptero Não-tripulado – somente requisitos funcionais

Continuando a análise do sistema, procede-se a análise dos requisitos não-funcionais. Realiza-se então o preenchimento das *checklists* na busca de se explicitar tais requisitos. O conjunto de *checklists* gerado por este estudo de caso se encontra disponível para consulta no Apêndice E. Segue então a procura por requisitos não-funcionais obscuros através do uso dos léxicos específicos para o domínio de interesse. O resultado do uso destes artefatos permite complementação das *checklists*.

Devidamente identificados, os requisitos não-funcionais passam a ser então especificados. Esta tarefa é realizada através do preenchimento de um *template* não-funcional para cada requisito desta natureza. Os *templates* que especificam os requisitos não-funcionais levantados neste estudo de caso encontram-se no Apêndice F. Com as informações levantadas na identificação e especificação monta-se a tabela de análise e resolução de conflitos. O conteúdo desta análise encontra-se apresentado na Tabela 5.3.

Tabela 5.3: Tabela de visualização de conflitos entre Requisitos Não-funcionais envolvidos no projeto do Sistema de Controle do Helicóptero Não-tripulado

	NFR-1	NFR-2	NFR-3	NFR-4	NFR-5	NFR-6	NFR-7	NFR-8	NFR-9	NFR-10	NFR-11
NFR-1											
NFR-2											
NFR-3											
NFR-4							X				
NFR-5											
NFR-6											
NFR-7											
NFR-8									X		
NFR-9										X	
NFR-10											
NFR-11											

Os conflitos identificados e documentados na Tabela 5.3 são os seguintes: Conflito entre o requisito NFR-4 (Prazo de Validade) e NFR-7 (Controle sobre a Comunicação) e o conflito entre o requisito NFR-9 (Alocação Espacial de Atividades) e os requisitos NFR-8 (Energia) e NFR-10 (Migração de Tarefas). O primeiro conflito se dá porque o NFR-7 estipula o controle da comunicação através do uso de um código de verificação de integridade além da confirmação do recebimento. A adição do controle de integridade acaba colidindo com as restrições impostas no requisito que trata a validade temporal do uso dos dados (o NFR-4). Isto acontece porque podem ocorrer situações nas quais o dado transmitido possa chegar com pouco tempo de vida útil ao destino, desta forma, o acréscimo da computação do código de integridade acaba por consumir um tempo que pode levar a inutilização do dado transmitido. Com isto, a solução para o conflito foi remover a utilização da verificação de integridade. Já o duplo conflito identificado entre o requisito NFR-9 e os requisitos NFR-8 e NFR-10, apresenta uma preocupação com a distribuição das atividades do sistema entre as unidades de processamento, levando em conta critérios de economia de energia. Na descrição do sistema foi mencionado que é interessante manter as atividades que tratam determinado processo do sistema, como o controle dos rotores ou leitura de dados, nas unidades de processamento próximas de onde ocorre o processo. Porém a descrição também deixa clara a preocupação com a economia de energia, o que demanda uma flexibilidade do sistema no sentido de poder migrar tarefas entre processadores de modo a minimizar o consumo e otimizar o uso do recurso computacional. Com isto, o requisito de alocação de atividades foi adaptado para que seja tratado, mas levando em consideração sua interação com os outros dois com os quais conflita. A versão pós-resolução de conflitos dos requisitos mencionados também se encontra no Apêndice F.

Após a resolução dos conflitos entre requisitos não-funcionais, relaciona-se graficamente estes requisitos com os funcionais. Este relacionamento pode ser observado na Figura 5.10, que apresenta o diagrama de casos de uso completo com os requisitos funcionais e não-funcionais e seus relacionamentos.

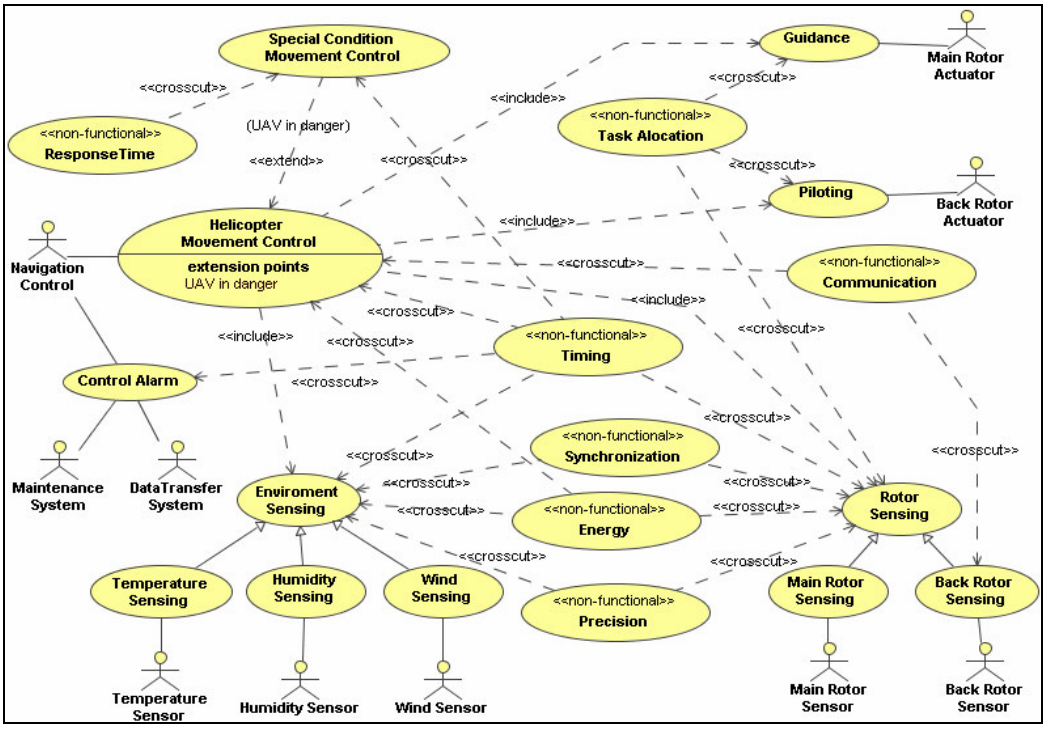

Figura 5.10: Diagrama de Casos de Uso do Sistema de Controle do Helicóptero Não-tripulado – representação de requisitos funcionais, não-funcionais e sua interação.

5.2.2 Mapeamento de Requisitos em Elementos de Projeto

Identificados e especificados os requisitos do sistema, passa-se então a fase de mapeamento de requisitos em elementos de projeto. Para isto deve-se, portanto analisar a documentação produzida na fase anterior e realizar a concepção de elementos de projeto que tratem os requisitos levantados.

Durante esta concepção, propõe-se classes para o atendimento dos requisitos funcionais e aspectos para o atendimento de requisitos não-funcionais. Como se dispõe de uma biblioteca de aspectos (a DERAF) especificada para o tratamento de requisitos não-funcionais, observa-se inicialmente que requisitos podem ser tratados por aspectos presentes na biblioteca, caso algum requisito não esteja contemplado, estende-se a biblioteca (para o caso de tratamento específico de requisito do domínio), ou cria-se um aspecto específico para atender a necessidade do projeto.

Quanto à definição das classes que farão parte do projeto, realiza-se a análise dos requisitos funcionais e determina-se que classes serão responsáveis por tratar determinado requisito. Neste ponto é importante atentar para a questão do reuso. Uma vez que já existe o projeto de um sistema de controle de movimento para a cadeira de rodas, pode-se voltar àquele projeto e tentar-se identificar alguma classe que seja reutilizável no projeto do sistema de controle do helicóptero. O resultado desta análise leva às seguintes conclusões:

a) Ambos os sistemas apresentam sensoriamento de dados do movimento e necessitam processar estes dados antes de utilizá-los em seus processos de controle. No sistema da cadeira de rodas as classes responsáveis pelo tratamento deste requisito são MovementEncoder, MovementInformation,

MovementSensorDriver, SpeedSensorDriver, AngleSensorDriver e SensingSubSystem. As três primeiras classes podem ser reutilizadas no projeto do sistema de controle de movimento do helicóptero. Com o isolamento dos requisitos não-funcionais que afetam tais classes no primeiro projeto em aspectos, a sua reutilização no segundo projeto é direta, pois a adaptação às restrições do novo projeto será feita através da aplicação de aspectos (com novas restrições e condições) sobre estas classes que forem reutilizadas.

b) O controle de movimento, requisito funcional comum aos dois projetos, apresenta as seguintes classes no projeto da cadeira de rodas: ControlSubSystem, MovementControler e MovementActuator. A primeira classe pode ser reutilizada com o acréscimo de pequenos detalhes que seriam os referentes aos atuadores específicos do projeto do helicóptero. O mesmo pode-se dizer para a classe MovementControler, que também pode ser reutilizada.

c) Tanto no projeto do sistema de controle da cadeira quanto no do controle do helicóptero existe a necessidade de um alarme que denuncie o comportamento incorreto de elementos do sistema. No primeiro projeto este requisito é tratado pela classe Alarm e ControlSubSystem. A primeira classe pode ser reutilizada diretamente, tendo suas propriedades temporais adaptadas através de aspectos. A segunda também pode ser reutilizada para o atendimento deste requisito sem modificações.

Criam-se então os elementos de projeto que faltam para tratar os demais requisitos e preenche-se a tabela de mapeamento de requisitos em elementos de projeto, obtendo-se como resultado a Tabela 5.4.

Tabela 5.4: Tabela de Mapeamento de Requisitos em Elementos de Projeto para o Sistema de Controle de Movimento Helicóptero Não-tripulado

		Requisitos Não-Funcionais											
		NFR-1	NFR-2	NFR-3	NFR-4	NFR-5	NFR-6	NFR-7	NFR-8	NFR-9	NFR-10	NFR-11	Classes Responsável pelo tratamento do Requisito Funcional
Requisitos Funcionais	RF-1		X	X	X			X	X			X	**MovementController**
	RF-2												SpecialConditionMovementControl
	RF-3								X	X	X		**RotorActuator** **MainRotorActuator**
	RF-4							X	X	X	X		RotorActuator BackRotorActuator
	RF-5					X			X				**ControlSubSystem** **Alarm**
	RF-6												EnviromentSensorDriver TemperatureSensorDriver
	RF-7												**EnviromentSensorDriver** **HumiditySensorDriver**
	RF-8												EnviromentSensorDriver WindSensorDriver
	RF-9	X		X	X		X		X		X	X	**EnviromentSensingSubSystem** **EnviromentDataSampler** **Enviromentinformation**
	RF-10												MovementSensorDriver MainRotorSensorDriver
	RF-11												**MovementSensorDriver** **BackRotorSensorDriver**
	RF-12	X		X	X		X	X	X	X	X	X	MovementSensingSubSystem MovementEncoder MovementInformation
	Aspectos Responsáveis pelo tratamento do Requisito Não-funcional	Periodic Timing Scheduling Support	Periodic Timing Scheduling Support	Timing Atributes	Data Freshness	Timing Atributes	Concurrency Control Tolered Delay	Message Ack Message Synchronization	Energy Control Energy Monitoring	Task Migration	Task Migration	Time Parameters Adapter	

5.2.3 Projeto

De posse da tabela de mapeamento de requisitos em elementos de projeto e das informações levantadas na fase de identificação e especificação de requisitos, inicia-se o projeto do sistema de controle da aeronave não-tripulada.

Constrói-se o diagrama de classes que explicita a relação entre as classes componentes do sistema, utilizando-se ainda os estereótipos definidos na RT-UML para anotar as classes conforme a necessidade. Na Figura 5.11 é apresentado o diagrama de classes no qual se pode observar ainda as classes reutilizadas do sistema de controle de movimento da cadeira de rodas, a saber: MovementControler, ControlSubSystem, Alarm, MovementEncoder, MovementInformation e MovementSensorDriver.

O diagrama apresentado na Figura 5.11 não apresenta uma visão completa das classes, ou seja, não apresenta seus métodos e atributos. O objetivo desta figura é apenas visualizar a estrutura do sistema através de suas classes e relacionamentos. Detalhes sobre a estrutura interna das classes são apresentados ao longo do texto quando necessário ao entendimento.

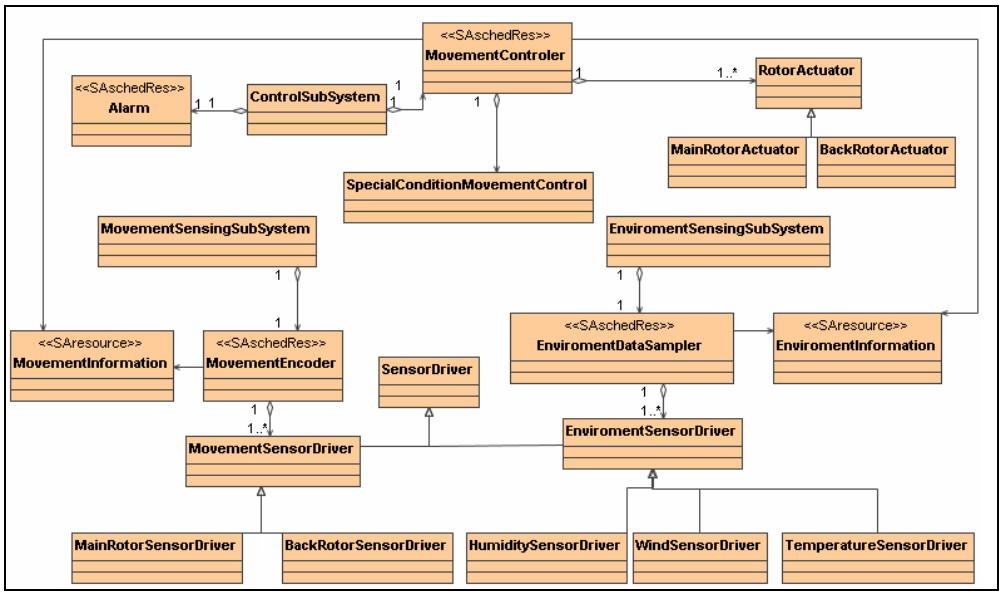

Figura 5.11: Diagrama de Classes do Sistema de Controle de Movimento do Helicóptero

O próximo diagrama a ser construído é o ACOD, que apresenta apenas as classes afetadas por requisitos não-funcionais e os aspectos utilizados para tratá-los. A versão compacta deste diagrama (sem a estrutura interna dos elementos) é apresentada na Figura 5.12.

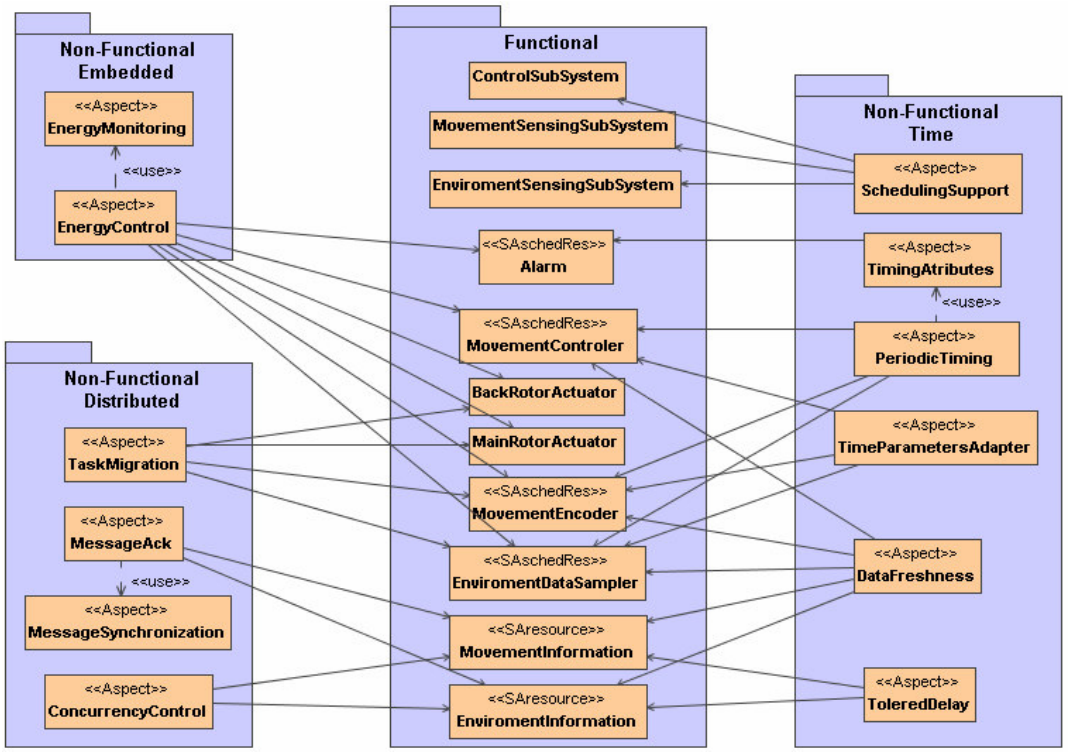

Figura 5.12: ACOD do Sistema de Controle de Movimento do Helicóptero

O diagrama ACOD da Figura 5.12 apresenta, além dos aspectos utilizados da biblioteca DERAF, um novo aspecto definido na fase de mapeamento, o TimeParametersAdapter. O objetivo deste aspecto é prover suporte ao requisito NFR-11, que trata da mudança de alguns parâmetros temporais em caso de movimentação da aeronave em condições especiais (críticas). Basicamente este aspecto re-configura os parâmetros temporais de alguns objetos ativos do sistema, de modo a atender a demanda expressa na descrição do sistema.

Os relacionamentos *crosscut*, que caracterizam a intervenção de um aspecto sobre uma classe, podem apresentar valores associados no caso em que estas relações inserem algum elemento estrutural na classe afetada. Na Figura 5.13 é apresentada uma visão parcial do diagrama ACOD detalhado que permite a visualização de valores de tempo passados pelo aspecto DataFreshness às classes MovementInformation e EnviromentInformation. Além disto, pode-se observar nesta mesma figura o relacionamento do aspecto com a Movementcontroler, MovementEncoder e EnviromentDataSampler.

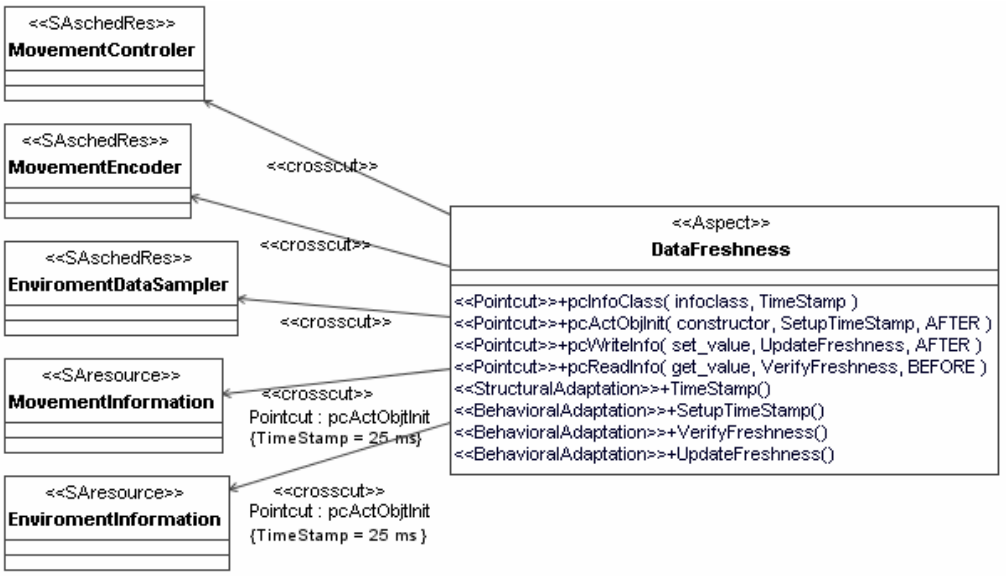

Figura 5.13: Visão parcial do ACOD detalhado destacando os entrelaçamentos realizados pelo aspecto DataFreshness

O aspecto DataFreshness apresenta quatro *pointcuts*, sendo o primeiro responsável por capturar as classes que contenham informação com validade temporal, que é a informação definida em seu primeiro parâmetro (*joinpoint* infclass), e neste ponto aplicar a adaptação estrutural TimeStamp. O segundo *pointcut* trata da captura da construção de um objeto da classe afetada pelo aspecto DataFreshness, sendo responsável por relacionar esta construção à adaptação comportamental SetupTimeStamp. O terceiro *pointcut* trata da captura da escrita de um dado com validade temporal. Seu primeiro parâmetro especifica o ponto do entrelaçamento, o segundo associa à adaptação comportamental UpdateFresshness, e o último define a posição em que será aplicada a modificação, sendo neste caso depois de ocorrer do *joinpoint* associado. O último *pointcut* é responsável pela captura da leitura de dados com validade temporal, tendo seus parâmetros interpretação semelhante ao anteriormente apresentado. Além destes elementos o aspecto em questão apresenta ainda as quatro adaptações utilizadas pelos *pointcuts*. A primeira adaptação consiste de uma mudança estrutural que insere o atributo TimeStamp associando-o a um dado da classe afetada, sendo a segundo uma mudança comportamental responsável pela atribuição do valor ao TimeStamp, enquanto as outras duas adaptações representam adaptações comportamentais que verificam e atualizam o valor do TimeStamp, respectivamente.

Complementando-se a informação da interação entre classes e o aspecto DataFreshness descrita acima, apresenta-se a seguir os diagramas JPDDs associados. Para construção destes diagramas é importante consultar o campo "contexto" dos *templates* de requisitos não-funcionais especificados na primeira fase do desenvolvimento.

A Figura 5.14(a) apresenta o diagrama JPPD InformationClass, que representa o *joinpoint* "infoclass" responsável por selecionar todas as classes estereotipadas como SAresource e que tenham a string "Information" no final de seu nome. Na Figura 5.14(b) observa-se o JPDD InformationObjectCreation, que representa o *joinpoint* de nome "constructor" que consiste na construção de um objeto de qualquer classe

estereotipada como SAresource que tenha o nome terminado pela string “Information”, no escopo de execução de qualquer objeto de qualquer classe.

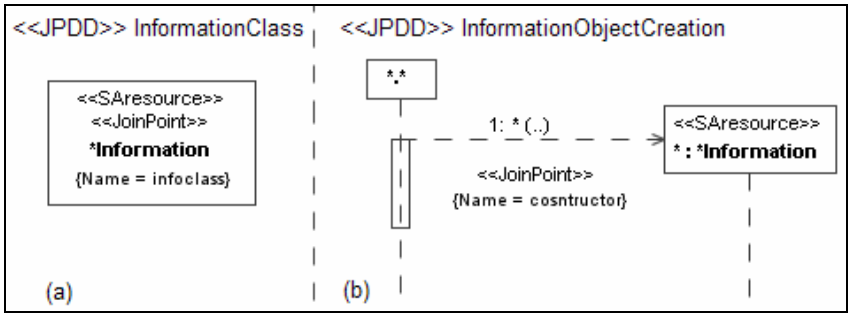

Figura 5.14: JPDDs: (a) InformationClass (b) InformationObjectCreation

O *joinpoint* “set_value” é descrito pelo JPDD WriteTimedInformation, que é apresentado na Figura 5.15(a). Nesta figura pode-se observar que este *joinpoint* especifica a chamada de qualquer método que comece com a string “set”, com qualquer parâmetro e tipo de retorno, de qualquer instância de todas as classes que tem a string “Information” no final de seu nome e sejam estereotipadas como SAresource, por qualquer instância de todas as classes estereotipadas como SAschedRes. Na Figura 5.15(b), é apresentado o JPDD ReadTimedInformation, que apresenta uma descrição semelhante a exposta para o WriteTimedInformation, porém para métodos que iniciam com a string “get”.

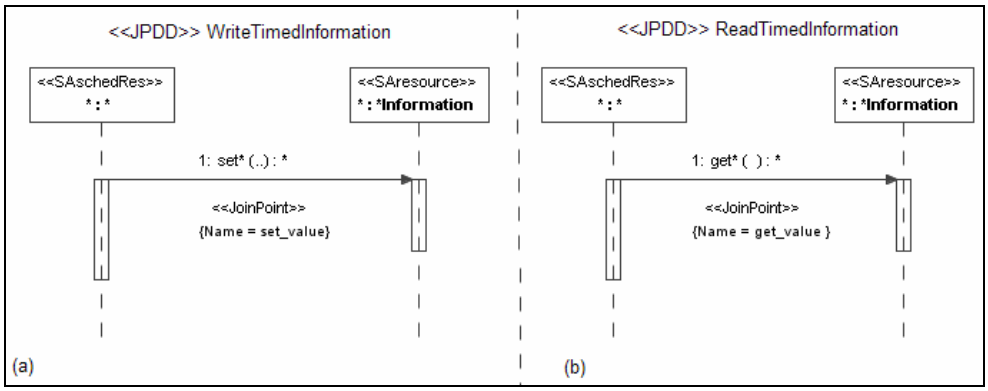

Figura 5.15: JPDDs: (a) WriteTimedInformation (b) ReadTimedInformation

6 AVALIAÇÃO DOS RESULTADOS

A aplicação de métricas sobre os modelos desenvolvidos para os estudos de caso propostos no capítulo 5 e sua comparação com modelos equivalentes orientados a objetos permite aferir o impacto da utilização da metodologia proposta ao projeto de sistemas TrED. Para realizar a análise dos resultados obtidos, são apresentadas primeiramente as métricas e critérios de avaliação considerados. Em seguida são apresentados os resultados comparativos acompanhados de sua análise.

6.1 Métricas e Critérios de Avaliação

As qualidades do sistema que este trabalho se propõe a analisar se referem à capacidade de reuso e manutenibilidade. Para isto, o modelo de qualidade utilizado para a análise é o proposto em (SANT'ANNA et al 2003), cuja descrição encontra-se na subseção 2.5.2. O motivo desta escolha se baseia no fato deste ser um modelo de aferição de qualidade baseado em métricas amplamente aceitas (como as utilizadas na suíte de métricas C&K), bem como por esta proposta adequar tais métricas ao contexto da orientação a aspectos. Outro motivo que contribuiu na escolha foi a adequação das métricas propostas à avaliação do projeto de sistemas e não apenas à sua implementação. Naturalmente não serão utilizadas todas as métricas propostas, justamente por algumas delas serem específicas para implementação, porém isto em nada invalida a proposta do modelo de qualidade, embora este tenha sido adaptado como a seguir descrito.

A adaptação da modelo de qualidade proposto em (SANT'ANNA et al 2003) consiste basicamente na eliminação das métricas destinadas à análise do código-fonte do sistema. Desta forma, o modelo utilizado no presente trabalho é o apresentado na Figura 6.1.

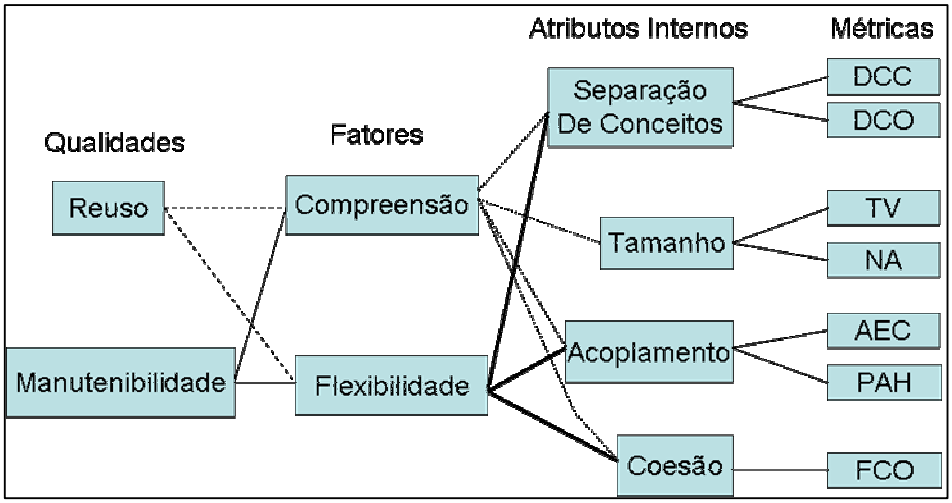

Figura 6.1: Modelo de Qualidade Adaptado

Como se pode observar na Figura 6.1, as métricas mantidas são as seguintes (de acordo com as suas respectivas classes): (1) Separação de Conceitos (Preocupações/Interesses): (i) Difusão de Conceito por Componentes (DCC); (ii) Difusão de Conceitos por Operações (DCO); (2) Tamanho: (i) Tamanho do Vocabulário (TV); (ii) Número de Atributos (NA); (3) Acoplamento: (i) Acoplamento Entre Componentes (AEC); (ii) Profundidade da Árvore de Herança (PAH); e (4) Coesão: (i) Falta de Coesão nas Operações (FCO). A semântica de cada métrica se mantém aquela descrita na subseção 2.5.2. Outro ponto importante de se observar é que nas métricas que apresentam cômputo de atributos ou métodos (elementos internos de componentes do sistema), está sendo levado em conta apenas os elementos com significado semântico para o projeto, ou seja, elementos internos que são criados para auxiliar a implementação (tais como variáveis auxiliares e métodos que realizam algum tipo de cálculo não especificamente relacionado a um requisito do projeto) não são contados. Isto se dá até mesmo porque tais elementos são geralmente criados na fase de implementação do sistema, fase esta que não é alvo de estudo do presente trabalho.

Para que se possa compreender a análise baseada nas métricas apresentadas, é necessário o entendimento de como estes critérios se relacionam para que se atinjam os atributos de qualidade do sistema. Este entendimento possibilita a interpretação das métricas de modo a se inferir as qualidades de reuso e manutenibilidade do sistema, conforme descrito na subseção 2.5.2.

6.2 Apresentação e Análise dos Resultados

Os resultados obtidos pelos estudos de caso apresentados neste trabalho são confrontados com os resultados do projeto dos mesmos sistemas com uso da orientação a objetos. No caso do projeto da cadeira de rodas, a versão orientada a objetos utilizada como fonte de comparação encontra-se descrita em (WEHRMEISTER, 2005). Já projeto orientado a objetos do sistema de controle da aeronave não-tripulada encontra-se resumidamente descrito no Apêndice G.

6.2.1 Cadeira de Rodas Automatizada

A aplicação do conjunto de métricas relacionadas à separação de conceitos (preocupações ou interesses) ao estudo de caso do sistema de controle de movimento da cadeira de rodas forneceu os resultados apresentado na Tabela 6.1.

Tabela 6.1: Resultados das Métricas de Separação de Conceitos Aplicadas ao Projeto do Sistema de Controle de Movimento da Cadeira de Rodas

Preocupação	**Tempo**		**Distribuição**		**Embarcados**		**TOTAL**	
Métricas	**DCC**	**DCO**	**DCC**	**DCO**	**DCC**	**DCO**	**DCC**	**DCO**
Projeto OO	10	23	8	17	7	8	**25**	**48**
Projeto OA	5	8	4	7	2	3	**11**	**18**

Na Tabela 6.1 pode-se observar a distribuição dos resultados por preocupação (tempo, distribuição e embarcado) das métricas DCC e DCO para os projetos orientados a objetos e a aspectos, além dos valores totais que representam a soma dos valores de cada preocupação. Pelos resultados apresentados, percebe-se uma melhora na separação das preocupações no projeto orientado a aspectos. Enquanto no projeto OO estas preocupações estavam espalhadas por diversos componentes (classes) do sistema, no projeto OA elas foram reunidas em componentes separados específicos para este fim (os aspectos). Com isto, houve uma diminuição no valor das métricas, o que aponta para uma melhora na qualidade do sistema. Nas preocupações envolvidas com o requisito "Tempo", a melhora foi de 50% para a métrica DCC e 65% para DCO. Estes resultados, apesar de serem bastante significativos, poderiam ser ainda mais expressivos no projeto de sistemas com maior número de objetos ativos (classes estereotipadas como SAschedRes). O mesmo tipo de comentário é válido para as preocupações que envolvem aspectos de distribuição e embarcados, uma vez que o maior número de classes afetadas por requisitos não-funcionais em um sistema OO aumentaria substancialmente o espalhamento destas preocupações, enquanto que num sistema OA as métricas teriam pequenos aumentos, tornando assim ainda mais expressiva a melhora percentual das métricas no projeto OA em relação ao OO.

O próximo conjunto de valores a ser apresentado traz as demais métricas empregadas na avaliação das duas versões do projeto do sistema de controle da cadeira de rodas, conforme disposto na Tabela 6.2.

Tabela 6.2: Resultados das Métricas de Acoplamento, Coesão e Tamanho Aplicadas ao Projeto do Sistema de Controle de Movimento da Cadeira de Rodas

Atributos Internos	**Acoplamento**		**Coesão**	**Tamanho**	
Métricas	**AEC**	**PAH**	**FCO**	**TV**	**NA**
Projeto OO	39	1	87	18	48
Projeto OA	35	1	62	23	25

Analisando os resultados obtidos para o acoplamento, pode-se observar que não houve uma grande diferença entre os projetos OA e OO. Houve sim uma pequena melhora na métrica de acoplamento entre componente (AEC), mas não muito expressiva. Já quanto à métrica relacionada à profundidade da árvore de herança (PAH), os valores obtidos foram os mesmos. Estes resultados podem ser explicados primeiramente devido ao tamanho do sistema e ainda devido ao fato do desenvolvimento OO ter se preocupado com a questão do acoplamento.

Com relação à métrica de coesão (FCO), a versão orientada a aspectos apresentou uma significativa melhora. Este resultado reflete principalmente o uso de aspectos como *PeriodicTiming* e *TimingAttributes*, que concentram em si os atributos de tempo que antes ficavam espalhados por diversas classes do sistema, juntamente com seus métodos de manipulação *get* e *set*.

Quanto às métricas de tamanho, a que mede o tamanho do vocabulário apresentou vantagem no projeto OO em relação ao valor obtido na versão OA. Esta versão teve o valor da métrica TV 27% maior do que o medido para a versão OO. Novamente isto pode ser explicado pela pequena dimensão do sistema. Uma vez que o sistema fosse maior e mais complexo, ele apresentaria mais classes específicas para o atendimento de suas funcionalidades, e provavelmente classes adicionais para tratar requisitos não-funcionais. A utilização de aspectos tornaria desnecessário o uso de tais classes, diminuindo o tamanho do sistema. Já a métrica que mede o tamanho do vocabulário interno dos componentes do sistema (NA) apresentou melhora de quase 50%. Este resultado reflete a concentração de atributos usados para tratar requisitos não-funcionais que antes (na versão OO) se encontravam espalhados e replicados em diversas classes do sistema, e na versão OA encontram-se reunidos nos aspectos.

Um resumo das informações apresentadas nas Tabelas 6.1 e 6.2 é representado graficamente na Figura 6.2.

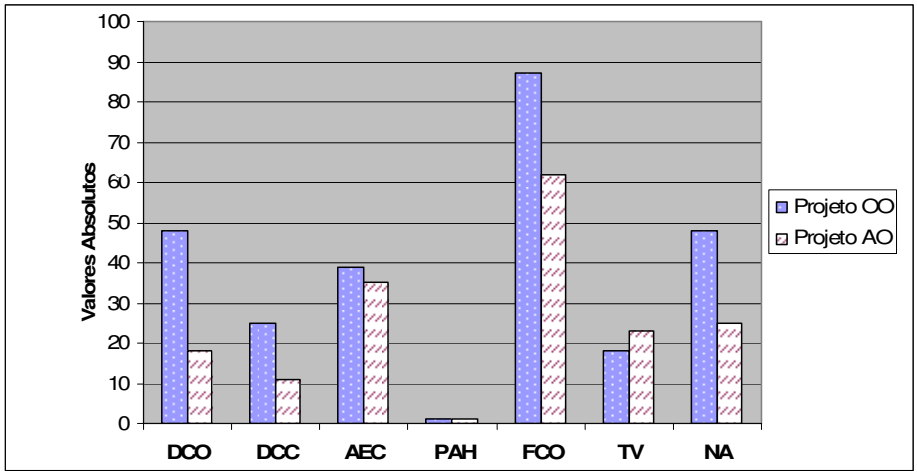

Figura 6.2: Apresentação dos Resultados da Aplicação das Métricas para o Sistema da Cadeira de Rodas

De posse dos valores das métricas, é possível realizar a análise comparativa baseada no modelo de qualidade apresentado na subseção 6.1.

Claramente houve uma melhora substancial na separação de conceitos (cerca de 57% considerando as duas métricas DCC e DCO), o que contribui tanto para a compreensão quanto para a flexibilidade do sistema. Quanto ao tamanho, houve uma melhora não tão expressiva, de aproximadamente 13%. Importante observar que este resultado teve contribuição negativa do tamanho do vocabulário (TV) do sistema, conforme já comentado na análise individual das métricas. O atributo interno do acoplamento foi o que apresentou menor melhora apenas 10%. Já a coesão apresentou uma melhora de 28% no projeto OA.

Desta forma, observa-se que todos os atributos internos contribuíram positivamente para a melhora dos fatores compreensão e flexibilidade. Considerando pesos iguais aos quatro atributos, tem-se uma melhora de aproximadamente 27% para o fator compreensão e 31% para o fator flexibilidade. Como tanto o reuso quanto a manutenibilidade são compostos destes fatores, pode-se dizer que o uso de orientação a aspectos melhorou em torno de 29% estas qualidades no projeto do sistema de controle de movimento da cadeira de rodas.

6.2.2 Veículo Aéreo Não-Tripulado

Os resultados obtidos para o estudo de caso do VANT não apresentaram grandes diferenças se comparados com os resultados analisados para o sistema de controle da cadeira de rodas. Mesmo assim, é interessante ressaltar que houve uma ligeira acentuação dos benefícios oriundos da utilização da metodologia orientada a aspectos em virtude o sistema apresentar um número maior de elementos estruturais funcionais (classes). Certamente esta diferença não é muito expressiva, devido ao fato de ambos os sistemas apresentarem porte semelhante, porém ela aponta a direção de que os benefícios da utilização de orientação a aspectos se acentuam com o aumento do sistema.

O resumo dos resultados da aplicação das métricas de separação de conceitos encontra-se na Tabela 6.3.

Tabela 6.3: Resultados das Métricas de Separação de Conceitos Aplicadas ao Projeto do Sistema de Controle do Helicóptero Não-tripulado

Preocupação	**Tempo**		**Distribuição**		**Embarcados**		**TOTAL**	
Métricas	**DCC**	**DCO**	**DCC**	**DCO**	**DCC**	**DCO**	**DCC**	**DCO**
Projeto OO	11	33	12	27	9	12	**32**	**72**
Projeto OA	7	8	4	6	2	3	**13**	**17**

Se comparado ao projeto da cadeira de rodas, o projeto do sistema de controle do helicóptero obteve um maior benefício no que se refere à separação de preocupações. Isto se dá principalmente devido ao fato deste sistema ter um maior número de elementos que sofrem influência de requisitos não-funcionais. Desta forma, o uso de aspectos proporcionou uma concentração mais acentuada destas preocupações em entidades específicas, deixando às classes somente o tratamento de requisitos funcionais.

A Tabela 6.4 apresenta os resultados obtidos da aplicação das métricas de acoplamento, coesão e tamanho ao estudo de caso do VANT.

Tabela 6.4: Resultados das Métricas de Acoplamento, Coesão e Tamanho Aplicadas ao Projeto do Sistema de Controle do Helicóptero Não-tripulado

Atributos Internos	**Acoplamento**		**Coesão**	**Tamanho**	
Métricas	**AEC**	**PAH**	**FCO**	**TV**	**NA**
Projeto OO	42	2	137	32	54
Projeto AO	37	2	91	33	30

Assim como no projeto da cadeira de rodas, as métricas de acoplamento não apresentaram melhora muito expressiva. Repetem-se neste caso os mesmos comentários apresentados para o estudo de caso da cadeira de rodas quanto ao tamanho do sistema e a preocupação do projeto OO com o acoplamento. Deve-se destacar, porém, que no

projeto OO do VANT, houve outras hierarquias de classes, mas todas com tamanho igual ou inferior a dois, o que não é identificado pela métrica PAH, que conta apenas a maior árvore presente no projeto do sistema.

Com relação à coesão, houve novamente um melhora expressiva (em torno de 66%) no resultado apresentado no projeto orientado a aspectos. Isto se deu não só pelo do uso dos aspectos TimingAtributes e PeriodicTiming, mas também pelo uso do aspecto de adaptação de atributos temporais, o TimeParameterAdapter, que possibilitou a remoção de outros métodos de manipulação dos atributos de validade temporal (nas classes MovementInformation e EnviromentInformation) e período (na classe MovementControl).

Os resultados obtidos com as métricas de tamanho do sistema de controle do VANT corroboram os comentários realizados na análise destas métricas para o sistema de controle da cadeira de rodas. Naquela análise foi dito que num sistema com maior número de classes, a parte de contribuição dos aspectos à métrica TV seria proporcionalmente menor. De fato no sistema de controle do VANT isto se aconteceu. A métrica TV teve uma contribuição de 12 aspectos para este sistema com 21 classes, enquanto sua versão OO possuía 32 classes. No sistema da cadeira de rodas foram 11 aspectos para 12 classes na versão OA contra 18 classes na versão OO. Isto indica que quanto maior o sistema, menor é a influência do número de aspectos no seu tamanho total. Quanto à métrica NA, ocorreu também neste estudo de caso uma diminuição proporcionada pela concentração de atributos não-funcionais em aspectos.

O gráfico apresentado na Figura 6.3 resume os resultados obtidos no estudo de caso do helicóptero não-tripulado.

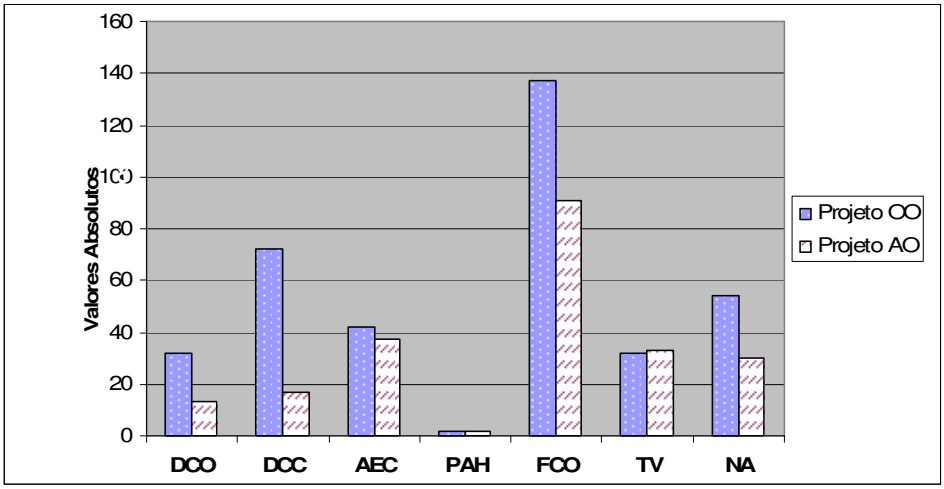

Figura 6.3: Apresentação dos Resultados da Aplicação das Métricas para o Sistema de Controle de Movimento do Helicóptero Não-tripulado

Buscando mensurar as qualidades de reuso e manutenibilidade do sistema, aplica-se o modelo de qualidade apresentado na subseção 6.1 com base nos resultados das métricas auferidos. A separação de conceitos melhorou aproximadamente 68% no projeto orientado a aspectos em relação ao projeto orientado a objetos. Já o atributo interno tamanho teve melhora de 21%, enquanto a coesão melhorou 34%. O acoplamento apresentou melhora, mas não muito expressiva ficando na casa de 12%.

Calculando a melhora na compreensão, obtêm-se o valor de 34% e quanto à melhora da flexibilidade, atinge-se o valor de 38%. Com isto tem-se uma melhora de aproximadamente 36% no valor final das qualidades de reuso e manutenibilidade do sistema de controle de movimento do helicóptero não-tripulado.

7 CONCLUSÕES E TRABALHOS FUTUROS

A presente dissertação se propôs a utilizar os conceitos de orientação a aspectos para analise e projeto de STrED. Esta proposta se consubstanciou através da metodologia RT-FRIDA, que trata requisitos funcionais e não-funcionais do domínio de interesse separadamente, desde o início do desenvolvimento do sistema. Esta metodologia foi baseada principalmente no método FRIDA, desenvolvido na tese de doutorado de Sílvia Bertagnolli (BERTAGNOLLI, 2004). Adicionalmente foi realizada uma avaliação quantitativa objetivando mensurar a contribuição do uso de aspectos no projeto de STrED.

A adoção de uma metodologia com preocupação de separar requisitos funcionais dos não-funcionais desde a fase de análise visou apresentar um projeto de sistema que tivesse elementos funcionais e não-funcionais pensados desde o início do projeto, e não a criação de elementos não-funcionais a partir da refatoração de um sistema orientado a objetos, como o trabalho apresentado em (TSANG et al, 2004). Este enfoque possibilitou mostrar como é possível (através dos artefatos utilizados na metodologia) manter o elo entre requisitos e projeto, de modo a prover a rastreabilidade que facilita o reuso e a manutenção dos sistemas.

Norteado por tais idéias, traçou-se como objetivo adaptar a metodologia FRIDA para o contexto dos sistemas tempo-real distribuídos e embarcados. Esta adaptação demandou inicialmente o levantamento de um conjunto mínimo de requisitos a serem tratados pela metodologia. Em seguida foi realizada a adaptação ao contexto dos artefatos utilizados para a elicitação dos requisitos. Finda esta parte, passou-se ao aprimoramento da fase de projeto, com a construção de uma biblioteca de aspectos de alto nível capaz de atender os requisitos específicos do domínio de interesse. Além disto, deve-se destacar o uso do perfil RT-UML em conjunto com aspectos e ainda a introdução de outras formas de representação de elementos importantes ao projeto, como o diagrama JPDD, e o diagrama ACOD. Porém a ligação entre os requisitos e o projeto ainda carecia de um elo mais forte. Com isto foi inserida uma fase intermediária no processo que além de servir como ferramenta de documentação que provê rastreabilidade, ainda auxilia o desenvolvedor a vislumbrar a utilização de elementos da biblioteca ou reusar elementos de outros projetos. Esta ferramenta, resumida numa tabela de mapeamento, reúne toda informação necessária para se compreender que elementos do sistema atendem determinados requisitos e quais funcionalidades do sistema são entrecortadas por propriedades não-funcionais.

O uso da RT-FRIDA proporcionou um projeto orientado a aspectos de STrED com preocupação não-funcional desde o seu início. Com isto foi possível avaliar a contribuição do projeto orientado a aspectos no domínio de interesse. Para realizar esta avaliação foi utilizado um modelo de qualidade que tem por objetivo mensurar as

qualidades de manutenibilidade e reuso. Este modelo, baseado em (SANT'ANNA et al 2003), aprecia diversos atributos mensurados através de um conjunto de métricas que acabam por culminar nas qualidades finais alvo da avaliação. Nesta dissertação dois projetos foram avaliados por este modelo, e o resultado de ambos sinalizou a melhora das qualidades finais do sistema através do uso da orientação a aspectos. Apesar do pequeno porte de ambos os sistemas analisados, os resultados mostram que houve ganho em diversas métricas mensuradas. O fraco desempenho de algumas métricas, como ocorreu com o tamanho, não invalida o resultado positivo obtido por outras métricas, uma vez que este resultado tente a melhorar com o crescimento do sistema.

Portanto, deve-se destacar que de fato o uso de aspectos no projeto do sistema com o suporte de uma metodologia de projeto preocupada com os requisitos não-funcionais desde a análise do problema, contribuiu para a obtenção de um sistema mais flexível e compreensível, o que resultou num sistema mais fácil de sofrer alterações e manutenção e com componentes mais facilmente reutilizáveis.

Finalmente é importante que se apresentem os caminhos da continuação do presente trabalho. Uma extensão natural seria a automação da metodologia, através de uma ferramenta CASE que forneça suporte ao desenvolvedor no uso dos artefatos desenvolvidos. Outro ponto que pode ser continuamente melhorado é o próprio conteúdo dos artefatos, como a introdução de novas entradas no conjunto de léxicos e novas questões nas *checklists*. O aprimoramento e extensão da biblioteca de aspectos de alto nível também se apresentam como interessantes prolongamentos do presente estudo. Quanto aos critérios de avaliação, pode-se vislumbrar a busca de novas métricas que possibilitem melhor valoração dos benefícios do uso de elementos da orientação a aspectos para o tratamento de requisitos não-funcionais. Pode-se ainda citar a implementação dos aspectos descritos em alto nível pela DERAF como um trabalho que pode ser desenvolvido na seqüência do estudo apresentado nesta dissertação.

REFERÊNCIAS

AKSIT, M. Composition and Separation of concerns in the object-oriented model. **ACM Computing Surveys**, New York, v. 28, n.4, 1996.

AMBLER, S. W. **The Object Primer - Agile Model Driven Development with UML 2**. 3rd ed. [S.l.]: Cambridge University Press, 2004.

AOSD-Europe. **Survey of Analysis and Design Approches**. 2005. Disponível em: <http://www.aosd-europe.net/documents/analys.pdf> Acesso em: jun. 2006.

ARAÚJO, J. et al. Aspect-Oriented Requirements with UML. In: WORKSHOP ON ASPECT-ORIENTED MODELING WITH UML, UML, 2002, Dresden, Germany. **Proceedings...** [S.l.: s.n.], 2002.

AXELSSON, J. Real-World Modeling in UML. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEMS ENGINEERING AND THEIR APPLICATIONS, ICSSEA, 13., 2000. **Proceedings…** [S.l.:s.n.], 2000.

BACKUS, J. et al. Revised Report on the algorithmic Language Algol 60. In: ROSEN, S. (Ed.). **Programming Systems and Languages**. New York: McGraw-Hill, 1969. p. 79-117.

BECKER, L. B. **Um Método para Abordar todo o Ciclo de Desenvolvimento de Aplicações Tempo Real**. 2003. 136p. Tese (Doutorado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BEIER, G.; KERN, M. Aspects in UML Models from a Code Generation Perspective. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE - THE LANGUAGE AND ITS APPLICATIONS, 5., 2002. **Proceedings…** [S.l.:s.n.], 2002.

BERTAGNOLLI, S. C. **FRIDA:** um método para elicitação e modelagem de RNFs. 2004. 153p. Tese (Doutorado em Ciências da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BOEHM, B. **Characteristics of Software Quality**. Amsterdam: North Holland Press, 1978.

BRITO, I.; MOREIRA, A.; ARAÚJO, J. A requirements model for quality attributes. In: ASPECT-ORIENTED REQUIREMENTS ENGINEERING AND ARCHITECTURE DESIGN, AOSD, 1., 2002. **Proceedings…** New York: ACM, 2002.

BURNS, A.; WELLINGS, A. **Real-time systems and programming languages**. 2nd ed. Harlow: Addison-Wesley, c1997. 611 p.

BURNS, A. et al. The meaning and role of value in scheduling flexible real-time systems. **Journal of Systems Architecture**, Amsterdam, v. 46, p. 305-325, 2000.

CARRO, L.; WAGNER, F. R. Sistemas Computacionais Embarcados. In: JORNADAS DE ATUALIZAÇÃO EM INFORMÁTICA, 22., 2003, Campinas. **Livro Texto**. Campinas: SBC, 2003. p. 45-94.

CAZZOLA, W.; SOSIO, R.; TISATO, F. Shifting Up Refelction from the Implementation to the Analysis Level. In: CAZZOLA, W.; SOSIO, R.; TISATO, F. (Ed.). **Reflection and Software Engineering**. Berlin: Springer, 2000. (Lecture Notes in Computer Science 1826).

CHIDAMBER, S.R.; KEMERER, C.F. A Metrics Suite for Object-Oriented Design. **IEEE Transactions on Software Engineering**, New York, v. 20, n. 6, p. 476-493, 1994.

CHUNG, L.; NIXON A. Dealing with Non- Functional Requirements: Three Experimental Studies of a Process-Oriented Approach. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ICSE, 17., 1995. **Proceedings…** New York: ACM, 1995. p. 25 – 37.

CLARKE, S.; WALKER, R. J. Towards a Standard Design Language for AOSD In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 1., 2002. **Proceedings…** New York: ACM, 2002.

DETERS, M.; LEIDENFROST, N.; CYTRON, R. K. Translation of Java to Real-Time Java Using Aspects. In: INTERNATIONAL WORKSHOP ON ASPECT-ORIENTED PROGRAMMING AND SEPARATION OF CONCERNS, 2001. **Proceedings…** [S.l.:s.n.], 2001.

DIAZ-HERRERA, J. L.; CHADHA, J.; PITTSLEY, N. Aspect-Oriented UML Modeling for Developing Embedded Systems Product Lines. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 1., 2002. **Proceedings…** New York: ACM, 2002.

DIBBLE, P.C. **Real-time Java Plataform Programming**. Palo Alto, CA: Sun Microsystems Press, 2002.

DEPARTAMENT OF DEFENSE. Unmanned Aircraft Systems Roadmap 2005-2030. Technical Report. Disponível em: < http://www.acq.osd.mil/usd/Roadmap%20Final2.pdf > Acesso em: set. 2006.

DOUGLASS, B. P. **Real-Time UML:** Developing Efficient Objects for Embedded Systems. Harlow: Addison-Wesley, 1999.

FREITAS, E.P. **Análise do Tratamento de Requisitos Não-Funcionais em Metodologias Orientadas a Objetos e Orientadas a Aspectos no Projeto de Sistema Tempo Real**. 2006. 69p. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

GAL, A.; SPINCZYK, O.; PREIKSCHAT, W. S. On Aspect-Orientation in Distributed Real-time Dependable Systems. In: INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, WORDS, 7., 2002. **Proceedings…** San Diego: IEEE, 2002.

GIIL, C. D. A Vision for Integration of Embedded System Properties Via a Model-Component-Aspect System Architecture. In: WORKSHOP ON SOFTWARE

ENGINEERING FOR EMBEDDED SYSTEMS: FROM REQUIREMENTS TO IMPLEMENTATION, 2003. **Proceedings…** [S.l.: s.n.], 2003. p. 24-26.

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Reading, MA: Addilson-Wesley, 1995.

GOSLING, J.; JOY, B.; STEELE, G. **Java Language Specification**. Reading, MA: Addison-Wesley, 1996.

GRAY, J.; BAPTY, T.; NEEMA, S. Aspectifying Constraints in Model-Integrated Computing. In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS; WORKSHOP ON ADVANCED SEPARATION OF CONCERNS, OOPSLA, 2000. **Proceedings…** New York: ACM, 2000.

GRUNDY, J.; PATEL, R. Developing Software Components with the UML, Enterprise Java Beans and Aspects. In: AUSTRALIAN SOFTWARE ENGINEERING CONFERENCE, 2001. **Proceedings…** [S.l.]: IEEE, 2001.

GYRON. **Sistema Helix**. 1998. Disponível em: <ftp://ftp.gyron.com.br/pub/doc/info/helix-desc-ptg.pdf> Acesso em: jul. 2006.

HARRISON, W.; OSSHER, H. Subject-Oriented programming (A critique of pure objects). **ACM SIGPLAN Noticies**, New York, v. 28, n. 10, 1993, p. 411-428, 1993.

HENZINGER, T. A.; KIRSCH, C. M.; HOROWITZ, B. Giotto: A time-triggered language for embedded programming. **Proceedings of the IEEE**, New York, v.91, n.1, p.84-99, Jan. 2003.

HO, W. et al. UMLAUT: an extendible UML transformation framework. In: AUTOMATED SOFTWARE ENGINEERING, ASE, 1999. **Proceedings…** [S.l.]: IEEE, 1999.

HUNT, J. **Real-time Java**. 1998. Disponível em: <www.planetjava.co.uk> Acesso em: maio 2006.

IEEE. **Glossário de termos técnicos online da IEEE**. Disponível em: <http://standards.ieee.org/catalog/olis/arch_se.html> Acesso em: mar. 2006.

ITO, S. A. **Projeto de Aplicações Específicas com Microcontroladores Java Dedicados**. 2000. 84 f. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

JACOBSON, I.; BOOCH, G.; and RUMBAUGH, J. **The Unified Software Development Process**. Reading, MA: Addison-Wesley, 1999.

JONG, G. A UML-Based Design Methodology for Real-Time and Embedded Sytems. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2002. **Proceedings…** [S.l.:s.n], 2002.

KATARA, M.; MIKKONEN, T. Aspect-Oriented Specification Architectures for Distributed Real-Time Systems. In: INTERNATIONAL CONFERENCE ON ENGINEERING OF COMPLEX COMPUTER SYSTEMS, ICECCS, 7., 2001. **Proceedings…** [S.l.:s.n], 2001.

KHAN, S. S.; JAFFAR-UR- REHMAN, M. A Survey on Early Separation of Concerns. In: ASIA-PACIFIC SOFTWARE ENGINEERING CONFERENCE, 12., 2005, **Proceedings…** [S.l.]: IEEE, 2005.

KICZALES, G. et al. Aspect-Oriented Programming. In: EUROPEAN CONFERENCE FOR OBJECT-ORIENTED PROGRAMMING, ECOOP, 1997. **Proceedings...** Berlin: Springer-Verlag, 1997. p. 220-240.

LAVAZZA, L.; QUARONI, G.; VENTURELLI, M. Combining UML and formal notations for modelling real-time systems. In: EUROPEAN SOFTWARE ENGINEERING CONFERENCE, 8., 2001. **Proceedings…** New York: ACM, 2001.

LEITE, J.C.S.P.; OLIVEIRA, A.P.A. A Client Oriented Requirements Baseline. In: IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING, 2., 1995, York. **Proceedings…** Los Alamitos, CA: IEEE Computer Society Press, 1995. p. 108-115.

LIEBERHERR, K. **Adaptative Object-Oriented Software:** The Demeter Method with Propagation Patterns. Boston: PWS Publishing Company, 1996. Disponivel em: <ftp://ftp.ccs.neu.edu/pub/people/lieber/book/aoos.PDF> Acesso em: out. 2005.

LINDSTRÖM, D. R. Five Ways to Destroy a Development Project. **IEEE Software**, New York, p. 55-58, Sept. 1993.

LSE - LABORATÓRIO DE SISTEMAS EMBARCADOS. **Projeto de Sistemas Eletrônicos Embarcados Baseados em Plataformas**. 2003. Disponível em: <http://www.inf.ufrgs.br/~lse/pag_projeto.php?cod_projeto=1 >. Acesso em: nov. 2006.

MARTIN, G. UML for Embedded Systems Specification and Design: Motivation and Overview. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE AND EXHIBITION, DATE, 2002. **Proceedings…** [S.l.:s.n], 2002.

McCALL, J.; RICHARDS, P.; WALTERS, G. **Factors in Software Quality**. [S.l.]: General Electric Command and Information Systems, 1977.

MYLOPOULOS, J. et al. Exploring Alternatives during Requirements Analysis. **IEEE Software**, New York, p. 2-6, Jan./Feb. 2001.

NIZ, D.; RAJKUMAR, R. Time Weaver : A Software-Through-Models Framework for Embedded Real-Time Systems. In: CONFERENCE ON LANGUAGE, COMPILER, AND TOOL FOR EMBEDDED SYSTEMS, 2003. **Proceedings…** New York: ACM, 2003.

NUSEIBEH, B. Crosscutting Requirements. In: ASPECT ORIENTED SOFTWARE DEVELOPMENT, AOSD, 2004. **Proceedings…** New York: ACM, 2004.

OBJECT MANAGEMENT GROUP. **UML Profile for Schedulability, Performance, and Time Specification.** 2004. Disponível em: <http://www.omg.org/cgi-bin/doc?ptc/04-02-01> Acesso em: jun. 2004.

OBJECT MANAGEMENT GROUP. **UML 2.0 Specification.** 2005. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/05-07-05> Acesso em: jul. 2005.

OSSLER, H.; TARR, P. Using subject-oriented programming to overcome common problems in object-oriented software development/evolution", In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1999. **Proceedings…** New York: IEEE, 1999. p. 687-688.

PEREIRA, C. E. Real-Time Active Objects in C++/Real-Time UNIX. In: WORKSHOP ON LANGUAGES, COMPILER, AND TOOL SUPPORT FOR REAL-TIME SYSTEMS, 1994. **Proceedings…** [S.l.:s.n], 1994.

RASHID, A. et al. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In: JOINT INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING, 2002. **Proceedings…** [S.l.]: IEEE, 2002. p. 199-202.

RASHID, A.; MOREIRA, A.; ARAÚJO, J. Modularisation and Composition of Aspectual Requirements. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 2002. **Proceedings…** New York: ACM, 2003.

ROSENBERG, L.H. **Applying and Interpreting Object Oriented Metrics**. 2003. Disponível em: <http://www.satc.gsfc.nasa.gov> Acesso em: out. 2006.

RTSJ - Real-Time Specification for Java. Disponível em: <http://www.rtsj.org/specjavadoc/book_index.html> Acesso em: set. 2006.

SANT'ANNA, C. N. et al. On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework In. SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 17., 2003, Manaus. **Anais…** Manaus: EDUA, 2003.

SCHAUERHUBER, A. et al. Towards a Common Reference Architecture for Aspect-Oriented Modeling. In: INTERNATIONAL WORKSHOP ON ASPECT-ORIENTED MODELING, 8., 2006, Bonn, Germany. **Proceedings…** [S.l.:s.n.], 2006.

SCHMIDT, C. D.; KLEFSTAD, R.; KRISHNA, A. S. Design and performance of a modular portable object adapter for distributed, real-time, and embedded CORBA applications. In: DISTRIBUTED OBJECTS AND APPLICATIONS, DOA, 2002. **Proceedings…** [S.l.:s.n], 2002. p. 549-562.

SCHMIDT, D.; LEVINE , D.; MUNGEE, S. The Design of the TAO Real-Time Object Request Broker. **Computer Communications**, [S.l.], 1998. Special Issue on Building Quality of Service into Distributed Systems.

SEI. **Glossário de termos técnicos online do Software Engineering Institute – Carnegie Mellon**. Disponivel em: <http://www.sei.cmu.edu/str/indexes/glossary/> Acesso em: mar. 2006.

SEIBEL, C.W. **Uma metodologia Formal para o Planejamento e Controle de Missões de Aeronaves Não-Tripuladas**. 2000. 120f. Tese (Doutorado em Engenharia Elétrica) – UFSC, Florianópolis.

SOUSA, G. et al. Separation of Crosscutting Concerns from Requirements to Design: Adapting a Use Case Driven Approach. In: ASPECT ORIENTED SOFTWARE DEVELOPMENT, AOSD, 2004. **Proceedings…** New York: ACM, 2004.

SELIC, B. The emerging real-time UML Standard. **International Journal of Computer Systems Science Engineering**, Leics, UK, v. 17, n. 2, 2002.

SHLAER, S.; MELLOR, S. J. **Object Lifecicles:** Modeling the World in States. Englewood Cliffs, NJ: Prentice Hall PTR, 1992.

STANKOVIC, J. A. et al. VEST: An Aspect-Based Composition Tool for Real-Time System. In: REAL-TIME AND EMBEDDED TECHNOLOGY AND APPLICATIONS SYMPOSIUM, RTAS, 9., 2003. **Proceedings…** Washington: IEEE, 2003.

STANKOVIC, J. A. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. **Computer**, New York, p. 10-19, Oct. 1988.

STEIN, D.; HANENBERG, S.; UNLAND, R. Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 5., 2006. **Proceedings…** Bonn: ACM, 2006. p.15-26.

STEIN, D.; HANENBERG, S.; UNLAND, R. A UML-based Aspect-Oriented Design Notation for AspectJ. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, 1., 2002. **Proceedings…** New York: ACM, 2002. p. 106-112.

SUN MICROSYSTEMS. **picoJava II Microarchitecture Guide**. [S.l.], 1999.

SZYPERSKI, C. **Component Software Beyond Object-Oriented Programming**. New York: Addison-Wesley: ACM Press, 1998.

TANENBAUM, A. S. **Distributed systems principles and paradigms**. Upper Saddle River: Prentice-Hall, c2002. 803 p.

TESANOVIC, A. et al. Aspects and Components in Real-Time System Development: Towards Reconfigurable and Reusable Software. **Journal of Embedded Computing**, Cambridge, v. 1, n. 1, Oct. 2004.

TSANG, S. L.; CLARKE, S.; BANIASSAD, E. An Evaluation of Aspect-Oriented Programming for Java-based Real-Time Systems Development. In: INTERNATIONAL SYMPOSIUM ON OBJECT-ORIENTED REAL-TIME DISTRIBUTED COMPUTING, ISORC, 7., 2004. **Proceedings…** [S.l.]: IEEE, 2004.

WAGELAAR, D. A concept-based approach for early aspect modelling. In: TECKINERDOGAN, B. **Early Aspects:** Aspect-Oriented Requirements Engineering and Architecture Design. [S.l.]: Telematics and Information Technology, University of Twente, 2003.

WEHRMEISTER, M. A. **Framework Orientado a Objetos para Projeto de Hardware e Software Embarcados para Sistemas Tempo-Real**. 2005. 104p. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

WEHRMEISTER, M. A. **Framework orientado a aspectos para sistemas tempo-real embarcados e distribuídos**. 2006. 64p. Proposta de Tese (Doutorado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

WOLF, W. H. **Computers as Components:** Principles of Embeded Computing System Design. San Francisco: Morgan Kaufmann Publishers, 2000.

YEH, R. et al. Software Requirements: New Directions and Perspectives. In: VICK, C.R.; RAMAMOORTHY, C.V. (Ed.). **Handbook of Software Engineering.** New York: Van Nostrand Reinhold, 1984. p. 519-543.

ZHANG L.; LIU, R. Aspect-Oriented Real-Time System Modeling Method Based on UML. In: INTERNATIONAL CONFERENCE ON EMBEDDED AND REAL-TIME COMPUTING SYSTEMS AND APPLICATIONS, RTCSA, 11., 2005. **Proceedings…** [S.l.]: IEEE, 2005.

ANEXO A REGRAS PARA DETERMINAÇÃO DE CONFLITOS

As regras descritas na Figura A definem se dois requisitos encontram-se em conflito quando alterações em um deles causam impacto no outro. Por exemplo, quanto mais justos os requisitos temporais, mais recursos de processamento para cumpri-los.

Segundo este conjunto de regras, a determinação da existência de um conflito é feita através da aplicação seqüencial das quatro regras apresentadas. Inicialmente determina-se a existência de uma alteração. Isto é feito através da aplicação da regra número 1. Em seguida, aplica-se a regra de número 2 para determinar-se a existência da alteração em um requisito. O passo seguinte tem por objeto descobrir se a alteração de um requisito gera impacto em outro. A regra de número três define que caso a alteração em requisito gere alteração em outro, o primeiro impacta neste. E por fim diz-se que dois requisitos conflitam quando as alterações em um geram impacto no outro, como está descrito na regra de número quatro.

(1) alteração : $V \rightarrow V$
alteração $(x) = x + \varepsilon$
onde: V é o conjunto de valores,
$x \in V$ e $x \neq \perp$,
$\varepsilon \in V$ e $x \neq \perp$,
$<v, +, \perp>$ é um grupo

(2) r_i altera $\Leftrightarrow$ alteração (x_{ri})
onde: r_i é o requisito i,
x_{ri} é o valor agregado
alteração é definida em (1)

(3) r_i impacta $r_j \Leftrightarrow ((r_i \text{ altera}) \rightarrow (r_j \text{ altera}))$
onde: $r_i \neq r_j$,
altera é definida em (2)

(4) r_i conflita $r_j \Leftrightarrow r_i$ impacta r_j
onde: $r_i \neq r_j$,
impacta é definida em (3)

Figura A: Regras para determinação da existência de conflitos entre requisitos distintos de um sistema

APÊNDICE A CONJUNTO DE TEMPLATES FUNCIONAIS: CADEIRA DE RODAS AUTOMATIZADA

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-1
		Nome	Change Movement Mode
		Objetivo	Realizar a mudança de modo de operação do controle de movimento da cadeira.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	O controle de navegação deve ter um modo de operação selecionado.
		Pós-condição	O sistema deve atuar de acordo com o novo modo de operação selecionado.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Média
		Situação	Finalizado
Caminhos		**Primário (Normal)**	O novo modo de operação é informado ao sistema e este realiza o chaveamento para operar de acordo com este novo modo.
		Alternativo	
		Excepcional	O novo modo informado é o mesmo do anteriormente programado, desta maneira o sistema não necessita realizar nenhuma alteração, apenas confirmar ao usuário que a opção selecionada já se encontra em operação.
Cenários		**Principal**	(1) O Sistema de Navegação recebe uma ordem de alteração no modo de operação. (2) A alteração é propagada aos demais componentes que controlam o movimento. (3) O Sistema de Navegação informa o status do modo de operação.
		Variações	(2) No caso da alteração não ser possível (modo anterior igual ao novo) o sistema não tem alterações a propagar. (3.1) Além do status do modo de operação, o sistema retorna um aviso de que o modo selecionado já é o que está em operação.

Figura A.1: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	Identificação	Identificador	FR-2
		Nome	Wheelchair Movement Control
		Objetivo	Realiza o controle do movimento da cadeira de rodas.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	Os dados dos sensores e joystick devem estar disponíveis.
		Pós-condição	Os parâmetros de moviemento da cadeira devem ser atualizados.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	Prioridade	Crítica
		Situação	Finalizado
Caminhos		Primário (Normal)	O sistema deve calcular periodicamente os novos parâmetros de moviemento com base nos dados fornecidos pelos sensores e joystick.
		Alternativo	
		Excepcional	
Cenários		Principal	(1) O sistema faz acessa os dados fornecidos pela leitura dos sensores de movimento e joystick. (2) O sistema realiza o cálculo dos novos parâmetros de moviemento. (3) Os novos parâmetros são então disponibilizados para serem aplicados ao controle dos atuadores de movimento.
		Variações	(2) No caso de mal funcionamento de algum elemento do sistema, os parâmetros devem ser calculados de acordo com uma política que garanta a segurança do movimento da cadeira.

Figura A.2: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	Identificação	Identificador	FR-3
		Nome	Movement Actuation
		Objetivo	Realiza o controle direto sobre os atuadores de movimento da cadeira de rodas.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	Os parâmetros de moviemento da cadeira devem estar atualizados.
		Pós-condição	Os atuadores devem realizar o movimento de acordo com os parâmetros estabelecidos.
		Ator primário	Navigation Control
		Ator secundário	Movement Actuator
	Decisão e Evolução	Prioridade	Alta
		Situação	Finalizado
Caminhos		Primário (Normal)	O sistema deve transmitir aos atuadores os parâmetros de movimento.
		Alternativo	
		Excepcional	
Cenários		Principal	(1) Dentro do ciclo do controle de movimento, o sistema deve realizar o controle direto sobre os atuadores de modo a efetivar os movimentos da cadeira.
		Variações	

Figura A.3: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-4
		Nome	Movement Control Malfunctional Corrective Action
		Objetivo	Realiza ações de correção em caso de mal funcionamento de elementos do sistema.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Detecção de mal funcionamento em algum elemento do sistema.
		Pós-condição	Funcionamento correto do sistema mesmo em presença de falhas.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Na presença de elementos que não estejam funcionando adequadamente, o sistema deve reagir de modo a garantir o resultado final correto do controle do movimento. Deve-se determinar o acionamento do alarme enquanto a condição de erro persistir.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) O sistema é informado de um mau funcionamento interno. (2) Uma medida corretiva deve ser tomada para eliminar possíveis efeitos negativos deste problema. (3) O alarme de alerta de mau funcionamento deve ser desligado caso o problema seja totalmente resolvido pela ação corretiva.
		Variações	(3) Caso a ação corretiva resolva parcialmente o problema, o alarme é mantido acionado.

Figura A.4: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-5
		Nome	Joystick Sensing
		Objetivo	Realiza a amostragem dos valores de posição do Joystick.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Mudança da posição do joystick.
		Pós-condição	Disponibilização dos valores amostrados.
		Ator primário	Joystick
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	A mudança de posição do Joystick é capturada periodicamente e a amostragem deste valor é disponibilizada ao controle de navegação.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) A posição do joystick é lida ciclicamente pelo sistema. (2) Mudanças no valor da posição do joystick são capturadas e digitalizadas. (3) O valor resultante da amostragem é disponibilizado ao sistema.
		Variações	

Figura A.5: *Template* do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	Identificação	Identificador	FR-6
		Nome	Movement Sensing
		Objetivo	Realiza a amostragem dos valores dos sensores de movimento da cadeira.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	
		Pós-condição	Disponibilização da informação de movimento da cadeira.
		Ator primário	Sensores de velocidade e ângulo das rodas.
		Ator secundário	
	Decisão e Evolução	Prioridade	Alta
		Situação	Finalizado
Caminhos		Primário (Normal)	As informações de ângulo de movimento e velocidade da cadeira são lidas pelos sistema.
		Alternativo	Caso a cadeira esteja parada, o sistema aguarda o valor da velocidade ser diferente de nulo para realizar o acompanhamento da evolução do movimento.
		Excepcional	
Cenários		Principal	(1) Ciclicamente o sistema recupera os valores lidos pelos sensores de velocidade e ângulo das rodas.(2) Os valores são digitalizados. (3) O resultado é disponibilizado para o sistema.
		Variações	

Figura A.6: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-7
		Nome	Signal alarm
		Objetivo	Controlar o acionamento do alarme.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Requisição de acionamento ou desligament do alarme em função de algum evento ocorrido n sistema.
		Pós-condição	Alarme acionado ou desligado conforme a demanda.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Média
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Ao detectar alguma falha no sistema ou a corrigir uma falha, o controle de ma funcionamento faz a requisição de acionament ou desligamento do alarme.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) Recebimento de requisição de acionamento ou desligamento do alarme. (2) Mudança do estado do alarme para acionado ou desligado.
		Variações	

Figura A.7: Template do FR1 do Sistema de Controle de Movimento da Cadeira de Rodas

APÊNDICE B CHECKLISTS : CADEIRA DE RODAS AUTOMATIZADA

	Relevante	Prioridade	Restrições/ Condições/ Descrição
Tempo			
Temporização			
Existem atividades ou amostragens periódicas?	**X**	**Alta**	Leitura dos dados do Joystick (1000 amostras por segundo); Controle do movimento da cadeira (20 milisegundos); Leitura dos dados de movimento da cadeira (1000 amostras por segundo).
Existem atividades esporádicas?			
Existem atividades aperiódicas?	**X**	**Alta**	Acionamento de alarme
Existe restrição quanto a latência para se efetivar o início da execução de alguma atividade no sistema?	**X**	**Alta**	Acionamento do alarme (2 milisegundos).
Existem instantes específicos de início/fim para execução de atividades do sistema?			
Foi especificado algum tempo de pior caso para a execução de atividades do sistema? (ou ao menos existe preocupação com relação a esta propriedade?)	**X**	**Alta**	Leitura dos dados do Joystick (2 milisegundos); Controle do movimento da cadeira (3 milisegundos); Leitura dos dados de movimento da cadeira (7 milisegundos).
Precisão			
Existem atividades com flexibilidade no atendimento de seus requisitos temporais?	**X**	**Alta**	Não há flexibilidade.
Caso exista flexibilidade no atendimento de requisitos temporais, o sistema suporta retardo em alguma atividade temporizada?			

O sistema suporta variações no atendimento de requisitos temporais?			
Em uma situação de uso degradado do sistema, existe a possibilidade de se utilizar dados antigos?			
Existe a necessidade de algum tipo de controle sobre a validade dos dados utilizados em algum processo do sistema?	**X**	**Alta**	Os dados de movimento têm um prazo (15 milisegundos) para serem utilizados pelo sistema de controle, caso este prazo expire, estes dados são considerados inválidos.
Existe limite quanto a diferença entre o tempo lógico utilizado pelo sistema e o tempo físico gerado por componentes do sistema?			

Figura B.1: Checklist para Requisitos de Tempo do Sistema de Controle de Movimento da Cadeira de Rodas

Desempenho	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Vazão			
Existe limite quanto ao número de atividades que o sistema pode executar?			
Existe alguma restrição quanto à captura ou armazenamento de dados?			
Existem pontos de convergência de dados?	**X**	**Média**	As variáveis de controle são transmitidas para o nodo resposável pelo controle da cadeira.
Existem restrições importantes quanto à utilização da banda no sistema?			
Tempo de Resposta			
Existem limitações temporais para o retorno de respostas finais do sistema?	**X**	**Alta**	A interatividade do sistema deve ser preservada. Existe um limite definido para o retorno de respostas finais do sistema (0,5 segundo).
Em caso de desempenho degradado (sobrecarga do sistema), existem respostas finais do sistema que podem ser penalizadas em detrimento de outras?			
Existe a possibilidade de se distribuir tarefas para garantir o tempo de resposta de uma atividade do sistema?			

Figura B.2: Checklist para Requisitos de Tempo do Sistema de Controle de Movimento da Cadeira de Rodas

Distribuição	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Alocação de Tarefas			
Existem critérios espaciais que determinem à distribuição das tarefas do sistema?	**X**	**Baixa**	Proximidade dos sensores.
Existindo pontos de convergência de dados, há possibilidade de redistribuição do tratamento destes dados?			
Estações Participantes			
As Estações Participantes tem processamento dedicado a uma determinada atividade?			
Em caso de sobrecarga do sistema, alguma estação pode substituir ou auxiliar no processamento de outra?			
Existe restrição quanto a capacidade de processamento das Estações Participantes?			
Existe restrição espacial que empeça a instalação de alguma estação em algum ponto de atuação ou leitura do sistema?			
Comunicação			
Existe restrição quanto ao uso de alguma tecnologia de comunicação?			
A comunicação exige garantia de entrega de mensagem?	**X**	**Média**	Deve-se garantir a entrega de mensagens contendo os dados lidos pelos sensores e joystick.
Existe restrição quanto ao tamanho dos dados transmitidos ou recebidos?			
Existe diferença (tamanho, prioridade, ...) entre tráfego de controle e de dados?			
É necessário tratamento que garanta a integridade dos dados que trafegam pelo sistema?	**X**	**Média**	Embora não seja uma preocupação crítica, existe a possibilidade de ocorrer problemas quanto a integridade dos dados que trafegam pelo sistema. Com isto, é desejável a existem de um controle sobre esta propriedade.
Sincronização			
Existem recursos ou dados compartilhados?	**X**	**Alta**	Os dados lidos pelos sensores e pelo joystick.
É necessário controle de concorrência sobre os dados compartilhados?	**X**	**Alta**	Os dados compartilhados são críticos para o sistema, de forma que o controle sobre o seu acesso

			é uma questão importante.
Existe alguma política pré-estabelecida para acesso de recursos ou dados compartilhados?			
Existe hierarquia quanto ao acesso aos recursos ou dados compartilhados?			

Figura B.3: Checklist para Requisitos de Distribuição do Sistema de Controle de Movimento da Cadeira de Rodas

Embarcados	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Área			
Existe restrição quanto à área (em silício ou placa de circuito integrado) ocupada por algum componente do sistema?			
Consumo de Potência			
Existe restrição quanto ao consumo de potência de algum elemento do sistema?			
Quanto ao consumo do sistema, existe necessidade de se inserir monitoramento ou controle?	**X**	**Média**	É importante que o sistema monitore suas atividades e eventualmente atue sobre o consumo de forma a reduzi-lo, aumentando assim a duração da fonte de energia.
Energia total			
Existe restrição quanto à energia disponibilizada ao sistema?	**X**	**Média**	A energia total disponibilizada para o sistema se restringe aquela armazenada em sua bateria.
Existe estimativa de quanto deve durar a energia disponibilizada ao sistema?			
Existem fontes alternativas de energia no sistema?			
Existe restrição referente a geração de calor pelo sistema (silhueta térmica)?			
Memória			
Existe restrição quanto ao uso da memória de armazenamento do sistema? (seja para dados ou programas)			
Existe limitação quanto ao uso da memória de execução no sistema?			

Figura B.4: *Checklist* para Requisitos de Embarcados do Sistema de Controle de Movimento da Cadeira de Rodas

APÊNDICE C TEMPLATES DE REQUISITOS NÃO-FUNCIONAIS : CADEIRA DE RODAS AUTOMATIZADA

	Item	
Identificação	**Identificador**	NFR-1
	Nome	Leitura Periodica de Dados
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Período
	Descrição	O sistema deve ser capaz de ler os dados relavantes ao controle a uma taxa de 1000 amostragem por segundo.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing.
	Contexto	A especificação das atividades relacionadas à leitura de dados relativos ao controle do movimento necessita levar em conta a periodicidade da atualização dos dados para garantir a eficiência do controle da cadeira. A freqüência de amostragem determina então a periodicidade da execução das atividades de leitura.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.1: *Template* do NFR1 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-2
	Nome	Temporização do Controle da Cadeira
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Período
	Descrição	O controle da cadeira deve prover novos valores válidos de movimento a cada 20 milisegundos. Este intervalo de atualização garante um deslocamento seguro tendo em vista a baixa velocidade desenvolvida pela cadeira.
	Casos de Uso Afetados	Movement Control.
	Contexto	A especificação da periodicidade do cálculo dos novos parâmetros de movimento da cadeira deve ser realizada na especificação do controle de movimento.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.2: *Template* do NFR2 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-3
	Nome	Tempo de leitura de dados no pior caso
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/WCET
	Descrição	Apresenta o pior caso de tempo de execução das atividades de leitura de dados no sistema.No pior caso, as atividades de leitura de dados sobre o movimento da cadeira não podem demandar mais que 3 milisegundos e a leitura do joystick, 2 milisegundos.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing
	Contexto	A especificação das atividades de leitura de dados deve levar em conta os critérios de pior caso de execução.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.3: *Template* do NFR3 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-4
	Nome	Tempo de execução do controle da cadeira no pior caso
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/WCET
	Descrição	Apresentao pior caso de tempo de execução do controle da cadeira. Para garantir a adequação dos movimentos da cadeira, o algoritmo de controle deve ser capaz de executar em 7 milisegundos no pior caso.
	Casos de Uso Afetados	Movement Control.
	Contexto	A especificação da atividade de controle de movimento da cadeira deve levar em conta seu pior caso de execução.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.4: *Template* do NFR4 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-5
	Nome	Prazo limite para execução de atividades
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Deadline
	Descrição	Apresenta os prazos limite para a execução de atividades do sistema. O funcionamento adequado da cadeira exige que as periodicidades da amostragem de valores e do controle da cadeira sejam respeitadas, não sendo admitido que a execução de uma destas atividades adentre a janela de tempo reservada ao próximo período de execução.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing; (3) Movement Control.
	Contexto	A especificação de atividades periódicas com limitação de deadline deve ser observada na parametrização dos atributos temporais de execução.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.4: *Template* do NFR5 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-6
	Nome	Prazo de Validade
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Precisão/Utilidade (Prazo de Validade)
	Descrição	Controla a validade de dados disponíveis ao controle do sistema. Os dados de movimento e de posição do joystick tem um prazo de validade de 15 milisegundos para serem utilizados pelo controle da cadeira. Caso estes dados não sejam utilizados neste intervalo de tempo, são considerados inválidos.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing; (3) Movement Control.
	Contexto	Toda vez que um dado amostrado pelo sistema for lido para ser utilizado no controle da cadeira, deve-se realizar a checagem de sua validade temporal.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.6: *Template* do NFR6 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-7
	Nome	Temporização do Alarme
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/WCET, Deadline, Latência de ativação
	Descrição	Condiciona temporalmente a atividade de alarme. A atividade que executa o alarme não deve demorar que 2 milisegundos para iniciar sua exeucção. Ela também não deve permanecer em execução por mais que 2 milisegundos no pior caso para executar, bem como não deve ultrapassar o limite de 5 milisegundos para terminar a sua execução.
	Casos de Uso Afetados	Signal Alarm
	Contexto	A especificação da atividade de alarme deve parametrizá-la conforme a descrição de sua temporização.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.7: *Template* do NFR7 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-8
	Nome	Controle de acesso a recursos
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Sincronização
	Descrição	Controla o acesso a recursos compartilhados e distribuídos do sistema. O acesso aos dados amostrados pelos sensores de movimento e pelo joystick deve ser controlado para que se possa manter sua consistência.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing
	Contexto	O controle de acesso, que deve ser limitado temporalmente, deve ocorrer toda vez que os dados controlados forem manipulados, seja para leitura ou atulização (escrita).
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura C.8: *Template* do NFR8 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-9
	Nome	Acesso remoto a dados
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Comunicação
	Descrição	Controla a comunicação que permite o acesso remoto a dados distribuídos no sistema. Os nodos do sistema que participam desta comunicação necessitam de garantias de integridade e entrega das mensagens pelas quais transmitem dos dados.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing
	Contexto	O controle da comunicação que permite o acesso aos dados amostrados pelos sensores e joystick deve ocorrer toda vez que houver troca de mensagens entre os nodos participantes da transmissão dos dados.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Média
	Situação	Finalizado

Figura C.9: *Template* do NFR9 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	Versão Aprovada Anterior à Resolução de Conflitos	Versão pós resolução de conflitos
Identificação	**Identificador**	NFR-10	NFR-10
	Nome	Consumo de Energia	Consumo de Energia
	Autor	Edison Pignaton de Freitas	Edison Pignaton de Freitas
Especificação	**Classificação**	Embarcado/Consumo de Potência	Embarcado/Consumo de Potência
	Descrição	A execução de atividades do sistema gera um gasto de energia que deve ser mensurado e controlado.	A execução de atividades do sistema gera um gasto de energia que deve ser mensurado.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing; (3) Movement Control; (4) Signal Alarm	(1) Joystick Sensing; (2) Movement Sensing; (3) Movement Control; (4) Signal Alarm
	Contexto	Toda vez que uma atividade é executada no sistema, deve-se verificar o seu consumo. Deve-se reduzir a demanda de energia de atividades ociosas do sistema.	Toda vez que uma atividade é executada no sistema, deve-se verificar o seu consumo.
	Escopo	Global	Global
Decisão e Evolução	**Prioridade**	Média	Média
	Situação	Aprovado	Finalizado

Figura C.10: *Template* do NFR10 do Sistema de Controle de Movimento da Cadeira de Rodas

	Item	
Identificação	**Identificador**	NFR-11
	Nome	Tempo de Resposta Final
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Desempenho/Tempo de Resposta
	Descrição	O sistema deve preservar a interatividade de modo que os comandos inseridos sejam executados com um retardo máximo de 0,5 segundo.
	Casos de Uso Afetados	(1) Joystick Sensing; (2) Movement Sensing; (3) Movement Control
	Contexto	Toda vez que o usuário executar um comando sobre o sistema, este deve responder em um intervalo de tempo menor que 0,5 segundo.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Crítica
	Situação	Finalizado

Figura C.11: *Template* do NFR11 do Sistema de Controle de Movimento da Cadeira de Rodas

APÊNDICE D CONJUNTO DE TEMPLATES FUNCIONAIS: HELICÓPTERO NÃO-TRIPULADO

		Item	Descrição
Geral	Identificação	**Identificador**	FR-1
		Nome	Helicopter Movement Control
		Objetivo	Responsável pelo controle helicóptero através do processamento dos dados amostrados e controle dos valores a serem aplicados sobre os atuadores.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	O controle de navegação deve informar o modo de operação e as coordenadas do movimento. Dados sensoriados devem estar disponíveis para uso.
		Pós-condição	Os parâmetros de atuação devem ser atualizados.
		Ator primário	Navigation Control
		Ator secundário	
	Decisao e Evolução	**Prioridade**	Crítica
		Situação	Finalizado
Caminhos		**Primário (Normal)**	O sistema deve calcular os novos parâmetros de atuação com base nos dados obtidos pelos sensores de movimento e ambiente além dos parâmetros de movimento fornecidos pelo controle de navegação.
		Alternativo	
		Excepcional	O controle de movimento passa a ser realizado em modo crítico, com prioridades estabelecidas pela doutrina adotada para a missão. A maneira como os dados são amostrados e os parâmetros de atuação a são atualizados depende desta política.
Cenários		**Principal**	(1) O sistema acessa os dados de movimento e ambiente. (2) O sistema recebe os parâmetros de movimento do sistema de navegação (3) O sistema realiza o cálculo dos novos parâmetros de movimento. (4) Os novos parâmetros são então disponibilizados para serem aplicados ao controle dos atuadores de movimento.
		Variações	

Figura D.1: *Template* do FR1 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-2
		Nome	Special Condition Movement Control
		Objetivo	Responsável pelo controle do movimento do helicóptero em situações críticas.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	O controle de navegação deve informar o modo de operação e as coordenadas do movimento.
		Pós-condição	Os parâmetros de atuação devem ser atualizados.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Crítica
		Situação	Finalizado
Caminhos		**Primário (Normal)**	O sistema deve calcular os novos parâmetros de atuação com base nos dados obtidos pelos sensores de movimento e ambiente além dos parâmetros de movimento fornecidos pelo controle de navegação.
		Alternativo	Caso os dados dos sensores não estejam disponíveis, utiliza-se os últimos dados válidos.
		Excepcional	
Cenários		**Principal**	(1) O sistema acessa os dados de movimento e ambiente. (2) O sistema recebe os parâmetros de movimento do sistema de navegação (3) O sistema realiza o cálculo dos novos parâmetros de movimento. (4) Os novos parâmetros são então disponibilizados para serem aplicados ao controle dos atuadores de movimento.
		Variações	(1) A indisponibilidade dos dados sensoriados faz com que o sistema utilize os últimos dados válidos para realizar o controle.

Figura D.2: *Template* do FR2 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-3
		Nome	Guidance
		Objetivo	Controlar o movimento do helicóptero em relação a um sistema de coordenadas. Realiza o controle direto sobre os motor e rotor principal.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Os parâmetros de moviemento do helicótero devem estar atualizados.
		Pós-condição	O motor principal deve alterar a rotação e torque para os novos valores estabelecidos, bem como o passo do rotor de principal deve ser atualizado em conformidade com o novo valor.
		Ator primário	Navigation Control
		Ator secundário	Main Rotor Actuator
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	O sistema deve transmitir ao motor principal as informações de rotação e torque e ao servo que controla o rotor o passo a ser aplicado.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) Dentro do ciclo do controle de movimento, o sistema realiza o controle direto sobre o rotor principal de modo a efetivar os movimentos do helicóptero em relação ao sistemas de coordenadas referencial, através da atualização dos valores dos parâmetros deste atuador.
		Variações	

Figura D.3: *Template* do FR3 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-4
		Nome	Piloting
		Objetivo	Controlar o movimento do helicóptero em torno de seu eixo vertical. Realiza o controle direto sobre os motor e rotor de cauda.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Os parâmetros de moviemento do helicótero devem estar atualizados.
		Pós-condição	O motor de cauda deve alterar a rotação e torque para os novos valores estabelecidos, bem como o passo do rotor de cauda deve ser atualizado em conformidade com o novo valor.
		Ator primário	Navigation Control
		Ator secundário	Back Rotor Actuator
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	O sistema deve transmitir ao motor de cauda as informações de rotação e torque e ao servo que controla o rotor o passo a ser aplicado.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) Dentro do ciclo do controle de movimento, o sistema realiza o controle direto sobre o rotor de cauda de modo a efetivar os movimentos do helicóptero em torno do seu eixo vertical, através da atualização dos valores dos parâmetros deste atuador.
		Variações	

Figura D.4: *Template* do FR4 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-5
		Nome	Control Alarm
		Objetivo	Controlar o acionamento e desligamento do alarme de eventos do sistema de controle de movimento.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	Ocorrência de alguma violação no funcionamento especificado de algum componente do controle de movimento.
		Pós-condição	Informação transmitida ao sistema de manutenção e ao sistema de comunicação com a estação base.
		Ator primário	Navigation Control
		Ator secundário	Maintenance System, Data Transfer System
	Decisão e Evolução	**Prioridade**	Média
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Ao ocorrer uma falha no sistema de controle o alarme é acionado informando o problema ao sistema de manutenção e ao sistema de transmissão de dados.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) A detecção de uma falha gera aciona o alarme. (2) O alarme envia a informação do problema ao sistema de manutenção. (3) o alarme envia a informação do problema ao sistema de transmissão de dados e solicita o seu envio à estação base.
		Variações	

Figura D.5: *Template* do FR5 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	Identificação	Identificador	FR-6
		Nome	Enviroment Sensing (Temperature Sensing)
		Objetivo	Realiza a amostragem da temperatura ambiente.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	
		Pós-condição	Dados de temperatura amostrados.
		Ator primário	Navigation Control
		Ator secundário	Temperature Sensor
	Decisão e Evolução	Prioridade	Média
		Situação	Finalizado
Caminhos		Primário (Normal)	Amostra a temperatura do ambiente.
		Alternativo	
		Excepcional	
Cenários		Principal	(1) O sistema captura a amostra da temperatura;
		Variações	

Figura D.6: *Template* do FR6 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	Identificação	Identificador	FR-7
		Nome	Enviroment Sensing (Humidity Sensing)
		Objetivo	Realiza a amostragem da umidade ambiente.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	
		Pós-condição	Dados de umidade amostrados.
		Ator primário	Navigation Control
		Ator secundário	Humidity Sensor
	Decisão e Evolução	Prioridade	Média
		Situação	Finalizado
Caminhos		Primário (Normal)	Amostra a umidade do ambiente.
		Alternativo	
		Excepcional	
Cenários		Principal	(1) O sistema captura a amostra da umidade;
		Variações	

Figura D.7: *Template* do FR7 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-8
		Nome	Enviroment Sensing (WindSensing)
		Objetivo	Realiza a amostragem da velocidade e direção do vento.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	
		Pós-condição	Dados de direção e velocidade do vento amostrados.
		Ator primário	Navigation Control
		Ator secundário	Wind Sensor
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Amostra a umidade do ambiente.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) O sistema captura a amostra da umidade;
		Variações	

Figura D.8: *Template* do FR8 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-9
		Nome	Enviroment Sensing
		Objetivo	Realiza o tratamento das informações de ambiente.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	
		Pós-condição	Dispoibilização dos dados de direção e velocidade do vento, temperatura e umidade amostrados.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Disponibiliza as informações de ambiente para o sistema.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) O sistema digitaliza as amostras de dados colhidos pelos sensores de ambiente; (2) O sistema disponibiliza o valor amostrado para o sistema.
		Variações	

Figura D.9: *Template* do FR9 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	Identificação	Identificador	FR-10
		Nome	Rotor Sensing (MainRotorSensing)
		Objetivo	Realiza a amostragem dos parâmetros de movimento do rotor principal.
		Autor	Edison Pignaton de Freitas
	Contexto	Pré-condição	
		Pós-condição	Dispoibilização dos parâmetros de movimento do rotor principal.
		Ator primário	Navigation Control
		Ator secundário	Main Rotor Sensor
	Decisão e Evolução	Prioridade	Alta
		Situação	Finalizado
Caminhos		Primário (Normal)	Amostra o passo e o torque do rotor principal.
		Alternativo	
		Excepcional	
Cenários		Principal	(1) O sistema captura as amostras do torque e do passo;
		Variações	

Figura D.10: *Template* do FR10 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-11
		Nome	Rotor Sensing (BackRotorSensing)
		Objetivo	Realiza a amostragem dos parâmetros de movimento do rotor de cauda.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	
		Pós-condição	Parâmetros de movimento do rotor de cauda amostrados.
		Ator primário	Navigation Control
		Ator secundário	Back Rotor Sensor
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Amostra o passo e o torque do rotor de cauda.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) O sistema captura as amostras do torque e do passo;
		Variações	

Figura D.11: *Template* do FR11 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

		Item	Descrição
Geral	**Identificação**	**Identificador**	FR-12
		Nome	Movement Sensing
		Objetivo	Realiza o tratamento das informações de movimento dos rotores.
		Autor	Edison Pignaton de Freitas
	Contexto	**Pré-condição**	
		Pós-condição	Dispoibilização dos dados de passo e torque dos rotores.
		Ator primário	Navigation Control
		Ator secundário	
	Decisão e Evolução	**Prioridade**	Alta
		Situação	Finalizado
Caminhos		**Primário (Normal)**	Disponibiliza as informações de movimento dos rotores para o sistema.
		Alternativo	
		Excepcional	
Cenários		**Principal**	(1) O sistema digitaliza as amostras de dados colhidos pelos sensores dos rotores; (2) O sistema disponibiliza os valores amostrados para o sistema.
		Variações	

Figura D.12: *Template* do FR12 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

APÊNDICE E CHECKLISTS: HELICÓPTERO NÃO-TRIPULADO

	Relevante	Prioridade	Restrições/ Condições/ Descrição
Tempo			
Temporização			
Existem atividades ou amostragens periódicas?	X	Alta	Leituras de sensores de ambiente (1000 amostras por segundo) e de elementos mecânicos do sistema (500 amostras por segundo). O sistema deve atualizar os dados de controle a cada 15 milisegundos.
Existem atividades esporádicas?			
Existem atividades aperiódicas?	X	Médio	Acionamento de Alarmes (deadline de 6 milisegundos)
Existe restrição quanto a latência para se efetivar o início da execução de alguma atividade no sistema?	X	Alta	A mudança de modo de operação deve refletir nos elementos de controle da aeronave (o quanto antes! - Tempos não definidos, reflexo sobre o tempo de resposta especificado pelo desempenho). O alarme deve entrar em funcionamento em 1 milisegundo quando acionado. A atividade de controle não pode demorar mais que 2 milisegundos para entrar em funcionamento.
Existem instantes específicos de início/fim para execução de atividades do sistema?	X	Médio	Envio de comando aos outros subsistemas no caso de operação em modo especial.
Foi especificado algum tempo de pior caso para a execução de atividades do sistema? (ou ao menos existe preocupação com relação a esta propriedade?)			
Precisão			

Existem atividades com flexibilidade no atendimento de seus requisitos temporais?	X	Alta	Apenas em condições especiais de operação, nas quais existe uma priorização de atividades consideradas mais importantes em função da doutrina estabelecida para o cumprimento da missão. No caso da preservação do aparelho, por exemplo, prioriza-se as atividades de controle, sensoramento de movimento e vento em detrimento de todas as outras. Além disto, os requisitos de tempo para as de controle passam a ser "hard".
Caso exista flexibilidade no atendimento de requisitos temporais, o sistema suporta retardo em alguma atividade temporizada?			
O sistema suporta variações no atendimento de requisitos temporais?	X	Alta	O algoritmo de controle utilizado nas atividades referentes à navegação e controle do aparelho não suportam tais variações na alteração das suas variáveis de controle, sob pena de perda de estabilidade do sistema.
Em uma situação de uso degradado do sistema, existe a possibilidade de se utilizar dados antigos?	X	Alta	Em condições especiais de operação, como cumprimento incodicional da missão, o prazo de validade de dados sensorados pelo sistema pode ser ampliado de 25 para 30 milisegundos.
Existe a necessidade de algum tipo de controle sobre a validade dos dados utilizados em algum processo do sistema?			
Existe limite quanto à diferença entre o tempo lógico utilizado pelo sistema e o tempo físico gerado por componentes do sistema?			

Figura E.1: *Checklist* para Requisitos de Tempo do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

Desempenho	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Vazão			
Existe limite quanto ao número de atividades que o sistema pode executar?	**X**	**Alta**	Existe limitação quanto ao número de tarefas a serem executadas em função dos objetivos de cumprimento da missão estabelecidos pela doutrina a ela aplicada.
Existe alguma restrição quanto a captura ou armazenamento de dados?	**X**		Em modo de operação especial, em situações críticas, a vazão de dados sensorados pode ser maior que a capacidade de processamento disponibilizada.
Existem pontos de convergência de dados?	**X**		O subsistema de navegação é um ponto de convergência de dados.
Existem restrições importantes quanto à utilização da banda no sistema?			
Tempo de Resposta			
Existem limitações temporais para o retorno de respostas finais do sistema?	**X**	**Alta**	Os comandos de controle da aeronave devem ter atendimento imediato em caso de situação crítica.
Em caso de desempenho degradado (sobrecarga do sistema), existem respostas finais do sistema que podem ser penalizadas em detrimento de outras?	**X**	**Alta**	O modo de operação do aparelho, estabelecido pela doutrina adotada na missão, derterminará que atividades do sistema devem ser penalizadas. A doutrina cumprimento incodicional determina que as atividades de sensoramento de ambiente, dianóstico, manutenção, e sensoramento de componentes do sistema devem ser penalizadas nesta ordem em detrimento das tarefas relativas ao controle e monitoramento (esta última com maior prioridade). Já a doutrina da preservação do aparelho, estabelece do desativamento do

			monitoramento e sensoramento do ambiente e diagnóstico em detrimento das demais atividades com vista a preservação do sistema.
Existe a possibilidade de se distribuir tarefas para garantir o tempo de resposta de uma atividade do sistema?	**X**		Tarefas de tratamento de dados de sensoramento podem ser redistribuídas para atender requisitos finais de tempo de resposta, como um laço de controle que dependa de tais dados.

Figura E.1: *Checklist* para Requisitos de Desempenho do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

Distribuição	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Alocação de Tarefas			
Existem critérios espaciais que determinem a distribuição das tarefas do sistema?	**X**	**Alta**	As atividades executadas pelo sistema devem atender os critérios de localidade, determinados pela proximidade dos sensores e atuadores.
Existindo pontos de convergência de dados, há possibilidade de redistribuição do tratamento destes dados?	**X**		Embora o subsistema de navegação centralize os dados referentes a navegação e controle do aparelho, o tratamento de dados sensorados pode ser redistribuído.
Estações Participantes			
As Estações Participantes tem processamento dedicado a uma determinada atividade?			
Em caso de sobrecarga do sistema, alguma estação pode substituir ou auxiliar no processamento de outra?	**X**	**Média**	Sim, desde que atenda os objetivos estabelecidos pela doutrina da missão. As atividades de sensoramento, atuação e controle, apesar de deverem atender a questão da localidade, podem migrar em casos de sobrecarga.
Existe restrição quanto a capacidade de processamento das Estações Participantes?			
Existe restrição espacial que empeça a instalação de alguma estação em algum ponto de atuação ou leitura do sistema?			
Comunicação			
Existe restrição quanto ao uso de alguma tecnologia			

de comunicação?			
A comunicação exige garantia de entrega de mensagem?	**X**	**Média**	Esta garantia somente é fornecida para dados de monitoramento em condições normais de funcionamento do aparelho. Em codições de operação em situação crítica, somente é fornecida a garantia para a comunicação de controle do aparelho (interna e com a estação base).
Existe restrição quanto ao tamanho dos dados transmitidos ou recebidos?			
Existe diferença (tamanho, prioridade, ...) entre tráfego de controle e de dados?	**X**		O tráfego de controle tem prioridade salvo se a doutrina de cumprimento de missão incodicionamente seja utilizada. Neste caso se igualam os privilégios dos dois tráfegos.
Sincronização			
Existem recursos ou dados compartilhados?	**X**		Os dados adquiridos pelos sensores podem ser acessados por mais de um componente do sistema.
É necessário controle de oncorrência sobre os dados compartilhados?	**X**	**Alta**	Os dados compartilhados são críticos para o sistema, de forma que o controle sobre o seu acesso é uma questão importante.
Existe alguma política pré-estabelecida para acesso de recursos ou dados compartilhados?			
Existe hierarquia quanto ao acesso aos recursos ou dados compartilhados?	**X**	**Alta**	Atividades de controle têm maior prioridade em relação às de diagnóstico.

Figura E.1: *Checklist* para Requisitos de Distribuição do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

Embarcados	**Relevante**	**Prioridade**	**Restrições/ Condições/ Descrição**
Área			
Existe restrição quanto a área (em silício ou placa de circuito integrado) ocupada por algum componente do sistema?			
Consumo de Potência			
Existe restrição quanto ao consumo de potência de algum elemento do sistema?	**X**	**Alta**	Elementos do sistema que não estejam sendo utilizados devem reduzir o consumo, mas devem ser capazes de realizar suas funções adequadamente quando acionados. O consumo deve ser priorizado de acordo com o modo de operação e a doutrina da missão.
Quanto ao consumo do sistema, existe necessidade de se inserir monitoramento ou controle?	**X**	**Alta**	Deve-se inserir este monitoramento e controle quando necessário ao suporte às restrições impostas por condições de adversas de operação.
Energia total			
Existe restrição quanto a energia disponibilizada ao sistema?			
Existe estimativa de quanto deve durar a energia disponibilizada ao sistema?			
Existem fontes alternativas de energia no sistema?	**X**	**Baixa**	Placas de aquisição de energia solar podem ser instaladas no aparelho, aumentando a disponibilidade de energia para as atividades prioritárias.
Existe restrição referente à geração de calor pelo sistema (silhueta térmica)?			

Memória			
Existe restrição quanto ao uso da memória de armazenamento do sistema? (seja para dados ou programas)			
Existe limitação quanto ao uso da memória de execução no sistema?			

Figura E.1: *Checklist* para Requisitos de Embarcados do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

APÊNDICE F TEMPLATES DE REQUISITOS NÃO-FUNCIONAIS: HELICÓPTERO NÃO-TRIPULADO

	Item	
Identificação	**Identificador**	NFR-1
	Nome	Leitura Periodica de Dados
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Período
	Descrição	O sistema realiza a leitura os dados monitoramento do movimento (dados dos rotores), a uma taxa de 1000 amostras por segundo, o que corresponde a leituras periódicas de 10 milisegundos. A leitura dos dados de ambiente (temperatura, umidade e, velocidade e direção do vento) é realizada a uma taxa de 500 amostragem por segundo. Isto corresponde a realizar leituras periódicas de 20 milisegundos.
	Casos de Uso Afetados	(1) Rotor Sensing (Main Rotor Sensing, Back Rotor Sensing); (2) Enviroment Sensing: (Temperature Sensing, Humidity Sensing, Wind Sensing).
	Contexto	A especificação das atividades relacionadas à leitura de dados relativos ao controle do movimento e leitura das condições de ambiente necessita levar em conta a periodicidade da atualização dos dados para garantir a eficiência do controle do helicóptero. A freqüência de amostragem determina então a periodicidade da execução das atividades de leitura.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.1: *Template* do NFR1 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-2
	Nome	Periodicidade do Controle do Helicóptero
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Período
	Descrição	O controle da cadeira deve prover novos valores válidos de movimento a cada 15 milisegundos.
	Casos de Uso Afetados	Helicopter Movement Control.
	Contexto	A especificação da periodicidade do cálculo dos novos parâmetros de movimento do helicóptero deve ser realizada na especificação do controle de movimento.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.2: *Template* do NFR2 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-3
	Nome	Prazo limite para execução de atividades
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Deadline
	Descrição	Apresenta os prazos limite para a execução de atividades do sistema. O funcionamento previsível da aeronave exige que as atividades respeitem os limites temporais a elas impostas. Desta forma, uma atividade deve realizar suas atividades dentro do espaço de tempo a ela reservado.
	Casos de Uso Afetados	(1) Helicopter Movement Control; (2) Enviroment Sensing; (3) Rotor Sensing.
	Contexto	A especificação de atividades periódicas com limitação de deadline deve ser observada na parametrização dos atributos temporais de execução.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.3: *Template* do NFR3 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-4
	Nome	Prazo de Validade
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Precisão/Utilidade (Prazo de Validade)
	Descrição	Controla a validade de dados disponíveis ao controle do sistema. Os dados sensoriados pelo sistema tem prao de validade de até 25 milisegundos em condições normais de funcionamento. Caso estes dados não sejam utilizados dentro do intervalo de tempo limite, são considerados inválidos.
	Casos de Uso Afetados	(1) Enviroment Sensing; (2) Rotor Sensing; (3) Helicopter Movement Control.
	Contexto	Toda vez que um dado amostrado pelo sistema for lido para ser utilizado no controle do movimento, deve-se realizar a checagem de sua validade temporal. Da mesma forma, toda vez que este dado for atualizado, deve-se atualizar também o seu prazo de validade.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.4: *Template* do NFR4 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-5
	Nome	Temporização do Alarme
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Deadline, Latência de Ativação, WCET
	Descrição	Condiciona temporalmente a atividade de alarme. A atividade que executa o alarme não deve demorar que 1 milisegundos para iniciar sua execução. Ela não deve ter um custo computacional maior que 3 milisegundos, bem como não deve ultrapassar o limite de 6 milisegundos para terminar a sua execução.
	Casos de Uso Afetados	Signal Alarm
	Contexto	A especificação da atividade de alarme deve parametrizá-la conforme a descrição de sua temporização.
	Escopo	Parcial
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.5: *Template* do NFR5 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-6
	Nome	Controle de acesso a recursos
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Sincronização
	Descrição	Controla o acesso a recursos compartilhados e distribuídos do sistema. O acesso aos dados amostrados pelos sensores de movimento e ambiente deve ser controlado para que se possa manter sua consistência.
	Casos de Uso Afetados	(1) Enviroment Sensing; (2) Rotor Sensing.
	Contexto	O controle de acesso, que deve ser limitado temporalmente, deve ocorrer toda vez que os dados controlados forem manipulados, seja para leitura ou atualização (escrita).
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.6: *Template* do NFR6 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	Versão anterior à resolução de conflitos	Versão pós resolução de conflitos
Identificação	**Identificador**	NFR-7	NFR-7
	Nome	Controle sobre a comunicação	Acesso remoto a dados
	Autor	Edison Pignaton de Freitas	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Comunicação	Distribuição/Comunicação
	Descrição	Controla a comunicação que permite o acesso remoto a dados distribuídos no sistema. Os nodos do sistema que participam desta comunicação necessitam de garantias de integridade e entrega das mensagens pelas quais transmitem dos dados.	Controla a comunicação que permite o acesso remoto a dados distribuídos no sistema. Os nodos do sistema que participam desta comunicação necessitam de garantias de entrega das mensagens pelas quais transmitem dos dados.
	Casos de Uso Afetados	(1) helicopter Movement Control; (2) Back Rotor Sensing; (3) Piloting	(1) Enviroment Sensing; (2) Back Rotor Sensing.
	Contexto	O controle da comunicação que permite o acesso aos dados amostrados pelos sensores deve ocorrer toda vez que houver troca de mensagens entre os nodos participantes da transmissão dos dados.	O controle da comunicação que permite o acesso aos dados amostrados pelos sensores deve ocorrer toda vez que houver troca de mensagens entre os nodos participantes da transmissão dos dados.
	Escopo	Parcial	Parcial
Decisão e Evolução	**Prioridade**	Média	Média
	Situação	Aprovado	Finalizado

Figura F.7: *Template* do NFR7 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-8
	Nome	Consumo de Energia
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Embarcado/Consumo de Potência
	Descrição	A execução de atividades do sistema gera um gasto de energia que deve ser mensurado e controlado. O controle pode ser realizado principalmente com a migração de atividades.
	Casos de Uso Afetados	(1) Enviroment Sensing; (2) Rotor Sensing; (3) Piloting; (4) Guidance. (5) Helicopter Movement Control; (6) Signal Alarm
	Contexto	Toda vez que uma atividade é executada no sistema, deve-se verificar o seu consumo. Deve-se reduzir a demanda de energia de atividades ociosas do sistema, ou realizar a migração de atividades entre unidades de processamento.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.8: *Template* do NFR8 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	Versão anterior à resolução de conflitos	Versão pós resolução de conflitos
Identificação	**Identificador**	NFR-9	NFR-9
	Nome	Alocação Espacial de Atividades	Alocação Espacial de Atividades
	Autor	Edison Pignaton de Freitas	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Alocação de Tarefas	Distribuição/Alocação de Tarefas
	Descrição	O projeto do sistema exige a alocação das atividades de sensoriamento e controle direto sobre os rotores em unidades próximas aos sensores e rotores respectivamente.	O projeto do sistema recomenda a alocação das atividades de sensoriamento e controle direto sobre os rotores em unidades próximas aos sensores e rotores respectivamente.
	Casos de Uso Afetados	(1) Rotor Sensing; (2) Piloting; (3) Guidance.	(1) Rotor Sensing; (2) Piloting; (3) Guidance.
	Contexto	Na distribuição das atividades do sistema de sensoriamento de movimento, exige-se alocar as atividades de tratamento de dados amostrados próximo aos sensores. O quesito de proximidade deve ser aplicado durante a distribuição das atividades de controle direto sobre os rotores.	Na distribuição das atividades do sistema, deve-se alocar as atividades de tratamento de dados de movimento amostrados preferencialmente próximo aos sensores. O quesito de proximidade deve ser observado durante a distribuição das atividades de controle direto sobre os rotores, porém é imperativo que exista a flexibilidade proporcionada pela migração de tarefas em questões relacionadas à economia de energia.
	Escopo	Global	Global
Decisão e Evolução	**Prioridade**	Média	Média
	Situação	Aprovado	Finalizado

Figura F.9: *Template* do NFR9 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-10
	Nome	Migração de Tarefas
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Distribuição/Alocação de Tarefas
	Descrição	As atividades de tratamento de dados amostrados e de atuação direta sobre os rotores podem ser migradas conforme critérios de economia de energia.
	Casos de Uso Afetados	(1) Enviroment Sensing; (2) Rotor Sensing; (3) Piloting; (4) Guidance.
	Contexto	Antes de iniciar ou retomar a execução de uma atividade de amostragem de dados ou atuação sobre rotores, o sistema pode migrá-las para outras unidades de processamento.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.10: *Template* do NFR10 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

	Item	
Identificação	**Identificador**	NFR-11
	Nome	Adaptação da Temporização e Precisão
	Autor	Edison Pignaton de Freitas
Especificação	**Classificação**	Tempo/Temporização/Período Tempo/Precisão/Utilidade (Prazo de Validade)
	Descrição	Adapta os requisitos temporais das atividades e de precisão dos dados às condições que fogem do funcionamento normal do sistema. Doutrina de cumprimento incondicional: prazo de validade dos dados amostrados pelos sensores é dilatado para 30 milisegundos. Doutrina de preservação do aparelho: alterar período de controle para 10 milisegundos.
	Casos de Uso Afetados	(1) Enviroment Sensing; (2) Rotor Sensing; (3) Helicopter Movement Control
	Contexto	Ao se alterar o modo de operação para crítico, o sistema deve observar a doutrina da missão e adaptar os parâmetros temporais conforme especificado.
	Escopo	Global
Decisão e Evolução	**Prioridade**	Alta
	Situação	Finalizado

Figura F.11: *Template* do NFR11 do Sistema de Controle de Movimento do Veículo Aéreo Não-Tripulado

APÊNDICE G MODELO ORIENTADO A OBJETOS DO HELICÓPTERO NÃO-TRIPULADO

O projeto orientado a objetos do VANT apresenta uma série de classes dedicadas a auxiliar o tratamento dos requisitos não-funcionais. Além destas classes, muitos atributos encontram-se espalhados pelas demais classes (que deveriam tratar apenas requisitos funcionais), como os parâmetros temporais, atributos de nível de energia, objetos da classe Menssage (que auxiliam na comunicação entre as unidades de processamento do sistema), objetos da classe Semaphore, responsáveis pelo tratamento da exclusão mútua, entre outros. Importante ressaltar ainda a presença de outras classes como Timer e Scheduler. Elas são responsáveis pelo suporte aos requisitos temporais do sistema. Outra característica que merece ser mencionada é a utilização do padrão Observer-Subject que permite o atendimento do requisito de mudança do estado de movimento de condição normal para especial, o que acarreta a mudança em atributos de tempo, como prazo de validade de dados e período de controle da aeronave. Deve-se enfatizar que todos estes elementos mencionados não concentram apenas em si o tratamento dos requisitos não-funcionais mencionados. Além deles, uma complexa estrutura de atributos e métodos encontram-se espalhados por todo o sistema, de forma que ao se realizar a manutenção de alguma destas características, pode ser necessário alterar diversos elementos do sistema. A Figura G.2 apresenta a versão OO do sistema.

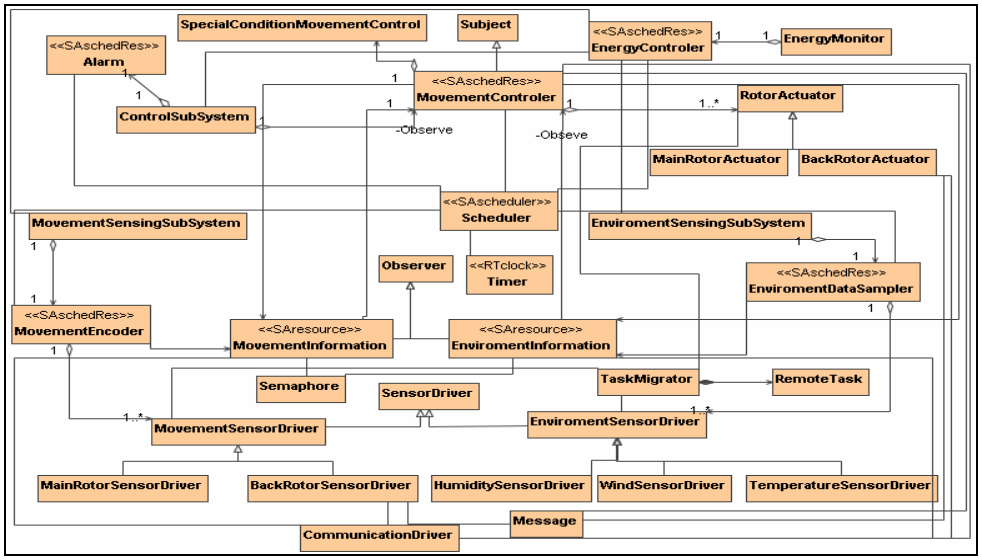

Figura G.2: Diagrama de Classes da Versão OO do Sistema de Controle do VANT