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

MARCIO FERREIRA DA SILVA OLIVEIRA

Exploração do Espaço de Projeto em Sistemas Embarcados Baseados em Plataformas Através de Estimativas Extraídas de Modelos UML

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

Prof. Dr. Flávio Rech Wagner
Orientador

Porto Alegre, maio de 2006.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Oliveira, Marcio Ferreira da Silva

Exploração do Espaço de Projeto em Sistemas Embarcados Baseados em Plataforma Através de Estimativas Extraídas de Modelos UML / Marcio Ferreira da Silva Oliveira – Porto Alegre: Programa de Pós-Graduação em Computação, 2006.

86 f. il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2006. Orientador: Flávio Rech Wagner.

1.Sistemas embarcados. 2.Exploração do espaço de projeto. 3.Estimativas de sistemas 4.*Unified Modeling Language* - UML. I. Wagner, Flávio Rech. II. Título.

AGRADECIMENTOS

Agradeço novamente à Força Superior que me fez forte para suportar todos os momentos de dificuldade que surgiram durante o caminho percorrido até final de mais essa jornada.

Agradeço aos meus avós, Mauricio e Laura, a minha mãe Catarina e a meu irmão Mauricio. A distância impôs dificuldades, mas o amor de vocês foi, e sempre será superior a qualquer adversidade. Sem vocês eu não teria terminado essa jornada. Sou grato também aos meus familiares, que também me apoiaram muito. Aqui no sul eu senti falta de todos vocês.

Agradeço também a presença indispensável na minha vida da Daniele Avila, sem o seu amor e sua compreensão eu não terminaria este trabalho. Muito obrigado por me acompanhar durante este período de dedicação e sacrifício. Igualmente importante, foi o auxilio da família Avila, Dalmo e Luzia (os pais) e Carlos, Rafael e Daiana (os imãos), os quais me acolheram como mais um membro da família.

Gostaria de agradecer ao Ronaldo Ferreira (Bixo) e Alexander Vinson, dois grandes amigos meus. Dividi com eles meus momentos de dificuldade, meu mau humor e a preocupação com o meu trabalho. Mas também dividimos momentos alegres em tempos de lazer ou nos divertimos com nossas próprias dificuldades.

Durante os dois anos de mestrado na UFRGS, contei com a colaboração de alunos e professores. Sou grato aos amigos que fiz durante o mestrado e aos companheiros do Laboratório de Sistemas Embarcados. Em especial, agradeço aos alunos Eduardo Rhod, Lisane Brisolara, e Francisco Assis, pela colaboração durante os trabalhos, pelos conselhos e pela amizade. Agradeço também aos professores Flávio Wagner e Luigi Carro, por ter compartilhado comigo um pouco de seus conhecimentos e por me guiar durante a minha trajetória até o final deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ARTEMIS	*Architecture and Methods for Embedded Media Systems*
CI	Circuito Integrado
CPU	*Central Processing Unit*
DESERT	*DEsign Space ExploRation Tool*
EDF	*Earliest Deadline First*
FIFO	First-In-First-Out
GraphML	*Graph Modeling Language*
GRM	*General Resource Model*
ILP	*Integer Linear Programming*
MDA	*Model Driven Architecture*
MOF	*Meta Object Facility*
OBDD	*Oriented Binary Decision Diagrams*
OMG	*Object Management Group*
PASA	*Performance Assessment of Software Architectures*
POOSL	*Parallel Object Oriented Specification Language*
RTL	*Register Transfer Level*
RTC	*Real-Time Clock*
SESAME	*Simulation of Embedded System Architectures for Multilevel Exploration*
SCPEx	*SystemC Perl Extension*
SHE	*Software/Hardware Engineering*
SoC	*System-on-Chip*
SPADE	*System level Performance Analysis and Design space Exploration*
SPE	*Software Performance Engineering*
SPEU	*System Properties estimation with* UML
QoS	Quality of Service

TDEU	*Trace Driven Execution Unit*
UML	*Unified Modeling Language*
UML-SPT	*Unified Modeling Language profile for Scheduling, Performance and Time*
XMI	*XML Metadata Interchange*
XML	*Extensible Markup Language*
YML	*Y-chart Modeling Language*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Objetivando implementar um sistema embarcado baseado principalmente em *software*, duas abordagens ortogonais estão sendo propostas: Desenvolvimento Baseado em Plataformas, que maximiza o reuso; Desenvolvimento Baseado em Modelos, que aumenta o nível de abstração utilizando conceitos de orientação a objetos e UML para modelar uma aplicação. Porém, com o aumento do nível de abstração, engenheiros de *software* não possuem a idéia exata do impacto de suas decisões de modelagem em questões importantes, como desempenho, e consumo de energia e de memória para uma plataforma embarcada específica. Neste trabalho, propõe-se estimar a memória de dados e de programa, o desempenho e o consumo de energia, diretamente de especificações em UML, como intuito de realizar a exploração do espaço de projeto já nos estágios iniciais do processo de desenvolvimento. Resultados experimentais apresentam erros reduzidos, quando componentes da plataforma são reutilizados e seus custos já são conhecidos para uma plataforma alvo. Aplicações reais foram modeladas de diferentes formas e demonstram a eficiência da abordagem de estimativa para o estagio inicial de exploração do espaço de projeto, permitindo ao desenvolvedor avaliar e comparar diferentes soluções de modelagem. Os valores estimados utilizados na exploração do espaço de projeto podem alcançar taxas de erros inferiores a 5%.

Palavras-Chave: Sistemas embarcados, exploração do espaço de projeto, estimativas de sistemas, UML.

Platform-based Embedded System Design Space Exploration Using UML Models Estimates

ABSTRACT

In order to quickly implement an embedded system that is mainly based on software, two orthogonal approaches have been proposed: Platform-based Design, which maximizes the reuse of components; and Model Driven Development, which rises the abstraction level by using object-oriented concepts and UML for modeling an application. However, with this increasing of the abstraction level, software engineers do not have an exact idea of the impact of their modeling decisions on important issues such as performance, energy, and memory footprint for a given embedded platform. This work proposes to estimate data and program memory, performance, and energy directly from UML model specifications to explore the design space in the early steps of development process. Experimental results show a very small estimation error when platform components are reused and their costs on the target platform are already known. Real-life applications are modeled in different ways and demonstrate the effectiveness of the estimates in an early design space exploration, allowing the designer to evaluate and compare different modeling solutions. The estimated values used in the design space exploration can achieve errors as low as 5%.

Keywords: Embedded Systems, design space exploration, system estimates, UML.

1 INTRODUÇÃO

Sistemas Embarcados podem ser definidos como sistemas eletrônicos de processamento de informação embutidos em um produto de forma transparente para o usuário (MARWEDEL, 2003). Atualmente, uma grande variedade de produtos contém sistemas embarcados: desde dispositivos simples encontrados no cotidiano de nossas vidas, como telefones celulares, aparelhos de DVD, televisões digitais e outros; a dispositivos complexos, como controles de sistemas automotivos, de auxilio médico, de controle em aeronaves ou indústrias. Com o avanço da tecnologia, cada vez mais serão criados produtos que contenham sistemas embarcados, e à medida que aumenta a quantidade e variedade destes produtos, há acréscimo de complexidade destes sistemas devido à integração de mais componentes de *software* e de *hardware* ao sistema (GRAAF, 2003). Adicionalmente, estes sistemas apresentam restrições quanto ao consumo de energia, ocupação de memória, desempenho e outros, que devem ser fortemente considerados durante o desenvolvimento do sistema, tornando o projeto destes um desafio crescente.

Algumas dificuldades surgiram recentemente no projeto de sistemas embarcados, as quais os novos processos de desenvolvimento devem considerar (SANGIOVANNI-VINCENTELLI, 2004): crescimento dos custos de engenharia não recorrente, elevando os custos da produção de um circuito integrado (CI); simultânea pressão para a redução do tempo de entrega (*time-to-market*) de um produto, e sua crescente complexidade; alteração de um modelo de negócio vertical para um modelo horizontal, onde a responsabilidade da produção de um novo projeto é distribuída entre companhias distintas.

Considerando as dificuldades aqui apresentadas, podemos observar o deslocamento do montante de desenvolvimento de *hardware* para *software* e a crescente adoção de métodos que favoreçam a reutilização de componentes em diversos níveis de abstração. Em (DOUGLASS, 2003), é dito que de 60 a 90 % de um sistema é muito similar a sistemas desenvolvidos previamente e que esta porção pode ser reutilizada. O mesmo fato foi observado em (SHANDLE, 2002), onde 95% dos componentes de um sistema embarcado são reusados e 90% deste sistema é composto de *software*.

O projeto de sistemas embarcados baseados em plataformas (*platform-based design*) (SANGIOVANNI-VINCENTELLI, 2001) apresenta características importantes para superar as dificuldades atuais no desenvolvimento de sistemas embarcados. Esta metodologia implementa uma estratégia *meet-in-the-middle*, promovendo a reutilização de componentes desenvolvidos previamente. Além disso, provê diversos níveis de abstração, integrando o esforço de desenvolvimento realizado por diferentes equipes, e também flexibilidade, permitindo que o projeto apresente melhor ajuste para alcançar os requisitos do sistema. Ambientes de desenvolvimento baseados em plataformas devem

prover uma biblioteca com muitos componentes de *software* e de *hardware* pré-caracterizados, permitindo que um projetista monte rapidamente um sistema. Esta biblioteca pré-caracterizada reduz drasticamente o tempo despendido em verificações e a incerteza sobre as propriedades do sistema, aumentando assim a produtividade.

Simultaneamente ao projeto baseado em plataformas, a utilização de níveis maiores de abstração vem sendo adotada para lidar com o acréscimo de complexidade. Nesse contexto, a *Unified Modeling Language* (UML) (OMG, 2005) e o perfil de extensão UML para especificações de sistemas de tempo real, UML *for Scheduling, Performance and Time* (UML-SPT) (OMG, 2002-b), vem ganhando popularidade entre os desenvolvedores de sistemas embarcados. Recentes trabalhos apresentam o potencial de aplicação da UML como uma linguagem para a especificação e projeto de sistemas embarcados (MARTIN, 2005 e LAVAGNO, 2003), devido à sua rica notação gráfica e à capacidade de modelagem, a qual habilita a captura da estrutura e do comportamento do sistema em múltiplos níveis de abstração. Além da UML, a Arquitetura Orientada a Modelo *(Model Driven Architecture* – MDA) (OMG, 2004) está sendo promovida como uma nova abordagem para desenvolver sistemas embarcados através da especificação de modelos. A MDA provê mecanismos para aumentar a portabilidade, a interoperabilidade, a manutenibilidade e o reuso de modelos. A MDA utiliza transformações de modelos, iniciando em modelos em alto nível de abstração, até que modelos detalhados e específicos para uma plataforma sejam obtidos.

A indústria de produtos eletrônicos atualmente provê um vasto número de componentes de *software* e de *hardware*, os quais apresentam diferentes custos de desempenho, energia, memória e outros. Em um ambiente de desenvolvimento baseado em plataformas, o projetista deve ser capaz de explorar um grande espaço de projeto e selecionar as melhores decisões de projeto, escolhendo quais e como os componentes serão utilizado na implementação final do sistema. Esta tarefa pode consumir muito tempo. Portanto, são necessários novos métodos e ferramentas para avaliar e selecionar as melhores alternativas de projeto.

1.1 Motivação

A alteração do foco do desenvolvimento de sistemas embarcados do código fonte para modelos (MRAIDHA, 2004) sugere que o suporte para a exploração do espaço de projeto seja realizado nos estágios iniciais do processo de desenvolvimento, onde agora está concentrado o esforço de desenvolvimento. Além disso, as decisões de projeto tomadas em alto nível de abstração provêm maiores ganhos nas propriedades finais do sistema em face às tomadas em baixo nível (MATTOS, 2004 e THEELEN, 2003).

Como foi destacado em (WANDELER, 2005), o maior desafio de um processo de desenvolvimento é analisar as características essenciais do sistema, logo nos estágios iniciais do projeto, para suportar decisões importantes, antes que muito tempo seja investido em implementações detalhadas. Porém, em altos níveis de abstração os engenheiros de *software* não têm como medir o impacto de suas decisões em questões essenciais durante o desenvolvimento de sistemas embarcados, como desempenho, energia e ocupação de memória para uma plataforma específica. Para que a exploração do espaço de projeto seja realizada nos estágios iniciais do desenvolvimento, é desejável que o projetista possa avaliar as soluções candidatas de forma antecipada, utilizando o mesmo nível de abstração em que o sistema está sendo especificado.

1.2 Objetivos

O objetivo deste trabalho é permitir a exploração do espaço de projeto antecipada através de estimativas extraídas de modelos UML. A técnica apresentada neste trabalho propõe estimar desempenho, consumo de energia e ocupação de memória, com o intuito de guiar o projetista durante a seleção do modelo mais adequado, antes da etapa de geração de código. Visto que esta estimativa é obtida de forma analítica e, portanto, não recorre à simulação, a exploração do espaço de projeto pode ser realizada rapidamente.

Este trabalho investigou e determinou um conjunto de regras para a extração dos parâmetros necessários para a realização das estimativas através de modelos em altos níveis de abstração especificados em UML. Além disso, ele se propôs a verificar como essas estimativas podem ser utilizadas por um projetista para avaliar e comparar diferentes soluções de modelagem para suportar as atividades de exploração do espaço de projeto em um nível muito alto de abstração. Adicionalmente, um conjunto de diretrizes de modelagem, que devem guiar a construção dos modelos do sistema, foi proposta para auxiliar na extração dos parâmetros necessários às estimativas.

Neste trabalho também foram investigadas duas abordagens para a utilização das estimativas. Na primeira abordagem o projetista usa as estimativas para avaliar e comparar diferentes soluções de modelagem para a mesma aplicação, objetivando dar suporte às atividades de exploração de espaço de projeto em alto nível de abstração. A segunda abordagem permite ao projetista dimensionar a plataforma, identificando a configuração mínima de um sistema, em termos de volume de memória, capacidade de processamento e outras propriedades configuráveis, necessária para que a plataforma alvo utilizada no desenvolvimento seja capaz de dar suporte ao sistema.

Para que estes objetivos fossem alcançados, uma ferramenta chamada SPEU (*System Properties Estimation with* UML) foi desenvolvida com o objetivo de estimar as propriedades de um sistema especificado em UML. Além da ferramenta de estimativa, foi proposto um modelo para representar e armazenar informações referentes aos componentes disponibilizados em uma plataforma, montando assim um repositório de informações, o qual pode ser consultado através de uma ferramenta de exploração.

1.3 Resultados Obtidos e Contribuições

Resultados experimentais mostram que, através do uso das estimativas obtidas pela ferramenta SPEU, o projetista pode analisar efetivamente o impacto de suas decisões sobre os modelos UML, permitindo, assim, a exploração do espaço de projeto utilizando esses modelos, antes da geração de código para simulação e validação. Além disso, o trabalho proposto apresenta uma abordagem que permite o reuso de informações de componentes em um ambiente de projeto baseado em plataformas, fato este que aumenta a precisão das estimativas realizadas sobre modelos e permite avaliar o impacto do reuso durante o projeto.

A primeira contribuição que pode ser destacada é que, através da abordagem proposta, o projetista pode avaliar o impacto da arquitetura de *software* utilizada logo nos estágios iniciais do desenvolvimento, apresentando as seguintes vantagens:

- O custo para encontrar e corrigir problemas do sistema nos estágios iniciais do projeto é reduzido;

- Em altos níveis de abstração podem ser encontradas mais oportunidades de otimização do sistema;
- Decisões tomadas em níveis mais elevados podem apresentar maiores impactos positivos na implementação final do sistema.

Como o sistema pode ser avaliado logo nos estágios iniciais, podemos realizar antecipadamente a exploração de alguns dos aspectos do espaço de projeto, os quais já são comumente explorados em níveis mais baixos de abstração, como a seleção de componentes de propriedade intelectual reutilizados da plataforma, o número de tarefas utilizadas, a alocação de tarefas em processadores e mapeamento de processadores em uma estrutura de comunicação.

Ao apresentar um método que permite avaliar quantitativamente os modelos UML, onde poucas atividades de exploração são realizadas, novos aspectos do sistema podem ser explorados, quando o projetista experimenta soluções alternativas que modificam a arquitetura de *software* utilizada para implementar uma determinada função. As novas oportunidades exploram os diferentes resultados, os quais podem ser obtidos alterando-se a distribuição de responsabilidades e as interações entre objetos, a interação com serviços da plataforma ou a forma de alocação dos dados (alocação estática ou dinâmica).

Ao realizar estimativas de sistemas sobre os modelos utilizados pelos desenvolvedores, este trabalho contribui também para a uniformização das notações utilizadas nas etapas de análise e desenvolvimento, visto que não será necessária a geração de modelos distintos para as duas etapas do projeto.

1.4 Organização do Texto

O texto deste trabalho está organizado em cinco capítulos. No próximo capítulo é apresentado o estado da arte em exploração e estimativas de sistemas embarcados, além das contribuições deste trabalho.

No capítulo três são apresentados os detalhes da abordagem proposta para exploração de sistemas embarcados, através de estimativas extraídas de modelos UML. São apresentadas as regras para modelagem, a representação da plataforma, o processo de estimativa através de modelos UML e a realização da exploração.

O capítulo quatro apresenta experimentos que analisam a precisão da estimativa e mostram como a abordagem pode ser utilizada para a exploração do espaço de projeto.

Finalmente, no capítulo cinco são apresentadas as conclusões e os trabalhos futuros.

2 ESTADO DA ARTE E TRABALHOS RELACIONADOS

Neste capítulo é apresentada uma análise do estado da arte dos trabalhos relacionados à exploração do espaço de projeto, à estimativa e à representação de sistemas embarcados. Ao final do capítulo uma análise compara os trabalhos apresentados à presente dissertação e suas contribuições.

2.1 *System level Performance Analysis and Design space Exploration* – SPADE

Em (LIEVERSE, 2001), é apresentada uma metodologia chamada SPADE para exploração do espaço de projeto no desenvolvimento de sistemas heterogêneos de processamento de sinais. A avaliação da arquitetura e da aplicação é realizada no nível de sistema, através da simulação de um conjunto de processos que formam redes de Kahn (KAHN, 1974). Estes processos são descritos em uma linguagem de implementação, C ou C++, e utilizam uma API (*Application Programming Interface*) que provê funções de leitura, escrita e execução. As arquiteturas são modeladas através de blocos genéricos. O projetista deve instanciar e interconectar um conjunto de blocos para definir uma arquitetura. Estes blocos não contêm informações comportamentais e formam unidades de processamento, de comunicação e de armazenamento, os quais devem ser modelados identificando-se interfaces e TDEU's (*Trace Driven Execution Unit*) – unidades que executam os *traces* gerados pelos processos mapeados a eles. Cada TDEU possui uma lista de instruções simbólicas (especificadas pela API) que pode executar e uma relação de custos para a execução destas instruções. Após uma simulação guiada pelo *trace* da aplicação modelada, pode-se recolher informações sobre a utilização do processador, a carga nas interfaces, o volume de dados trafegados e os tempos de atraso nos barramentos.

Segundo Lieverse, a aplicação deve ser modelada separadamente da arquitetura (*hardware* que suporta a aplicação: unidade de processamento, estrutura de comunicação e unidades de armazenamento/memórias) e uma etapa de mapeamento explícito deve ser realizada, permitindo o reuso dos modelos e flexibilizando a exploração do espaço de projeto.

A proposta de Lieverse apresenta um ponto importante no desenvolvimento de sistemas embarcados: a separação entre a aplicação e a arquitetura que irá suportá-la. Mas a forma de especificação recorre a linguagens de implementação, baixando o nível de abstração. A utilização de instruções simbólicas permite que a mesma descrição seja utilizada para diferentes arquiteturas, além de aumentar o nível de abstração. Porém, estas instruções simbólicas devem ser especificadas na aplicação pelo projetista, gerando, assim, um modelo para o desenvolvimento e outro para a estimativa.

2.2 ARTEMIS/SESAME

O *framework* SESAME (*Simulation of Embedded System Architectures for Multilevel Exploration*) (PIMENTEL, 2006) integrante do projeto ARTEMIS (*Architecture and Methods for Embedded Media Systems*) (PIMENTEL, 2001) provê métodos para modelagem e simulação em alto nível de abstração e ferramentas para avaliação e exploração automática de sistemas embarcados heterogêneos baseados em SoC orientados para aplicações multimídia.

O *framework* SESAME é uma evolução do projeto SPADE e apresentam algumas similaridades. Assim como no SPADE, o SESAME tem como princípio a separação dos conceitos ortogonais. Sendo assim, o sistema embarcado é especificado através do modelo da aplicação independente da arquitetura, o modelo da arquitetura independente da aplicação e do mapeamento explícito entre esses dois modelos. O SESAME suporta múltiplos modelos de computação para representação do sistema, de acordo com a tarefa a ser executada. Redes de Khan são utilizadas para modelar o comportamento da aplicação, Grafos de Fluxo de Dados são utilizados para facilitar o refinamento dos modelos e modelo de eventos discretos para simulação rápida dos modelos da arquitetura.

Para realizar a exploração do espaço de projeto, o desenvolvedor deve experimentar as diversas opções de mapeamento. O sistema embarcado é avaliado após o mapeamento através de simulação em nível de sistema, assim como na proposta do SPADE. Porém, para guiar o desenvolvedor na escolha dos mapeamentos que podem apresentar os melhores resultados, o SESAME utiliza um método de otimização multi-objetivo (ERBAS, 2003). Este remove do espaço de projeto as alternativas que não satisfazem as restrições do sistema e disponibiliza o conjunto das melhores alternativas através da otimização de um problema multi-objetivo, que avalia o tempo de processamento, consumo de energia e o custo da arquitetura. O conjunto resultante do processo analítico pode, então, ser avaliado através de simulação. O modelo da aplicação é composto por uma descrição comportamental e outra estrutural. O primeiro são Redes de Khan especificadas em C/C++ e uma API. O segundo representa os processos utilizados e como eles estão interconectados, e é descrito em YML (*Y-chart Modeling Language*). Uma máquina virtual executa a especificação da aplicação e gera um conjunto de *traces* que serão utilizados como carga de trabalho para o modelo da arquitetura. Este modelo é especificado em nível de transação (*transaction level*), através da YML e SCPEx, uma biblioteca embutida para SystemC, e representa a estrutura da arquitetura. O mapeamento utiliza processadores virtuais e FIFO *buffers*, os quais são automaticamente mapeados para os processos Khan formando uma camada de mapeamento especificada em YML. Os processadores virtuais da camada de mapeamento lêem os eventos dos traces dos processos Khan e repassam para os processadores reais do modelo da arquitetura. O mapeamento entre os processadores virtuais e FIFO *buffers* são também representados em YML e devem ser especificados manualmente. Na proposta do SESAME, o sistema pode ser refinado para extrair mais informações a respeito do comportamento da aplicação sobre a arquitetura mapeada. Este refinamento é realizado somente na camada de mapeamento, onde os processadores virtuais recebem os traces de alto nível da aplicação e repassam ao processador da arquitetura eventos de níveis mais detalhados, através de transformações em Grafos de Fluxo de Dados. Desta forma o sistema é refinado mantendo-se a descrição de alto nível da aplicação para que possa ser reutilizada em diferentes mapeamentos.

A proposta do framework SESAME apresenta uma boa solução para a avaliação do sistema em diferentes níveis de abstração, reduzindo a diferença entre a especificação abstrata da aplicação e a arquitetura detalhada que a suportará, através de transformações sucessivas. Objetivando aumentar a eficiência do processo de exploração do espaço de projeto, a remoção automática das alternativas inválidas é realizada antes da exploração e através de simulação. Porém apresenta a mesma desvantagem do SPADE, a aplicação é especificada em nível de implementação. Além disso, exige que o desenvolvedor utilize linguagens diferentes para especificação de diferentes aspectos do sistema.

2.3 Bontempi

A proposta encontrada em (BONTEMPI, 2002) utiliza métodos de análise de dados para extrair as características funcionais (*software*) e arquiteturais (*hardware*) que serão importantes para a predição do desempenho, determinando, assim, uma assinatura. Esta assinatura é extraída do código fonte da aplicação executada em uma arquitetura. Através de métodos não-lineares, como o algoritmo de inteligência artificial *lazy learning*, utilizado pelos autores, são gerados modelos estimadores de desempenho calibrados com as assinaturas de um conjunto de aplicações. Esta abordagem tem o objetivo de separar a assinatura extraída da aplicação do modelo de estimativa. Através desta técnica Bontempi realiza a exploração do espaço de projeto em nível de sistema, verificando diferentes métricas, tais como, latência, ciclos de execução, consumo de energia e outras. Após a geração do modelo do estimador, para estimar as propriedades de uma nova aplicação, deve-se extrair a assinatura e aplicá-la como entrada para o modelo do estimador.

A proposta de Bontempi, assim como na metodologia SPADE, modela separadamente o comportamento da aplicação e a arquitetura alvo e, ainda, utiliza o mapeamento explícito entre os dois modelos para posteriormente extrair a assinatura do sistema. Embora a assinatura seja separada do modelo do estimador, ela inclui a arquitetura e, por isso, para explorar diferentes mapeamentos, é necessário gerar uma nova assinatura através de *traces* e de análise de dados. O uso de métodos não-lineares para realizar estimativas permite capturar características não lineares do funcionamento de um sistema, aumentando, assim, a precisão das estimativas. Porém, a análise da aplicação é realizada após toda a sua implementação, reduzindo o espaço de projeto a ser explorado e aumentando os custos, caso seja necessária alguma alteração na especificação do sistema.

2.4 Platune

Em (GIVARGIS, 2002a e GIVARGIS, 2002b), é apresentado o *framework* Platune, um ambiente de ajuste de desempenho e energia para plataformas SoC's (*System-on-Chip*), baseado em um algoritmo para exploração de configurações em um SoC parametrizável em nível de sistema. Este ambiente utiliza modelos comportamentais e de energia, descritos em C, dos componentes parametrizáveis de um SoC (processador, memórias, barramentos e outros). Através destes, o ambiente captura as informações dinâmicas de desempenho e de consumo de energia em uma simulação com precisão de ciclos, realizada em nível de sistema. Para construir os modelos, são utilizadas informações obtidas através de simulações em níveis mais baixos de abstração, mas, após a construção dos modelos, este tipo de simulação não é mais necessário.

Além dos modelos de simulação, o ambiente utiliza código C compilado para a plataforma alvo como modelo funcional da aplicação. Para realizar as análises deve ser realizado um mapeamento entre a arquitetura do SoC e a aplicação, do mesmo modo que na metodologia SPADE (ver seção 2.1). Um grafo direcionado é utilizado para identificar interdependência entre os parâmetros da plataforma. Esta informação reduz o espaço de projeto, visto que algumas configurações do SoC não necessitam ser avaliadas. De posse dos resultados das análises, as configurações arquiteturais da plataforma podem ser alteradas para ajustar o desempenho e o consumo de energia aos requisitos do sistema. A exploração do espaço de projeto é baseada em uma busca em uma curva de pontos de Pareto, dado um parâmetro fixo. Por exemplo, fixado o desempenho, o algoritmo retorna a configuração de menor consumo de energia.

A metodologia utilizada pelo Platune permite que uma diversidade de parâmetros de uma arquitetura seja explorada, mas, assim como na proposta de Bontempi (ver Seção 2.2), todo *software* deve estar anteriormente desenvolvido. A metodologia permite um ajuste fino da configuração de memória *cache*, barramentos, tensão do processador e outros. Mas não permite a avaliação do *software* propriamente dito, que será executado nesta arquitetura. E, portanto, permite somente o ajuste da arquitetura de *hardware* para adequar a aplicação a seus requisitos, o que pode acarretar em alto custo de *hardware*. Além do mais, recorre a níveis baixos de especificação e simulação. O Platune demonstra-se mais adequado para a realização de ajustes finos quando o sistema já estiver em etapas avançadas do desenvolvimento.

2.5 Russell

Em (RUSSELL, 2003), propõe-se uma técnica para avaliação de desempenho com o objetivo de realizar as explorações arquiteturais para o desenvolvimento de sistemas embarcados baseados em componentes. Nessa proposta, uma arquitetura é decomposta em um modelo funcional (*software*) e outro estrutural (*hardware*). O modelo funcional da arquitetura especifica um conjunto de processos seqüenciais e/ou concorrentes, onde cada processo é definido por um ou mais módulos. Estes módulos são compostos por procedimentos especificados através de operações seqüenciais descritas em linguagens imperativas, como C ou Verilog, de onde será extraído um grafo de controle de fluxo. O modelo estrutural consiste nos subsistemas que suportam a execução das operações especificadas no modelo funcional e nas conexões entre subsistemas que representam a comunicação entre eles. Os subsistemas podem ser elementos primitivos estruturais, como a memória e um controlador de memória, ou tipos de elementos que representam agregações, como uma CPU composta por um núcleo de processamento, um controlador de memória e um controlador de barramento (*bridge*).

Assim como em outras abordagens, também é realizado o mapeamento do modelo funcional ao modelo estrutural, de forma que as operações do modelo funcional sejam mapeadas para elementos de execução do modelo estrutural e operandos sejam mapeados para elementos de armazenamento. O custo de uma operação é determinado pelas informações de busca de instruções, tempo de processamento do núcleo e tempo de acesso a um elemento de armazenamento. Um modelo probabilístico é utilizado para representar os efeitos de uma falha na memória *cache*. Esta abordagem prevê que os elementos do modelo estrutural estejam previamente caracterizados, eliminando detalhes da micro-arquitetura. O modelo abstrato proposto assume independência entre as operações, reduzindo a precisão das estimativas.

Nessa abordagem, um cenário deve ser determinado utilizando o conjunto de grafos de fluxo de controle resultante do modelo funcional. Este cenário contém as operações que serão executadas em um determinado fluxo. Então, um *pseudo-trace* é gerado caminhando pelo grafo. Ao final, utilizando-se os modelos, o desempenho do sistema é calculado. De posse dos dados de desempenho, pode-se alterar um componente de uma arquitetura candidata, a fim de explorar o espaço de projeto.

A análise estática utilizada por Russel facilita a exploração do espaço de projeto, visto que o sistema em desenvolvimento pode ser rapidamente avaliado. Fato que também contribui para a exploração do espaço de projeto é que a metodologia utilizada permite a exploração dos módulos funcionais e não somente do *hardware* que compõe a arquitetura. Um ponto relevante deste trabalho é a representação do sistema através de um grafo de fluxo de controle e a geração de um *pseudo-trace* utilizado para facilitar a estimativa. O inconveniente presente nesta abordagem é a utilização de linguagens de baixo nível para especificação do sistema.

2.6 Cinderella

Em (LI, 1995), é proposto um método para obter, através do uso de equações lineares, o número de execuções de blocos básicos de uma aplicação, a partir de um código fonte em linguagem de montagem, e estimar seu desempenho. A ferramenta Cinderella implementa esta metodologia gerando um grafo de fluxo de controle. Neste grafo, os nós são blocos básicos e as arestas são fluxos da execução. Restrições estruturais e funcionais da aplicação são extraídas a partir deste grafo, na forma de restrições lineares. As restrições estruturais estão explicitamente especificadas no grafo de fluxo de controle, por exemplo, como uma expressão *if-then-else*, e podem ser obtidas automaticamente. Restrições funcionais, necessárias para especificar como a aplicação será executada dependendo dos dados de entrada, são especificadas manualmente pelo projetista. Estas restrições especificam os limites inferiores e superiores de execuções iterativas e informações adicionais sobre execuções condicionais, por exemplo indicando que um determinado bloco básico deve ser executado pelo menos uma vez. Após a especificação de todas as restrições o problema é resolvido maximizando ou minimizando uma função de custo construída a partir do grafo, provendo o pior ou melhor caso de execução. Os custos dos nós são obtidos através da soma do custo de todas as instruções que ele contém e o custo das instruções é obtido por simulação ou informado pelo fabricante do processador, para o qual a aplicação está sendo avaliada.

O método implementado permite a avaliação rápida de uma aplicação, informando-se assim se ela atende seus requisitos. Mas como só informação estática é disponível, espera-se erros relativamente altos para a análise de desempenho. Mesmo assim, estes resultados podem ser utilizados de forma eficiente para a exploração do espaço de projeto.

2.7 DESERT

A metodologia utilizada pela ferramenta DESERT (*DEsign Space ExploRation Tool*) (NEEMA, 2003) propõe, a partir de um conjunto de modelos de sub-componentes e outro de restrições de projeto que devem ser satisfeitas, a composição automática de um modelo de um sistema embarcado. DESERT é uma ferramenta independente de domínio, capaz de gerar espaços de projeto e explorá-los atendendo restrições de

projeto. Como essa ferramenta é independente de domínio, ela pode ser associada a diferentes ambientes de modelagem de domínio específico, através de transformação de modelos. Porém, toda a metodologia é apresentada utilizando-se sistemas automotivos modelos em Simulink.

A DESERT recebe como entrada um conjunto de modelos da aplicação e modelos de componentes. A metodologia é dividida em seis atividades. A primeira atividade realiza uma transformação dos modelos de componentes para modelos mais abstratos. Os modelos de componentes são representados através de um meta-modelo Simulink, especificado em UML. Os modelos abstratos gerados são representados, também, através de meta-modelos especificados em UML, o qual representa o modelo do espaço de projeto. A segunda atividade é a geração do modelo do espaço de projeto, realizado manualmente quando um desenvolvedor acrescenta novos componentes. À medida que o espaço de projeto cresce, ele pode ser restringido, acrescentando restrições estruturais (compatibilidade de interfases, compatibilidade semântica, interdependência entre componentes e outras). A terceira atividade codifica as informações contidas no modelo do espaço de projeto em uma codificação binária e a quarta atividade manipulada simbolicamente as informações codificadas através de Diagramas Ordenados de Decisões Binários (*Ordered Binary Decision Diagrams* – OBDD). A terceira e a quarta atividade devem compor os diversos componentes que formam o espaço de projeto, além de encontrar alguns conjuntos de componentes que satisfazem as restrições estruturais impostas. Durante essas atividades o espaço de projeto é restringido através das manipulações simbólicas baseado nas restrições estruturais impostas sobre os componentes. O resultado é um conjunto de modelos, os quais podem ser analisados de forma mais detalhada, através de simulação, para a seleção do modelo mais apropriado. Na quinta atividade o conjunto de soluções é decodificado em conjuntos abstratos de componentes. E finalmente, na sexta atividade os modelos detalhados especificados em Simulink são reconstruídos a partir dos modelos de projeto abstratos.

A metodologia utilizada na ferramenta DESERT apresenta uma importante contribuição, uma metodologia independente do domínio para exploração do espaço de projeto. Segundo a metodologia, um desenvolvedor poderia adaptar os meta-modelos utilizados pela DESERT para aplicá-la a outro domínio. Neste caso, o desenvolvedor poderia utilizar o meta-modelo UML e criar o modelo do domínio da aplicação utilizado pelo DESERT como representação mais abstrata do sistema, composto por diversos componentes, os quais serão selecionados pela ferramenta. Também poderia ser necessário alterar o meta-modelo do espaço de projeto e definir a transformação entre estes dois modelos. Porém, deve-se notar que a exploração do espaço de projeto realizada restringe-se à seleção de componentes em uma estrutura definida previamente pelo projetista, não havendo nenhuma avaliação comportamental. A possibilidade de aplicar a metodologia a diferentes domínios depende da disponibilidade de modelos para cada um destes domínios. O desenvolvedor deve possuir os diferentes modelos para ampliar o suporte da ferramenta. Finalmente, segundo a observação dos autores, a aplicação automática das restrições sobre grandes espaços de projeto pode resultar em uma explosão de OBDD, aumentando a complexidade computacional e a escalabilidade da ferramenta.

2.8 Bernardi

O trabalho encontrado em (BERNARDI, 2002) propõe a geração de modelos de redes de Petri a partir de diagramas UML de seqüência e de estados para, a partir destes,

analisar o desempenho de um sistema. Algumas regras para a especificação e tradução dos diagramas UML foram especificadas, de forma que somente algumas características são traduzidas para os modelos de redes de Petri, visto que na UML não há um formalismo para descrever o sistema. Após a tradução, os modelos de redes de Petri gerados podem ser analisados por ferramentas disponíveis na comunidade. O trabalho proposto por Bernardi apresenta duas formas de análise sobre os modelos. A primeira permite verificar a correção dos cenários descritos por um diagrama de seqüência. Desta forma, é possível identificar situações de corrida ou mesmo se existe um caminho que, partindo do estado inicial do cenário, possibilite sua execução até o final, realizando, assim, validações funcionais. A segunda permite a verificação do desempenho do sistema, como o tempo médio de execução entre o estado inicial e o estado final de um cenário traduzido. Para realizar medidas de tempo, um atraso deve ser atribuído às transições que correspondem a um consumo de tempo no sistema final, como a execução de um método ou uma troca de mensagem.

A metodologia apresentada por Bernardini disponibiliza um conjunto de regras que viabilizam a transformação dos modelos informais da UML em um modelo que apresenta um formalismo matemático. Isto é importante para validação das características do sistema. A metodologia também permite uma especificação detalhada do sistema, quando uma ação associada a uma mensagem de um diagrama de seqüência é detalhada em um diagrama de estados. Este fato aumenta a precisão da análise, mas, em contra-partida, exige modelos com mais detalhes. Embora esta proposta permita a análise do tempo de execução do sistema, informações sobre o tempo de operação devem ser incluídas manualmente no modelo pelo projetista.

2.9 *Software/Hardware Engineering* - SHE

No trabalho encontrado em (THEELEN, 2003), uma metodologia de desenvolvimento chamada *Software/Hardware Engineering* (SHE) (GEILEN, 2001) é utilizada para modelar o desempenho de sistemas baseados em UML. SHE provê um conjunto de heurísticas para a construção de modelos UML e um perfil UML, também chamado SHE. Nesta metodologia, os modelos UML são transformados em modelos executáveis, especificados através de uma linguagem de alto nível, *Parallel Object Oriented Specification Language* (POOSL). Esta linguagem apresenta semântica formal, e, com base neste formalismo, pode-se realizar simulações empíricas e análises matemáticas sobre os modelos e extrair informações sobre o desempenho do sistema, durante os estágios iniciais do projeto. Para permitir a execução da especificação em POOSL, modelos probabilísticos, como Cadeia de Markov, são utilizados para determinar transição de estados, atrasos e comportamento condicional. Uma biblioteca que permite a instanciação de variáveis aleatórias e funções probabilísticas é disponibilizada para permitir a simulação do comportamento do sistema e realizar medidas sobre as questões de desempenho levantadas durante a fase inicial do projeto. As informações relativas ao desempenho podem ser explicitamente especificadas nos modelos, através de valores de qualidade de serviço (*Quality of Service* – QoS) utilizando o UML-SPT. A análise de desempenho dos modelos pode ser computada através de simulação ou, então, analiticamente através de métodos padrão, como "análise de equilíbrio".

Theelen, porém, ressalta três dificuldades encontradas para realizar a análise de desempenho através da metodologia proposta. Primeiro, a tradução dos modelos UML para POOSL é realizada manualmente. Segundo, sistemas reais possuem muitas

atividades concorrentes, tornando a manipulação matemática inviável. E, por último, há dificuldade em determinar o tempo de simulação necessário para extrair resultados precisos. Além das dificuldades ressaltadas por Theelen, o projetista deve utilizar o perfil UML-SHE, que altera consideravelmente a forma dos modelos com novos diagramas e classificadores. Porém, na UML 2.0, alguns dos conceitos disponibilizados no perfil UML-SHE foram incorporados, o que aproxima a metodologia SHE de uma forma padrão de especificação.

2.10 *Software Performance Engineering* – SPE e *Performance Assessment of Software Architectures* – PASA

Uma série de trabalhos relacionados à análise de desempenho em sistemas computacionais está sendo desenvolvida por Williams e Smith. Como resultado destes trabalhos, em (SMITH, 1997) e (WILLIAMS, 1998), são propostos métodos e ferramentas para modelar um sistema em estudo. Em (SMITH, 2003), foi apresentado um processo genérico, chamado SPE (*Software Performance Engineering*), para avaliação de desempenho em sistemas especificados em UML que utiliza as técnicas expostas em seus trabalhos anteriores. Uma extensão das técnicas utilizadas em SPE foi apresentada em (WILLIAMS, 2002), com uma metodologia chamada PASA (*Performance Assessment of Software Architectures*), para verificação do desempenho em arquiteturas de *software*, a qual utiliza os mesmos princípios da SPE.

PASA é uma metodologia baseada em cenários especificados em UML. A análise é baseada nas possíveis interações com o sistema para verificar como ele atende aos requisitos de qualidade. Esta metodologia propõe a utilização de dois modelos para a análise: modelos de execução de *software* e modelos de execução do sistema. O primeiro modelo é mais simples e apresenta informações sobre os aspectos comportamentais da execução e, normalmente, é suficiente para indicar problemas de uma arquitetura fraca. O segundo modelo é dinâmico e apresenta informações sobre carga de usuários e disponibilidade de recursos. O modelo de execução do sistema é mais preciso e complexo, e utiliza os resultados do modelo de execução de *software* como entrada. Através da utilização do modelo mais complexo, informações adicionais são obtidas, como variação do desempenho sobre diferentes cargas de trabalho, informações mais precisas sobre contingência de recursos e o efeito da execução de outros *softwares* sobre os objetos em nível de serviço. Além da possibilidade de extrair dados comparativos sobre a influência no desempenho devido às alterações no *hardware* ou no *software*, identificar a carga de trabalho e recursos que causam atrasos.

Através de alguns experimentos, Williams e Smith demonstram que suas ferramentas e metodologias podem ser bem aplicadas para a análise de desempenho em aplicações de diversos domínios, tais como comércio eletrônico, sistemas de tempo real e aplicações financeiras. Mas a geração dos modelos para a análise de desempenho ainda é realizada de forma manual e exige que o projetista possua forte conhecimento da técnica. Porém, com os avanços da UML esta tradução pode ser facilitada, visto que a UML 2.0 possui mais recursos de expressão e formalismo que não eram disponibilizados anteriormente. A solução dos modelos pode ser obtida através de simulação empírica ou analítica, utilizando Redes de Filas ou de Petri. Contudo, as informações em relação ao desempenho do sistema dependem fortemente das estimativas obtidas pelos projetistas durante a construção dos modelos.

2.11 Petriu e Gu

Em (PETRIU, 2003), a UML e o perfil UML-SPT são utilizados para modelar sistemas e permitir que estes modelos possam ser analisados em alto nível. Em (GU, 2003), é proposta uma metodologia para a tradução dos modelos UML em um modelo intermediário que possa ser traduzido para diferentes modelos de desempenho, como Redes de Filas, Redes de Filas Estendidas, Redes de Filas em Camadas ou, ainda, em um modelo de simulação.

A metodologia proposta por Petriu e Gu é baseada em casos de uso e cenários. Estes cenários podem ser especificados utilizando-se diagramas de seqüência e/ou de atividades. Durante a especificação de um cenário, podem ser especificados caminhos alternativos, execuções paralelas e também refinamentos de um cenário. Os valores medidos ou estimados do sistema podem ser anotados e serão usados durante a análise do sistema. Parâmetros de carga de trabalho (*workload*) devem ser especificados e serão utilizados como dados de entrada para a análise. Eles definem a intensidade de uma requisição, a freqüência relativa de um caminho para um cenário e a demanda por um determinado recurso. As informações sobre os recursos disponíveis podem ser anotadas através de diagramas de implantação, os quais definem recursos, como rede de comunicação, dispositivos de entrada e saída e processadores. Para cada tipo de recurso, o perfil UML-SPT disponibiliza um conjunto de estereótipos e propriedades (*tagged values*), os quais podem ser utilizados para especificar políticas de escalonamento, prioridade, tempo de processamento, tempo de troca de contexto, capacidade e outras informações importantes para a realização das análises. A utilização destes recursos pode ser definida com um valor medido, um valor estimado, uma expressão ou os três. A um caminho alternativo pode-se associar uma probabilidade de execução, que deverá ser considerada durante a análise. E, para uma execução interativa, pode-se especificar o número de interações máximo, mínimo ou uma expressão.

A metodologia proposta por Petriu permite que o projetista especifique um conjunto de parâmetros para avaliar o sistema e retorna o desempenho deste para uma situação especificada por esses parâmetros. A resposta da análise de desempenho pode ser utilizada para avaliar a escalonabilidade de tarefas, o comportamento do sistema em diferentes situações ou para encontrar relações entre as diferentes métricas de desempenho como tempo de resposta, latências e outras. Mas a metodologia não permite, contudo, estimar diretamente o desempenho de um componente em termos de tempo de processamento, o qual deve ser fornecido pelo projetista.

2.12 Hoeben

Como nos diagramas UML geralmente faltam informações para a realização de estimativas de desempenho, o método apresentado em (HOEBEN, 2000) explora um conjunto de idéias e regras para remover de forma automática indireções e abstrações que dificultam a análise de desempenho em modelos UML. Neste trabalho, Hoeben utiliza a Teoria de Filas para realizar cálculos de uso de recursos e tempo de resposta. O tempo de resposta é medido para cada tarefa que o sistema deve executar, identificada através de diagramas de casos de uso. De forma mais completa que Bernardi (ver Seção 2.6), Hoeben utiliza ainda diagramas de classe, os quais representam informações sobre a estrutura estática do sistema que é utilizada para a compreensão e análise do comportamento dinâmico. Diagramas de Interação são utilizados para descrever a decomposição do comportamento de tarefas de usuários até um recurso de *hardware*.

Cada diagrama descreve um método que deve ser combinado com outros diagramas até obter a descrição completa de um comportamento. Diagramas de implantação são utilizados para representar as unidades de processamento, as conexões de rede de comunicação utilizadas e a distribuição dos componentes de *software* entre as unidades de processamento. A informação mais importante contida neste diagrama é a caracterização das propriedades das unidades de processamento e das conexões de rede.

A capacidade de abstração apresentada pela UML recebe diversas críticas, mas, segundo Hoeben, esta é uma característica desejável também para modelos de desempenho. Uma linguagem deve permitir um nível flexível de abstração, permitindo ao projetista focalizar em uma determinada abstração e refiná-la. Porém, para permitir que esta capacidade de abstração não prejudique as estimativas de desempenho, Hoeben descreve um conjunto de regras; estas geram aproximações para lidar com cinco tipos de abstrações que prejudicam a obtenção das informações necessárias para a análise.

A primeira regra permite a estimativa baseada em modelos de granularidade grossa, onde um grande conjunto de informações está encapsulado em componentes, permitindo a estimativa sobre modelos em alto nível de abstração. A segunda regra aproxima informações perdidas com a utilização de interfaces e polimorfismo. Hoeben propõe inserir informações sobre a probabilidade de uma classe pertencer à instância que responderá a uma requisição. Esta probabilidade é baseada no número de instâncias de cada classe que pode servir a uma requisição. A terceira regra diz que, se existem diversas instâncias de uma classe em processadores diferentes, formando diferentes conjuntos por processador, a probabilidade de um conjunto de instâncias atender a uma mensagem é proporcional ao total de instâncias da classe daquele conjunto. A quarta regra, associada ao desenvolvimento de sistemas distribuídos, comenta a respeito da visibilidade, visto que somente as instâncias visíveis são utilizadas nos cálculos de probabilidade descritos anteriormente. Hoeben destaca três níveis de visibilidade: componentes, onde as instâncias são visíveis somente para objetos do mesmo componente; nó, onde as instâncias são visíveis para todos os objetos do processador onde o componente está localizado; pública, onde as instâncias são visíveis para todos os objetos do sistema. Por fim, a quinta regra propõe que conexões de redes de comunicação sejam modeladas em um diagrama de classes com estereótipos que identificam nós de processamento e conexões de rede. Nestas classes serão determinadas as latências para as trocas de mensagens. Não é interessante que uma instância cliente saiba qual é a localização da instância alvo de sua requisição. Mesmo assim, foi proposta a utilização dos mecanismos de extensão da UML sobre os diagramas de interação, para determinar o roteamento de uma mensagem pela rede.

O trabalho de Hoeben contribui para a realização de estimativas através de informações extraídas de modelos UML, pois, com regras de manipulação de abstração presentes nos modelos, a análise de desempenho pode ser facilmente integrada ao processo de desenvolvimento, auxiliando as atividades de exploração do espaço de projeto. Porém, a metodologia prevê que os modelos de interação contenham muitos detalhes, chegando a níveis de abstração semelhantes aos de instruções, além de ser necessário que o projetista adicione informações de desempenho diretamente nos modelos, como nos trabalhos anteriores.

2.13 Análise do estado da arte

Há um grande esforço sendo realizado para permitir que os projetos de sistemas embarcados possam ser implementados de forma rápida e atendendo aos requisitos aos quais eles são submetidos. Nas seções anteriores foram abordados diversos trabalhos onde podemos destacar contribuições e limitações.

Algumas metodologias estudadas extraem suas estimativas de informações obtidas em níveis muito baixos de abstração. A metodologia SPADE, ARTEMIS/SESAME e a proposta de Russell, utilizam uma especificação em uma linguagem de implementação, como C ou C++. Platune, Cinderella e a proposta de Bontempi utilizam níveis ainda mais baixos, em linguagem de montagem, para operar a exploração ou a estimativa. Essas metodologias esperam que todo o *software* do sistema esteja desenvolvido, para então realizarem estimativas e exploração. Embora em níveis mais baixos de abstração seja possível extrair mais informações, muito tempo do projeto é despendido até que o sistema seja avaliado.

Já as metodologias propostas por Bernardi, SHE, SPE/PASA, Petriu/Gu e Hoeben, assim como a proposta deste trabalho, procuram antecipar a avaliação e a exploração do espaço de projeto para as etapas iniciais do processo de desenvolvimento, onde muitas oportunidades de otimizações já estão disponíveis e pouco tempo foi investido na implementação do sistema. Para antecipar a avaliação do sistema essas propostas utilizam modelos UML como fonte de informações. A metodologia proposta em DESERT também utiliza modelos de níveis mais altos de abstração, e foca suas atividades sobre os modelos especificados em Simulink.

Para facilitar o processo de exploração, as propostas procuram apresentar métodos que dividam o sistema em partes ortogonais, permitindo que estas sejam alteradas com poucos ou nenhum ajuste em outras. As metodologias SPADE e ARTEMIS/SESAME definem que o sistema é dividido em aplicação (*software*) e arquitetura (*hardware*). Então, um mapeamento entre ambas as partes deve ser realizado para compor o sistema. Esta abordagem é importante e permite a reutilização da especificação da aplicação quando avaliamos seu comportamento em arquiteturas diferentes. As metodologias propostas por Bontempi e em Cinderella são semelhantes. Essas não permitem que as informações conhecidas previamente a respeito dos componentes de *software* reutilizados sejam empregadas de forma direta na estimativa ou na exploração. Nessas propostas, é necessário que os componentes de *software* reutilizados sejam incorporados à aplicação em desenvolvimento e esta seja novamente avaliada. Isto obriga o desenvolvedor a alterar a especificação da aplicação, sempre que desejar avaliar novos componentes. Como na metodologia adotada em DESERT utiliza apenas informações estruturais, onde um sistema é composto de componentes, o relacionamento entre eles é definido através do acesso à interfaces, não há mapeamento. Logo, a divisão entre aplicação e plataforma assume outra interpretação, onde um ponto de projeto na plataforma da aplicação é refinado até alcançar o modelo da plataforma com seus componentes e parâmetros definidos para a implementação final.

Nas metodologias baseadas em UML, os componentes da plataforma (*hardware*) são considerados recursos que são utilizados pela aplicação. Então, através de modelos de simulação matemática, são estimados os tempos que um componente de *software* espera para ter sua requisição de recurso atendida. Em alguns casos, como na proposta de Bernardi, a plataforma é totalmente abstraída dos modelos, e o desenvolvedor deve especificar os tempos de atraso para as transições de uma Rede de Filas, baseado no seu

conhecimento do sistema. O modelo da plataforma para estas abordagens é o diagrama de implantação, que especifica onde os componentes de *software* serão executados. Esta é uma simplificação da plataforma, na qual, assim como em todas as outras metodologias, a plataforma é resumida nos componentes de *hardware* que suportará a aplicação. Na abordagem proposta neste trabalho, a especificação do sistema está dividida entre o modelo da aplicação, o modelo da plataforma e o mapeamento entre eles. A plataforma assume um sentido mais amplo e pode conter tanto componentes de *software* como de *hardware*, disponibilizando mais conhecimento prévio sobre o sistema, fato este que aumenta a precisão das estimativas.

De forma geral, as metodologias que empregam a UML realizam traduções dos modelos para algum formato ou modelo interno. Nas metodologias propostas por Bernardi e Hoeben, são utilizadas as Redes de Filas. A proposta de SPE/PASA permite a utilização de Redes de Fila ou de Petri. Em SHE, os modelos são traduzidos para uma linguagem de implementação, POOSL que apresenta semântica formal. Já a proposta de Petriu e Gu é utilizar uma representação interna que resume o modelo do sistema e que pode ser traduzido para qualquer formalismo desejado. Essa tradução é necessária visto que a UML não apresenta formalismo que permita a realização de análises formais diretamente sobre os modelos. Também a metodologia DESERT transforma os modelos Simulink em um formato interno. Este é um meta-modelo do espaço de projeto especificado em UML/OCL, o qual é também transformado em OBDD. Nesta dissertação, os modelos UML são convertidos para um conjunto de grafos de instruções e, assim como no Cinderella, são utilizados para formular um problema de programação linear inteira, o qual é utilizado para realizar as estimativas.

As metodologias de Bernardi, de SHE, de Petriu/Gu, de SPE/PASA e de Hoeben recorrem à simulação matemática para obter as estimativas sobre o sistema. As metodologias SPADE, ARTEMIS/SESAME, Platune e a proposta de Russell obtêm esses valores através de simulação empírica utilizando modelos de seus componentes de *hardware* especificados em uma linguagem de implementação. Essas abordagens são mais precisas em suas estimativas, pois utilizam informações dinâmicas sobre o sistema. Porém, são mais lentas que as abordagens como Cinderella e Bontempi. Nessas abordagens, e também na abordagem proposta neste trabalho, métodos analíticos são utilizados e somente informações estáticas sobre o sistema são analisadas. Portanto, podem apresentar erros mais elevados. Com o objetivo de aumentar a precisão das estimativas, a abordagem proposta neste trabalho reutiliza as informações conhecidas previamente a respeito da plataforma, permitindo, assim, que os resultados obtidos sejam utilizados para explorar o espaço de projeto. Na metodologia DESERT não são realizadas estimativas. As análises são realizadas através de manipulações simbólicas sobre os OBDD, os quais contam com informações sobre os componentes inseridos pelo desenvolvedor, através dos modelos.

A maioria dos métodos de exploração e estimativa estudados considera o desempenho (número de instruções, tempo de processamento e outros) como métrica para a avaliação. Mas as metodologias Platune, DESERT, de Bontempi e SPE/PASA prevêm que suas metodologias possam ser aplicadas a outras métricas, como consumo de energia e ocupação de memória. De modo semelhante, a abordagem proposta neste trabalho utiliza uma abordagem de estimativas multi-variável, permitindo ao projetista verificar diferentes requisitos do sistema através de quatro métricas: desempenho, medido em ciclos; energia, medida em atividades de chaveamento de portas; e memórias de dados e de programa, medidas em *bytes*.

Para especificar os modelos UML, a metodologia SHE propõe seu próprio perfil, UML-SHE, que altera consideravelmente a forma dos modelos com novos diagramas e classificadores. Porém, na UML 2.0, alguns dos conceitos disponibilizados no perfil UML-SHE foram incorporados, o que aproxima a metodologia SHE de uma forma padrão de especificação. Hoeben propõe que os diagramas de seqüência sejam decompostos, aumentando o nível de detalhe expresso no diagrama. Hoeben sugere que cada diagrama de seqüência represente o comportamento de um método. Desta forma, o diagrama chega a representar o nível de instruções, reduzindo drasticamente o nível de abstração desejado por desenvolvedores UML. Porém, ele propõe cinco regras para manipular abstração presentes nos modelos UML que facilitam o processo de estimativa. A abordagem proposta neste trabalho propõe regras de modelagem que interferem pouco na forma como um desenvolvedor poderia especificar seus modelos. Também se evitou acrescentar detalhes nos modelos que pudessem baixar demasiadamente o nível de abstração e, por isso, os métodos não são decompostos em unidades menores, embora possam encapsular outros métodos. Uma limitação da abordagem proposta aqui é que ela não considera as regras propostas por Hoeben, e, portanto, embora seja possível, não trata de questões como herança e interface.

As metodologias Platune, ARTEMIS/SESAME e DESERT, realizam a exploração do espaço de projeto de forma automática. As outras metodologias, mais direcionadas a avaliação do sistema, exigem a intervenção do desenvolvedor para realizar a exploração. Embora a abordagem proposta neste trabalho seja direcionada a exploração do espaço de projeto, ela também depende da interação com o desenvolvedor. Porém, parte do espaço de projeto pode ser explorada automaticamente caso a ferramenta SPEU, proposta neste trabalho, seja integrada a um explorador.

Assim como na metodologia Platune, na metodologia proposta por este trabalho procurou-se desenvolver um estimador que fosse independente do mecanismo de exploração. Deste modo, as informações de interesse do explorador são mantidas no formato padrão da UML, como os próprios resultados das estimativas e o mapeamento entre a aplicação e a plataforma.

Na metodologia SPADE, instruções simbólicas são utilizadas objetivando aumentar o nível de abstração e permitir que diferentes processadores sejam avaliados, sem alterar a descrição da aplicação. Essas instruções são utilizadas em *traces* que capturam o comportamento dinâmico do sistema, além de permitir que os custos para um determinado comportamento sejam conhecidos para diferentes processadores, visto que o custo de uma instrução é fornecido pelo desenvolvedor. De forma similar, este trabalho também apresenta um conjunto de instruções simbólicas. Porém, ao contrário da proposta de SPADE, o desenvolvedor não incorpora as instruções na especificação da aplicação, e as instruções simbólicas são extraídas diretamente da especificação da aplicação. Desta forma, não é necessário criar modelos para a avaliação e para o desenvolvimento.

Para reduzir a complexidade das análises, Russell propõe a definição de cenários de execução. Neste caso, o desenvolvedor deverá restringir um grafo de fluxo de controle extraído da especificação do sistema para determinar somente os cenários mais importantes para a avaliação do sistema. Como somente os cenários mais importantes serão representados nos grafos, a avaliação é mais rápida. Porém, as restrições são especificadas de forma manual diretamente sobre outra representação da aplicação que não a original, gerando mais modelos durante o processo de desenvolvimento. De forma semelhante, na abordagem proposta em SPE/PASA, os cenários mais importantes

podem ser selecionados. Só que estes cenários são especificações UML e, portanto, o mesmo modelo utilizado para o desenvolvimento é utilizado para a avaliação. A proposta deste trabalho também é baseada em cenários especificados em UML, permitindo também que só a porção de interesse do desenvolvedor seja avaliada, sem a necessidade de especificação de novos modelos.

3 ABORDAGEM PROPOSTA

Neste capítulo é apresentada a abordagem proposta de exploração do espaço de projeto de sistemas embarcados baseados em plataforma, através de estimativas realizadas analiticamente com informações extraídas de modelos UML.

3.1 Visão Geral

Nas abordagens tradicionais para a exploração do espaço de projeto em sistemas embarcados, os desenvolvedores devem especificar os modelos da aplicação e definir a arquitetura, a partir dos requisitos e dos componentes disponíveis na plataforma. Após este passo, o mapeamento entre a aplicação e a arquitetura também deve ser especificado. Estas atividades podem ser realizadas com alguns passos de refinamento, como de modelos abstratos (p.ex: UML, Máquinas de Estados, Blocos Funcionais e outros) para o código fonte em alguma linguagem imperativa (p.ex: C ou C++), até que alcance um nível menos abstrato, o nível RTL (*Register Transfer Level*). Somente neste ponto, quando a implementação do sistema está praticamente completa, o desenvolvedor pode estimar as propriedades físicas do sistema, como memória, consumo de energia e tempo de processamento, através de algum processo de simulação. Então, de posse das estimativas, o desenvolvedor pode verificar a necessidade de implementar uma nova solução. A Figura 3.1 ilustra o fluxograma de uma abordagem tradicional.

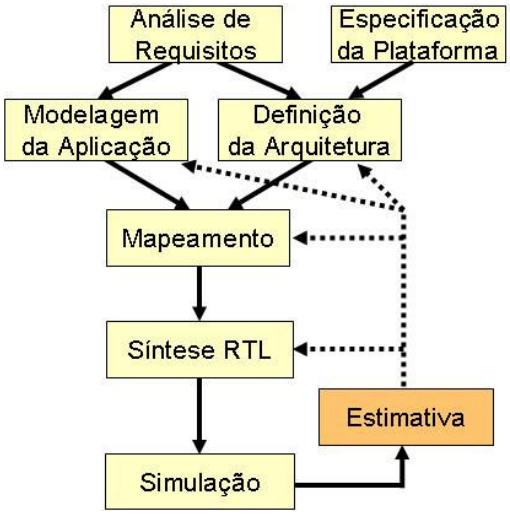

Figura 3.1: Abordagem tradicional para a exploração do espaço de projeto.

A abordagem tradicional apresenta algumas desvantagens. Por ela ser realizada em baixos níveis de abstração, ela apresenta menos oportunidades de otimização. Assim,

como muito tempo foi despendido até a verificação do sistema, é mais custoso corrigir os problemas que podem ser encontrados, além de aumentar o tempo entre as iterações de um processo de exploração de um espaço de projeto.

Neste trabalho, é proposta uma abordagem onde a estimativa é realizada nas etapas iniciais do processo de desenvolvimento, sobre modelos UML. As estimativas são obtidas após o mapeamento entre a aplicação e a arquitetura definida, antes que muito tempo seja despendido na implementação de uma possível solução. A Figura 3.2 ilustra o fluxograma geral para a exploração do espaço de projeto utilizando a abordagem proposta.

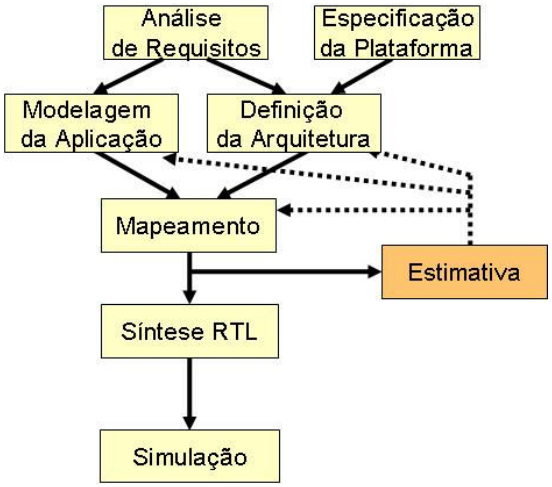

Figura 3.2: Abordagem proposta para a exploração do espaço de projeto.

Esta abordagem apresenta algumas vantagens quando comparada à abordagem tradicional. Com esta abordagem é mais barato encontrar e corrigir problemas, pois menos tempo será necessário até a avaliação da solução proposta para implementação. Como a avaliação é realizada em níveis mais altos de abstração, o espaço de projeto é maior, apresentando mais oportunidades de otimização. Do mesmo modo, avaliar modelos em níveis mais altos de abstração reduz o tempo entre iterações para a exploração do espaço de projeto.

O suporte para exploração do espaço de projeto sobre modelos especificados em UML requer a definição de cinco aspectos:

- As regras de modelagem em UML;
- O modelo que representa uma plataforma, formando um repositório que contenha informações sobre os componentes disponíveis para reuso, o qual será consultado durante o processo de estimativa;
- Os mecanismos para mapeamento entre o modelo de alto nível da aplicação em UML e a plataforma utilizada, para que os custos de um sistema possam ser avaliados;
- O processo de estimativas que permita a avaliação de modelos UML;
- O processo de exploração do espaço de projeto que, através do estimador, gere um espaço de soluções candidatas e selecione a melhor solução para o sistema.

Os aspectos mencionados interferem diretamente no processo de desenvolvimento de sistemas embarcados. Portanto, esta proposta associa estes cinco aspectos às responsabilidades entre os possíveis atores nesta abordagem. A Figura 3.3 apresenta um diagrama de casos de uso que ilustra a distribuição de responsabilidades entre os atores.

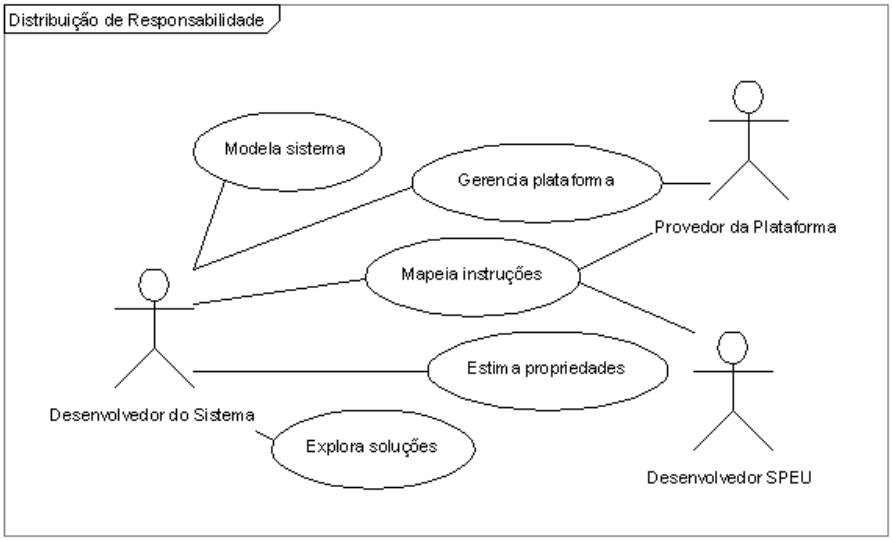

Figura 3.3: Distribuição de responsabilidades na abordagem proposta.

Os seguintes atores são definidos nesta abordagem:

- *Desenvolvedor do Sistema*: Ator que irá construir os modelos UML e deseja explorar o espaço de projeto.
- *Provedor da Plataforma*: Ator que provê (desenvolve, vende e/ou mantém) os componentes que poderão compor o repositório da plataforma, utilizado por desenvolvedores de sistemas.
- *Desenvolvedor SPEU*: Ator que mantém as ferramentas que implementam a abordagem proposta.

Estes atores participam das seguintes atividades:

- *Modela sistema*

  Atores: Desenvolvedor do sistema.

  Descrição: Seguindo o processo de desenvolvimento de sua preferência, o Desenvolvedor do Sistema especifica a estrutura, o comportamento e o mapeamento de sua aplicação, através dos digramas UML. Pode ser utilizada qualquer ferramenta de modelagem que tenha suporte a UML 2.0 e seja capaz de exportar seus modelos em formato XMI (OMG, 2002-c). Porém, o Desenvolvedor do Sistema deve seguir algumas regras durante as atividades de modelagem. Na Seção 3.2 são apresentados detalhes sobre as regras de modelagem necessárias para esta atividade.

  Resultado: Um conjunto de modelos UML que estarão prontos para serem utilizados no processo de estimativa e exploração.

- *Gerencia plataforma*

Atores: Provedor da Plataforma, Desenvolvedor do Sistema.

Descrição: O desenvolvimento de sistemas embarcados baseados em plataforma prevê a utilização de um repositório de componentes reutilizáveis e pré-caracterizados. A pré-caracterização pode ser realizada pelo Provedor da Plataforma e disponibilizada para o desenvolvedor junto com seus componentes. Novos componentes podem ser acrescidos a este repositório à medida que o Desenvolvedor do Sistema implementa novos componentes reutilizáveis, bastando que o Desenvolvedor do Sistema disponibilize no repositório os dados da pré-caracterização dos novos componentes. Detalhes sobre o repositório da plataforma são apresentados na Seção 3.3.

Resultado: Um repositório extensível, com diversos componentes que podem ser reutilizados durante o desenvolvimento para compor um sistema. As informações contidas no repositório, de forma padronizada, serão utilizadas durante o processo de estimativa e exploração do espaço de projeto.

- *Mapeia instruções*

  Atores: Desenvolvedor do sistema, Provedor da Plataforma, Desenvolvedor SPEU.

  Descrição: Antes que as ferramentas de exploração e estimativas sejam utilizadas, as instruções simbólicas, disponíveis na ferramenta de estimativa, devem ser mapeadas para instruções reais caracterizadas (em termos de energia, desempenho e memória) das unidades de processamento disponibilizadas no repositório da plataforma. Este mapeamento pode ser disponibilizado pelo Provedor da Plataforma, junto com a distribuição do processador, pelo Desenvolvedor SPEU, ampliando o suporte da ferramenta, ou pelo próprio Desenvolvedor do Sistema, caso o SPEU não apresente suporte ao processador desejado. Embora este processo de mapeamento seja manual, o SPEU disponibiliza uma ferramenta de auxilio para abstrair o formato dos arquivos. O conjunto de instruções simbólicas e os detalhes sobre o mapeamento são apresentados na Seção 3.4.

  Resultado: O mapeamento de instruções simbólicas gera um arquivo XML para cada mapeamento e descreve quais instruções reais de uma unidade de processamento implementam uma determinada instrução simbólica. Este arquivo contém os custos individuais de cada instrução real e simbólica, em termos de desempenho, energia e memória.

- *Estima propriedades*

  Atores: Desenvolvedor do sistema

  Descrição: O Desenvolvedor do Sistema fornece os modelos UML, no formato XMI, à ferramenta de estimativa. Algumas informações adicionais podem ser fornecidas para a realização das estimativas, as quais são realizadas de forma semi-automática. Esta atividade é descrita em detalhes na Seção 3.5.

  Resultado: As estimativas são disponibilizadas ao Desenvolvedor do Sistema na forma de relatórios ou nos próprios modelos UML.

- *Explora soluções*

Atores: Desenvolvedor do sistema

Descrição: A exploração pode ser realizada de forma manual ou automática. A exploração manual é realizada pelo Desenvolvedor do Sistema, que, de posse dos resultados da estimativa, pode alterar os modelos e efetuar novas estimativas e desta forma explorar algumas soluções. A exploração automática pode ser realizada através da integração do SPEU a uma ferramenta de exploração. Os modelos são disponibilizados para o SPEU realizar as estimativas, enquanto que a ferramenta de exploração altera os modelos UML através de transformações destes modelos. O processo de exploração do espaço de projeto é abordado na Seção 3.6.

Resultado: Um conjunto de pontos de Pareto é disponibilizado ao Desenvolvedor do Sistema. A exploração automática pode apontar as melhores soluções.

A Figura 3.4 ilustra o fluxo de atividades sob o ponto de vista do Desenvolvedor do Sistema.

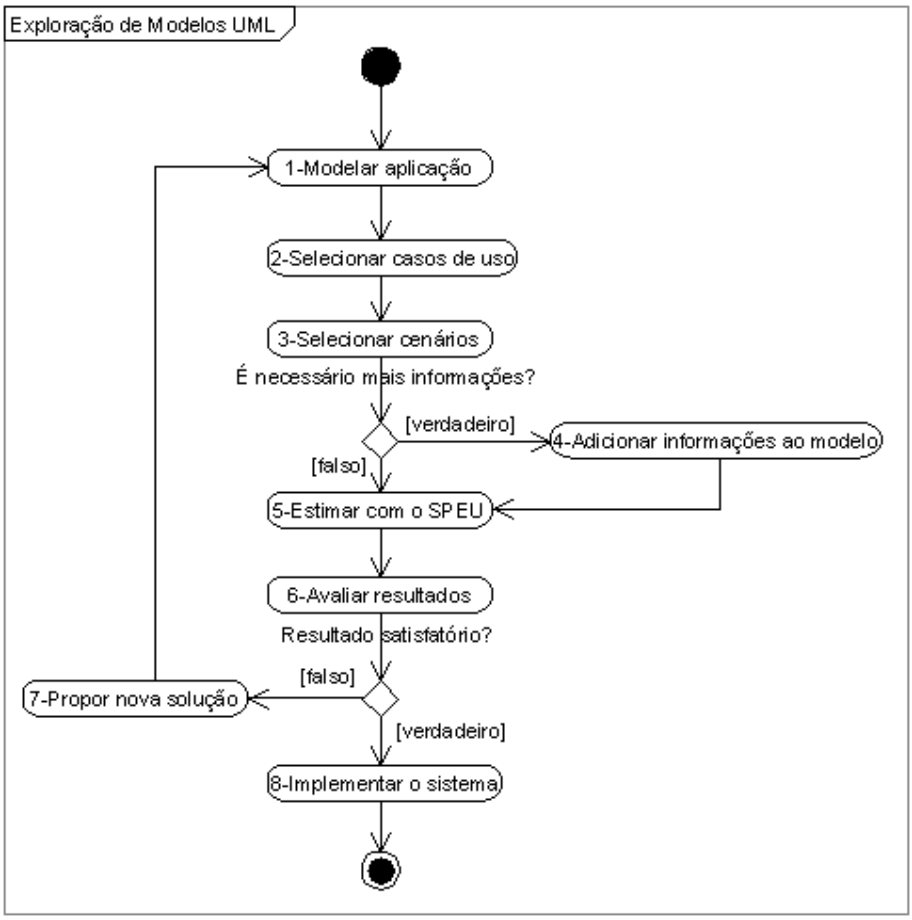

Figura 3.4: Diagrama de atividades para a realização da exploração/estimativa.

As atividades exercidas pelo Desenvolvedor do Sistema são detalhadas a seguir:

1- *Modelar aplicação*: A primeira atividade exercida pelo desenvolvedor, naturalmente, é modelar a aplicação. Esta atividade deve ser realizada com auxilio de uma ferramenta de modelagem, que possua suporte a modelos UML

2.0 e XMI 2.1. Desta forma, os modelos podem ser exportados no formado XMI e utilizados pelas ferramentas de suporte à estimativa e à exploração;

2- *Selecionar casos de uso*: O Desenvolvedor pode selecionar todos ou somente os casos de uso que deseja avaliar;

3- *Selecionar cenários*: Um caso de uso pode conter alguns cenários especificados em diversos diagramas de seqüência. Logo, o Desenvolvedor deve selecionar os cenários que serão utilizados durante a análise. Este passo e o passo anterior são baseados na experiência do Desenvolvedor do Sistema e devem ser um processo de análise de risco. Desta forma, somente os casos de uso e cenários críticos são selecionados, reduzindo o tempo de avaliação e exploração do espaço de projeto.

4- *Adicionar informações ao modelo*: Caso seja necessário, o Desenvolvedor deve inserir mais informações sobre o sistema, complementando os modelos com informações, como o número de execuções de um cenário ou limites inferior e superior de execuções iterativas. Estas informações podem ser adicionadas diretamente nos modelos;

5- *Estimar com o SPEU*: Após as atividades anteriores, o modelo pode ser analisado em termos de estrutura e comportamento utilizando a ferramenta SPEU. Através das informações nos diagramas de classes e seqüência são obtidas as informações estruturais e comportamentais. O mapeamento é extraído do diagrama de implantação e de referências à plataforma contidas nos outros diagramas. Esta ferramenta analisa o modelo e anota os valores estimados de energia, desempenho e memória nos modelos. O processo de estimativa é estático, portanto, não requer simulação, apresentando os valores estimados rapidamente. O estimador pode ser acionado pelo Desenvolvedor do Sistema ou por uma ferramenta de exploração, embora a integração com uma ferramenta de exploração não esteja implementada.

6- *Avaliar resultados*: Após a realização das estimativas, o Desenvolvedor do Sistema pode avaliar os resultados, compará-los com os requisitos do sistema e verificar a necessidade de alterações. Caso o SPEU seja integrado a uma ferramenta de exploração, esta pode avaliar automaticamente os resultados que foram anotados nos modelos;

7- *Propor nova solução*: De posse dos resultados, o Desenvolvedor do Sistema pode retornar aos modelos e alterar a estrutura de classes e a distribuição de responsabilidades. Pode ainda, alterar a interação entre os objetos, incluir outros componentes da plataforma ou alterar o mapeamento dos componentes para outra unidade de processamento. Algumas destas alterações, como o mapeamento entre unidades de processamento, poderia ser realizado por uma ferramenta de exploração;

8- *Implementar o sistema*: Se, após as análises das estimativas, os resultados forem satisfatórios, o Desenvolvedor pode continuar a refinar o sistema seguindo o processo de desenvolvimento. Atividades de estimativa e exploração do espaço de projeto, utilizando análises mais complexas, podem ser realizadas utilizando-se os resultados obtidos com o SPEU ou realizadas em níveis inferiores de abstração, utilizando as abordagens tradicionais até que se homologue a implementação final do sistema.

3.2 Regras para especificação dos modelos UML

A UML 2.0 deve ser utilizada para especificar a aplicação em desenvolvimento, seguindo o método de modelagem UML preferido pelo projetista. Porém, é necessário seguir algumas regras e extensões para que seja possível a realização de estimativas e exploração automática, sem comprometer o nível de abstração. Os diagramas de casos de uso, classe, seqüência e implantação são os diagramas utilizados atualmente na abordagem proposta. A estes diagramas devem ser adicionados os estereótipos e propriedades (*tagged values*) disponíveis no perfil UML-SPT (*UML profile for Schedulability, Performance and Time*), como sugerido em (WEHRMEISTER, 2005), para aumentar as informações sobre os modelos. Outros diagramas podem ser utilizados pelo projetista para especificar o sistema, porém eles não serão considerados por esta abordagem.

Assim como em (WILLIAMS, 2003), esta abordagem é baseada em casos de uso e nos cenários que especificam os mesmos. O projetista deve basear seu desenvolvimento nos casos de uso para que a abordagem proposta seja bem sucedida. A utilização destes como base para o desenvolvimento de sistemas é uma abordagem tradicional, sugerida pela maioria das metodologias de modelagem UML, visto que os casos de uso e cenários especificam a carga de trabalho e os requisitos funcionais e não-funcionais de um sistema. A Figura 3.5 ilustra um diagrama de casos de uso.

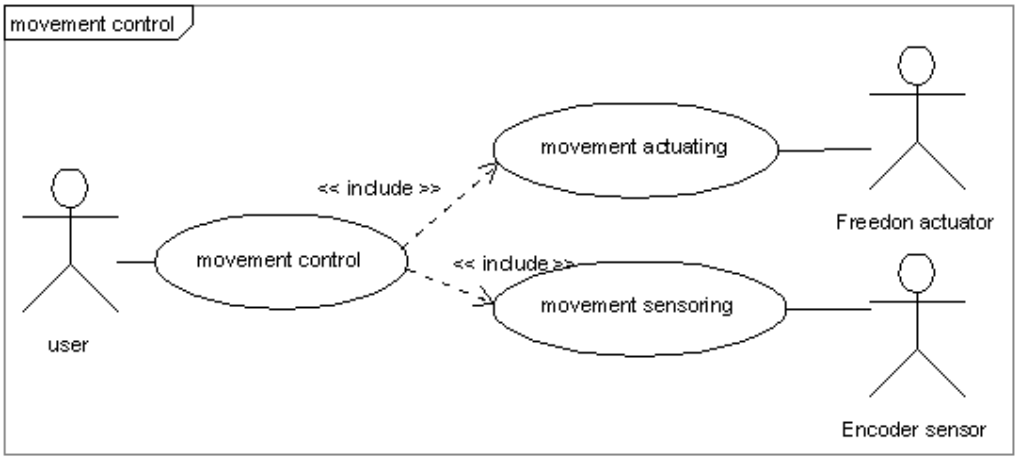

Figura 3.5: Exemplo de um diagrama de casos de uso.

O modelo estrutural da aplicação é especificado em um diagrama de classes. Nestes diagramas, além da definição das classes da aplicação, o projetista deve identificar, sempre que possível, os limites para as multiplicidades de vetores, matrizes e instâncias. Campos de classes que possuam valores conhecidos também devem ser especificados, pois podem ser necessários para extrair os limites inferiores e superiores de execuções iterativas. Além disso, o projetista pode, também, utilizar os estereótipos do UML-SPT para indicar tarefas e recursos. Métodos de acesso (p.ex.: *getValue()* e *setValue(int value)*) são largamente empregados nas abordagens orientadas a objetos e, por isso, é utilizada uma regra para identificar estes métodos. Os métodos de acesso devem utilizar os prefixos "*get*" ou "*set*", para os métodos de leitura e de escrita respectivamente. Além disso, o nome do campo acessado deve suceder o prefixo. O sufixo "*Content*" deve ser utilizado para indicar que o método acessa o conteúdo de uma estrutura, como posições de um vetor ou matriz. A distinção destes métodos é importante, pois eles possuem custos diferentes e são amplamente utilizados. Seguindo estas regras, também

é possível identificar delegação de métodos, recurso esse, muito utilizado para encapsular objetos. A Figura 3.6 ilustra um diagrama de classes, o qual apresenta estas regras.

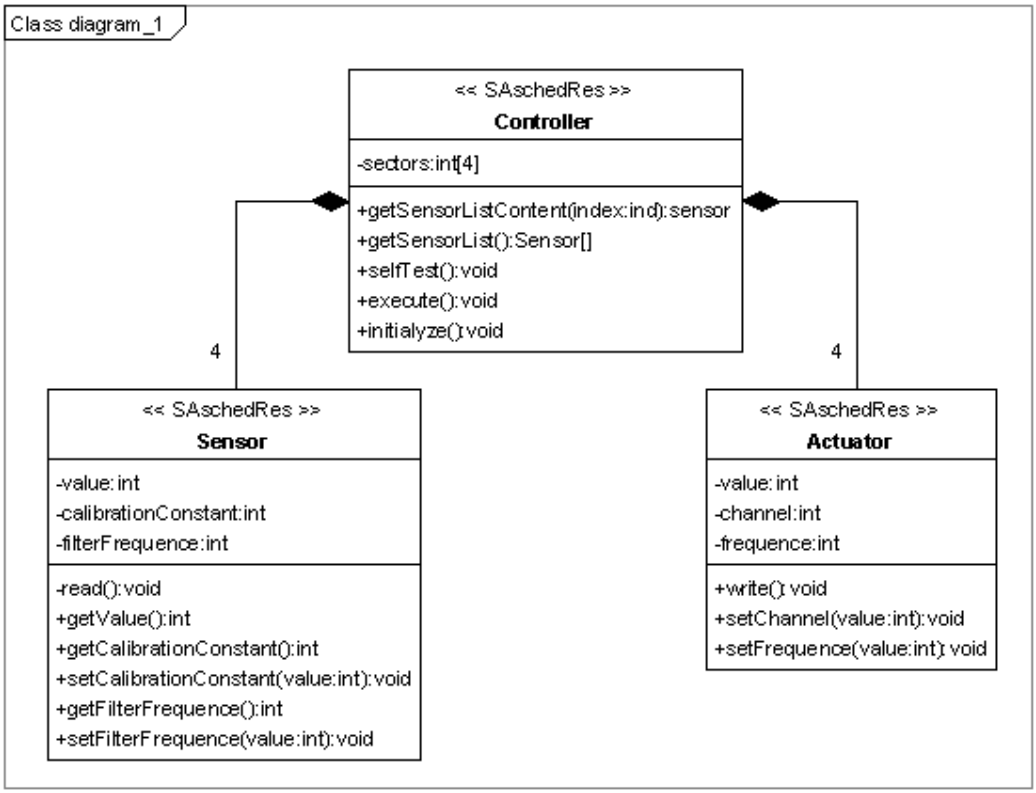

Figura 3.6: Exemplo de um diagrama de classes segundo as regras de modelagem.

Para especificar o modelo comportamental da aplicação o projetista utiliza os diagramas de seqüência, onde serão utilizados alguns dos novos recursos da UML 2.0 para especificar o comportamento da aplicação:

- Execução alternativa: operadores "*alt*" ou "*opt*";
- Execução iterativa: operador "*loop*";
- Referência a outro diagrama de seqüência: operador "*ref*".

A Figura 3.7 ilustra um diagrama de seqüência, onde execuções alternativas são especificadas. Note que estes operadores permitem a especificação de uma condição de guarda (especificada como uma restrição UML através de uma expressão entre colchetes), a qual deve ser satisfeita para que o fragmento ao qual o operador se refere seja executado.

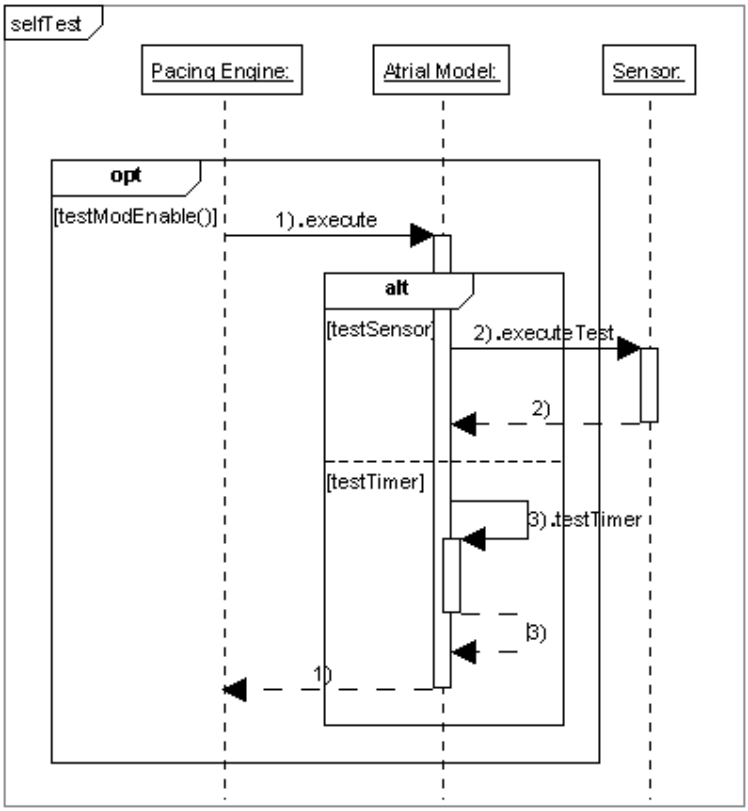

Figura 3.7: Diagrama de seqüência ilustrando os operadores de execução alternativa.

A Figura 3.8 ilustra o operador de execução iterativa utilizado nos diagramas de seqüência, que, assim como os operadores de execução alternativa, também permite a especificação de uma condição de parada. Além disso, os limites inferiores e superiores de execução também podem ser especificados.

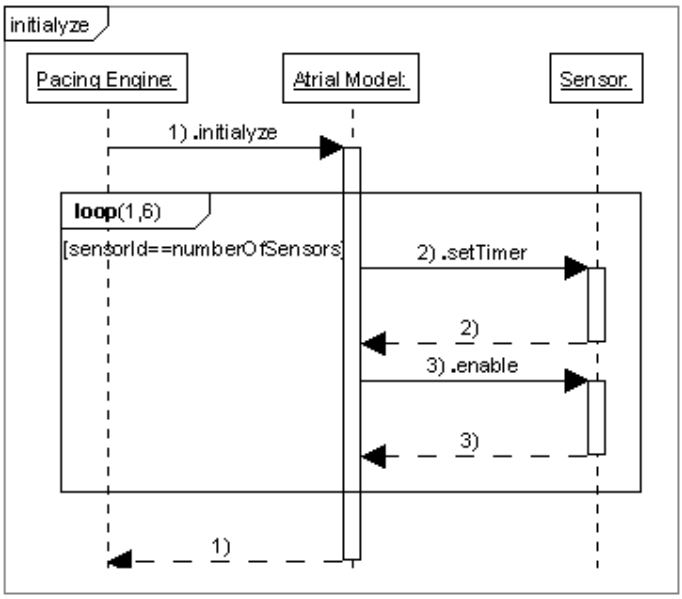

Figura 3.8: Diagrama de seqüência ilustrando o operador de execução iterativa.

A decomposição de diagramas de seqüência utilizando o operador "*ref*" está ilustrada na Figura 3.9.

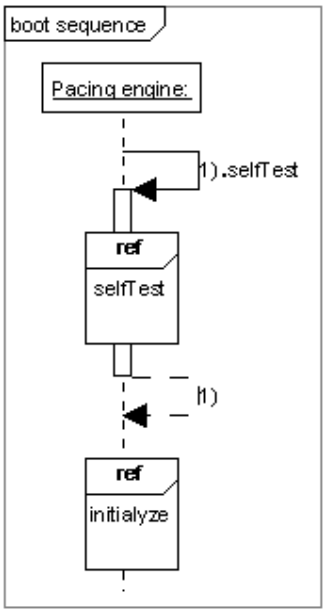

Figura 3.9: Diagrama de seqüência ilustrando o operador de referência.

As interações entre objetos são especificadas neste diagramas através das flechas padrão da UML (indicando a operação, a fonte e o alvo da interação), considerando que o método é a menor unidade de especificação. Um método pode ser considerado uma caixa preta e conter outros métodos, porém um método não pode ser decomposto em instruções através do uso de interações com a semântica de instruções – uma interação deve ser sempre considerada uma invocação de um método. Na prática esta regra poderia ser uma limitação, visto que o projetista não pode inserir mais detalhes da implementação nos diagramas. Porém, isto garante um nível elevado de abstração, motivo pelo quais os modelos são utilizados, além de evitar problemas de semântica entre a definição de métodos e as instruções que os compõem.

Os serviços disponibilizados pela plataforma e requisitados pela aplicação também devem ser especificados nos diagramas de seqüência. A Figura 3.10 ilustra como um projetista pode especificar os serviços requisitados. Há duas opções para especificar o uso de serviços da plataforma. A primeira é utilizando os estereótipos disponíveis no perfil UML-SPT e objetos específicos da plataforma, como um temporizador ou um escalonador, da maneira ilustrada na Figura 3.10 e destacada através de um retângulo. Uma outra forma é a especificação através de uma chamada explícita a um serviço para o objeto reservado, chamado "*platform*", como os serviços de um componente de funções matemáticas (*sin* e *cos*), ilustrado também na Figura 3.10 e destacado por uma elipse. O método invocado deve possuir a mesma assinatura do serviço disponibilizado pela plataforma, através da qual a ferramenta de estimação irá buscar os seus custos. Essa técnica provê um bom mecanismo para abstrair detalhes da plataforma e identificar os componentes reutilizados da mesma.

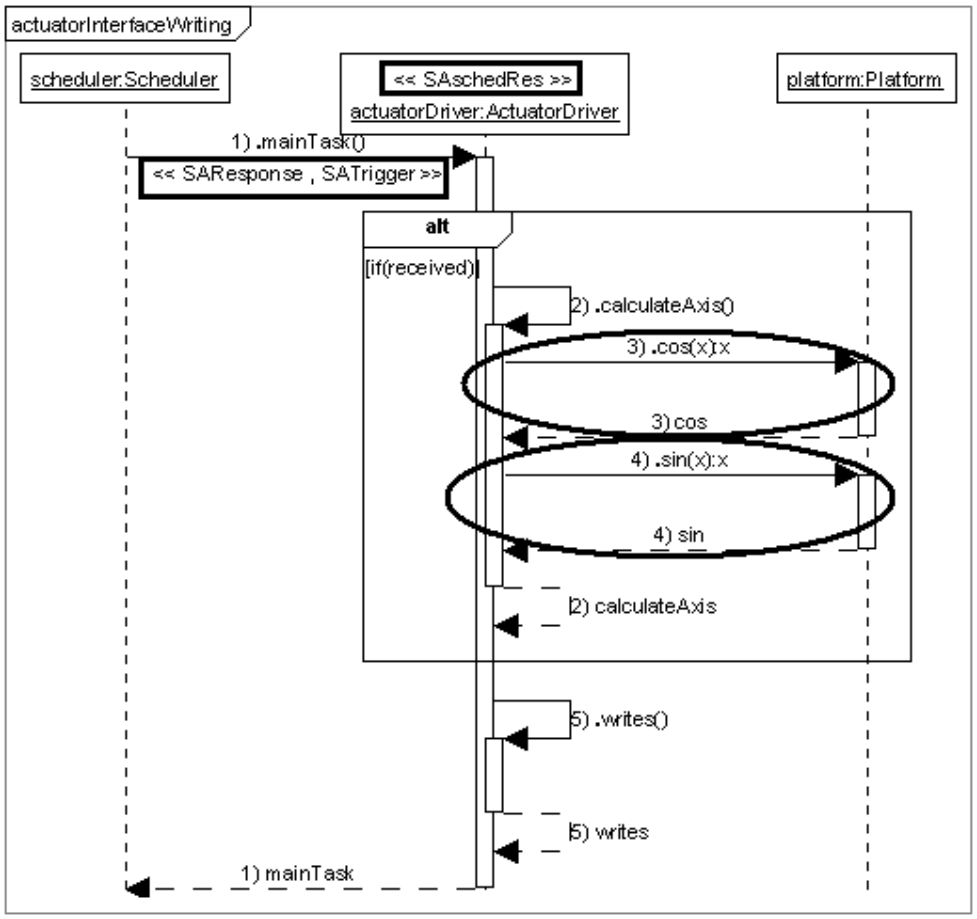

Figura 3.10: Especificação dos serviços requeridos pela aplicação.

Ao especificar o comportamento da aplicação através dos diagramas de seqüência, o Desenvolvedor do Sistema deve atentar também para mais quatro regras que auxiliam na análise dos diagramas:

- Em um diagrama de seqüência pode ser especificado somente o comportamento de uma linha de execução. Neste diagrama, dois ou mais objetos ativos podem interagir, desde que estejam executando dentro da mesma linha de execução;
- Se um cenário é executado condicionalmente, este deve ser referenciado em um diagrama de seqüência de nível superior dentro de uma alternativa de um operador *alt*;
- Cenários independentes, que não são referenciados por nenhum outro cenário, são considerados execuções seqüenciais;
- Tarefas mapeadas para a mesma unidade de processamento serão consideradas seqüenciais.

Através do diagrama de implantação o projetista deve especificar o mapeamento das tarefas para as unidades de processamento disponíveis na plataforma. Como atualmente a abordagem não analisa o custo de comunicação, basta mapear somente as tarefas, visto que os objetos em uma linha de execução são automaticamente mapeados para a unidade de processamento para a qual o objeto ativo presente no diagrama de seqüência

foi mapeado. Nesses diagramas são especificados os tipos e as instâncias de cada unidade de processamento, caso mais de uma unidade esteja presente. Este mapeamento permite identificar o processador e seus serviços disponíveis e os custos associados a estes serviços, incluindo o conjunto de instruções. A Figura 3.11 ilustra um diagrama de implantação.

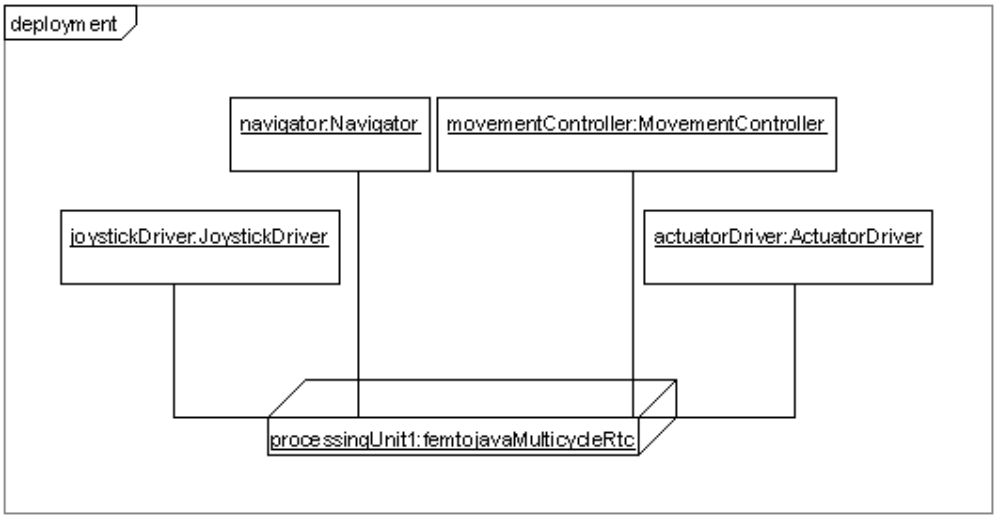

Figura 3.11: Diagrama de implantação ilustrando o mapeamento das tarefas para as unidades de processamento.

3.3 Repositório da Plataforma

Em ambientes de desenvolvimento baseados em plataformas, um grande número de componentes de *software* e de *hardware* estão disponíveis para serem reutilizados durante o desenvolvimento do sistema, formando um repositório de componentes. Para avaliar diferentes soluções em altos níveis de abstração, os componentes reutilizados devem ser pré-caracterizados em termos de desempenho, energia, utilização de memória e outros.

Na abordagem proposta, as informações de pré-caracterização estão armazenadas em um repositório, cujo modelo de dados é baseado no pacote do Modelo Geral de Recursos (*General Resource Model* – GRM) do perfil UML-SPT. No modelo de dados proposto, a caracterização dos componentes da plataforma é especificada em termos de QoS. O modelo de repositório da plataforma pode representar informações sobre componentes de *software* e de *hardware*, podendo ser processadores, API's, sistemas operacionais, componentes de aplicação desenvolvidos pelos usuários do repositório, *drivers* de dispositivos e outros, além de poderem conter informações estruturais e comportamentais sobre estes componentes.

O nível de detalhamento das informações contidas no repositório pode variar de acordo com os objetivos do repositório. Neste trabalho foi utilizada uma representação simples para consulta a respeito dos serviços da plataforma. Desta forma, o modelo da plataforma baseia-se somente nos modelos estáticos de uso de recursos do pacote GRM. Outra simplificação utilizada é a restrição imposta às assinaturas dos serviços, que não devem ser repetidas – os métodos não devem conter mesmo nome, retorno e parametros. Esta simplificação dispensa um modelo complexo de mapeamento, assumindo que o diagrama de implantação, as instruções simbólicas (ver Seção 3.4) e as

referências nos diagramas de seqüência representam as informações de mapeamento necessárias para as estimativas. Porém, esta representação pode ser adicionada a uma representação mais complexa da plataforma, objetivando integrar a abordagem de exploração e estimativa com ferramentas de síntese. A Figura 3.12 ilustra o modelo que representa a plataforma utilizada.

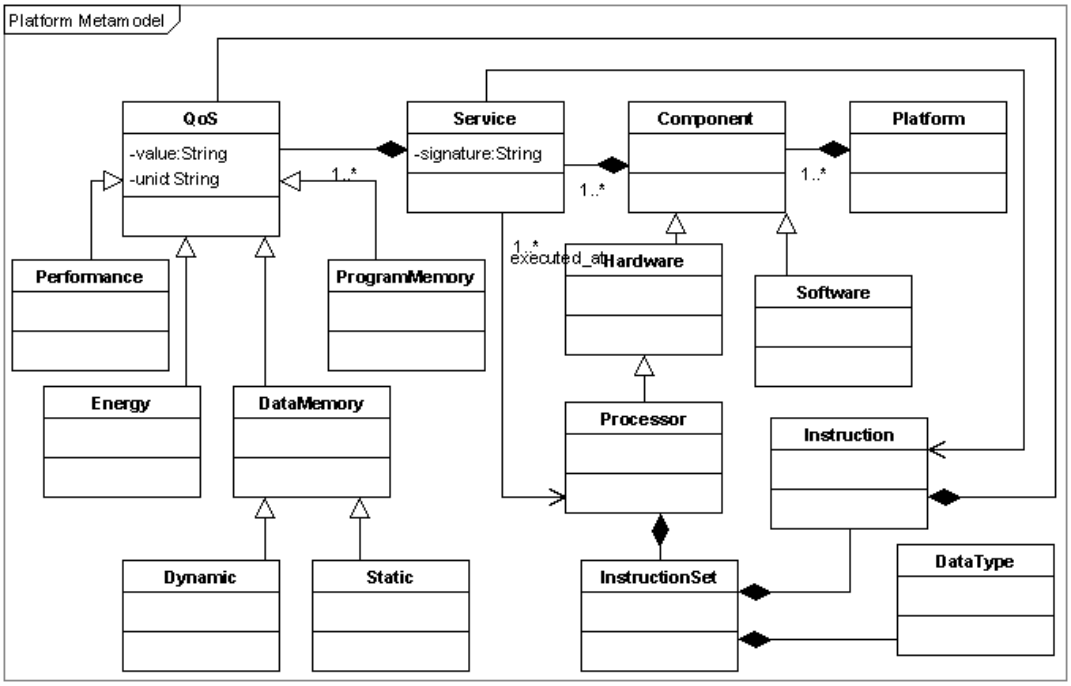

Figura 3.12: Modelo de representação da plataforma.

No modelo de plataforma proposto, aspectos arquiteturais de uma unidade de processamento estão representados, como o conjunto de instruções e os tipos de dados representados. Também são representados os custos para alocação de dados, instruções e memória reservada. Os métodos dos componentes de *software* são representados como serviços, e as informações de QoS são armazenadas segundo as opções de mapeamento para uma unidade de execução.

Atualmente, o repositório da plataforma contém informações sobre três diferentes unidades de processamento, os microcontroladores Femtojava Multiciclo (ITO, 2001), Femtojava Multiciclo RTC (*Real-Time Clock*) (WEHRMEISTER, 2005) e Femtojava *Pipelined* (BECK, 2003). Possui também informações sobre serviços de um *framework* (WEHRMEISTER, 2005) para aplicações de tempo real. Neste, há três diferentes algoritmos de escalonamento, o *Earliest Deadline First* (EDF), o Prioridade Fixa e o *Rate Monotonic*. Há também temporizadores e uma API para especificação de tarefas. O repositório também contém uma biblioteca com funções matemáticas para cálculo de seno, cosseno, arco tangente, raiz quadrada e divisão de inteiros.

O Desenvolvedor do Sistema pode acrescentar novos componentes ao repositório da plataforma, à medida que ele desenvolve novos componentes ou estes são adquiridos de um provedor de componentes. Basta acrescentar as informações sobre a caracterização destes no repositório, para que a ferramenta de estimativa possa usá-los.

3.4 Mapeamento de instruções simbólicas

Nesta proposta, três mecanismos permitem que as informações obtidas nos modelos descritos em alto nível de abstração, especificados em UML, sejam utilizadas para extrair informações de baixo nível de abstração, de modo que as propriedades físicas do sistema possam ser estimadas. O primeiro mecanismo é o diagrama de implantação, que indica qual unidade de processamento irá executar o *software* da aplicação. O segundo mecanismo é a utilização das informações referentes aos componentes pré-caracterizados presentes no repositório da plataforma. A pré-caracterização inclui de forma implícita as informações de baixo nível de abstração, pois são informações influenciadas pela arquitetura da unidade de processamento e tecnologias utilizadas. O terceiro é um conjunto de instruções simbólicas.

O conjunto de instruções simbólicas permite a representação dos serviços oferecidos pela plataforma e do comportamento extraído dos diagramas de seqüência. As instruções simbólicas providas pelo SPEU são baseadas em aplicações orientadas a objetos e incluem instruções para o comportamento comum encontrado nessas aplicações, como leitura e escrita em campos de objetos, invocação, retorno e leitura de parâmetros de métodos, execução iterativa e condicional. Este conjunto pode ser facilmente estendido para incorporar novas instruções que permitam representar o comportamento mais específico, o qual não foi abordado neste trabalho, como instruções de atribuição e de lógica.

Antes que as ferramentas de exploração e estimativas sejam utilizadas, as instruções simbólicas providas pelo SPEU devem ser mapeadas para instruções reais pré-caracterizadas (em termos de energia, desempenho, memória de dados e de programa) das unidades de processamento disponibilizadas no repositório da plataforma. Para realizar este mapeamento, deve-se:

1- Identificar quais instruções reais implementam o comportamento especificado por cada instrução simbólica, caso esta possa ser implementada em uma determinada unidade de processamento;

2- Indicar a forma de cálculo do custo da instrução. Pode-se optar pela forma direta (quando uma instrução simbólica é equivalente a uma instrução real), soma, média ou uma expressão. No ultimo caso, uma expressão é montada com as instruções reais que implementam a instrução simbólica.

3- Inserir a instrução mapeada no conjunto de instruções simbólicas mapeadas. Estas serão armazenadas em um arquivo descrito em XML que conterá as informações do processador utilizado no mapeamento, das instruções simbólicas, das instruções reais que a implementam e a forma de cálculo utilizada para gerar o custo.

A Tabela 3.1 ilustra o mapeamento de algumas instruções simbólicas para o microcontrolador Femtojava Multiciclo RTC e a forma de cálculo de custo utilizada.

Tabela 3.1: Exemplo de instruções simbólicas mapeadas para o microcontrolador Femtojava Multiciclo.

Instruções Simbólicas	*Instruções reais*	*Cálculo de custo*
getDynamicObjectField	aload_0, getfield	soma
interactionStatic	invokestatic	direto
loadParameterByValue	iload_0	direto
condicional	ifgt, ifeq, ifle, ifnonull, if_cmpne, if_cmple, if_cmpge, if_cmpge	média
returnInteger	ireturn	direto
setStaticPrimitiveField,	putstatic	direto

Porém, algumas instruções simbólicas não são mapeáveis para um conjunto de instruções reais. Estas instruções são reservadas para o SPEU e utilizadas para auxiliar em seus processos internos. A Tabela 3.2 apresenta o conjunto de instruções simbólicas reservadas. As instruções "*start*", "*finish*" e "*reference*" são utilizadas para manipular os grafos de instruções simbólicas que representam o comportamento da aplicação. Estes grafos serão detalhados na Seção 3.5.1. A instrução "*method*" é utilizada para referenciar os métodos da aplicação e abstrair as informações que não puderam ser obtidas diretamente pelo estimador, manipulando o método como uma caixa preta. Em futuras extensões da abordagem, o Desenvolvedor do Sistema poderá acrescentar informações sobre este método diretamente nos modelos, para que o SPEU passe estas informações para a instrução "*method*" e seja capaz de extrair mais informações. A instrução "*service*" é utilizada para referenciar um serviço provido pelo repositório da plataforma. Desta forma, alguns serviços podem ser manipulados como uma instrução simbólica, pois fazem parte do comportamento especificado nos diagramas de seqüência.

Tabela 3.2: Conjunto de instruções simbólicas reservadas providas pelo SPEU.

Instruções	*Descrição*
start	Indica o inicio de um grafo de instruções.
finish	Indica o final de um grafo de instruções.
method	Faz referência a métodos da aplicação e abstrai informações internas.
reference	Faz referência a um grafo de instrução, indicando a qual grafo esta instrução pertence, adicionado a este o custo de outro grafo.
service	Faz referência a um serviço provido pelo repositório da plataforma.

A Tabela 3.3 apresenta o conjunto de instruções simbólicas providas pelo SPEU ao Desenvolvedor do Sistema e que devem ser mapeadas para instruções reais.

Tabela 3.3: Conjunto de instruções simbólicas providas pelo SPEU.

Instruções	*Descrição*
conditional	Indica uma execução alternativa
getDynamicObjectField, getDynamicObjectMatrixField, getDynamicObjectMatrixFieldContent, getDynamicObjectVectorField, getDynamicObjectVectorFieldContent	Indica a leitura de um campo do tipo objeto ou do tipo estrutura de dados ou a leitura do conteúdo de uma estrutura. Todos são alocados de forma dinâmica.
getDynamicPrimitiveField, getDynamicPrimitiveMatrixField, getDynamicPrimitiveMatrixFieldContent, getDynamicPrimitiveVectorField, getDynamicPrimitiveVectorFieldContent	Indica a leitura de um campo do tipo primitivo ou tipo estrutura de dados ou a leitura do conteúdo de uma estrutura. Todos são alocados de forma dinâmica.
getStaticObjectField, getStaticObjectMatrixField, getStaticObjectMatrixFieldContent, getStaticObjectVectorField, getStaticObjectVectorFieldContent	Indica a leitura de um campo do tipo objeto ou do tipo estrutura de dados ou a leitura do conteúdo de uma estrutura. Todos são alocados de forma estática.
getStaticPrimitiveField, getStaticPrimitiveMatrixField, getStaticPrimitiveMatrixFieldContent, getStaticPrimitiveVectorField, getStaticPrimitiveVectorFieldContent	Indica a leitura de um campo do tipo primitivo ou tipo estrutura de dados ou a leitura do conteúdo de uma estrutura. Todos são alocados de forma estática.
interactionDynamic, interactionStatic	Indica interação entre objetos (invocação de métodos).
loadParameterByReference, loadParameterByValue	Indica a leitura de parâmetros de um método passados por referência ou por valor.
loopDefined, loopUndefined	Indica a execução iterativa definida ou indefinida.
returnInteger, returnReference, returnVoid	Indica a forma de retorno de um método.
setDynamicObjectField, setDynamicObjectMatrixField, setDynamicObjectMatrixFieldContent, setDynamicObjectVectorField, setDynamicObjectVectorFieldContent	Indica a escrita de um campo do tipo objeto ou do tipo estrutura de dados ou a escrita do conteúdo de uma estrutura. Todos são alocados de forma dinâmica.
setDynamicPrimitiveField, setDynamicPrimitiveMatrixField, setDynamicPrimitiveMatrixFieldContent, setDynamicPrimitiveVectorField, setDynamicPrimitiveVectorFieldContent	Indica a escrita de um campo do tipo primitivo ou tipo estrutura de dados ou a escrita do conteúdo de uma estrutura. Todos são alocados de forma dinâmica.
setStaticObjectField, setStaticObjectMatrixField, setStaticObjectMatrixFieldContent, setStaticObjectVectorField, setStaticObjectVectorFieldContent	Indica a escrita de um campo do tipo objeto ou do tipo estrutura de dados ou a escrita do conteúdo de uma estrutura. Todos são alocados de forma estática.
setStaticPrimitiveField, setStaticPrimitiveMatrixField, setStaticPrimitiveMatrixFieldContent, setStaticPrimitiveVectorField, setStaticPrimitiveVectorFieldContent	Indica a escrita de um campo do tipo primitivo ou tipo estrutura de dados ou a escrita do conteúdo de uma estrutura. Todos são alocados de forma estática.

3.5 Processo de Estimativa baseado em UML

A abordagem proposta para estimar as propriedades de um sistema embarcado baseado em plataforma através de modelos UML permite estimar o desempenho, consumo de energia e memória de dados e de programa. Estas estimativas são resultados de uma análise inicial para o sistema especificado em alto nível de abstração. Logo, esta abordagem fornece informações a respeito das propriedades básicas necessárias a análises mais complexas, como escalonabilidade, utilização de recursos, comunicação e avaliação da qualidade de serviços.

O processo de estimativa é realizado em um conjunto de etapas, partindo da extração de um conjunto de informações dos modelos UML até a anotação dos resultados em uma base de dados. A Figura 3.13 ilustra as etapas do processo de estimativa.

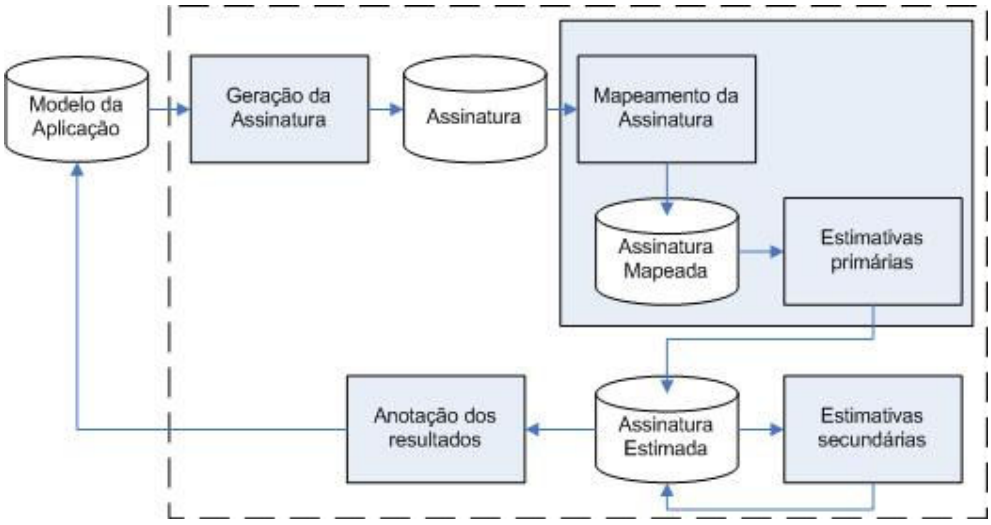

Figura 3.13: Processo de estimativa.

Idealmente, todas as etapas do processo de estimativa deveriam ser automáticas. Os processos e dados dentro da área delimitada pelo retângulo de linhas tracejadas fazem parte do processo de estimativa e poderão ser completamente automatizados. Porém, somente os processos e dados delimitados pelo retângulo de linhas sólidas foram automatizados e, por isso, os processos restantes exigem intervenção do Desenvolvedor do Sistema.

3.5.1 Geração da assinatura

A primeira etapa do processo de estimativa é a geração de um conjunto de informações extraídas dos modelos UML da aplicação, o qual utilizar-se-á para estimar as propriedades do sistema. Este conjunto de informações foi chamado de assinatura da aplicação. Ele contém informações estruturais e comportamentais sobre aplicação e, também, contém as referências aos serviços da plataforma requeridos pela aplicação.

O gerador de assinatura proposto é baseado nas tecnologias envolvidas com a UML 2.0 e apresenta os seguintes requisitos:

1- Uma ferramenta de modelagem deve ter suporte a especificação de modelos segundo a UML /MOF (*Meta Object Facility)* 2.0 (OMG, 2002);

2- Esta ferramenta também deve ser capaz de exportar/importar os modelos UML gerados através do formato XMI (*XML Metadata Interchange*) 2.1, que é capaz de representar modelos na versão UML/MOF 2.0;

3- Uma ferramenta de manipulação de repositórios de meta dados baseados nas especificações UML/MOF 2.0 e XMI 2.1.

Durante o desenvolvimento deste trabalho, um conjunto de tecnologias e ferramentas alternativas foi estudado para implementar o gerador da assinatura. Porém, não foi encontrado um conjunto de ferramentas que empregasse simultaneamente estes três requisitos, pois a proposta da UML 2.0 é recente e poucas ferramentas disponíveis nas comunidades apresentam algum suporte às tecnologias mais recentes propostas pela OMG. Logo, a geração da assinatura deve ser realizada manualmente até que um gerador de assinatura automático possa ser incorporado ao SPEU.

Durante o processo de estimativa, três assinaturas são utilizadas pela ferramenta. A primeira é uma assinatura sem informações específicas sobre uma plataforma, tornando-a independente e, por isso, reutilizável para diferentes mapeamentos entre aplicação e plataforma. No passo seguinte, a etapa de mapeamento, utilizando-se da assinatura gerada diretamente dos modelos UML, cria-se uma cópia e anota-se informações sobre mapeamento e custos, gerando assim uma assinatura mapeada. A terceira assinatura é gerada ao final do processo de estimativa, quando as propriedades estimadas são anotadas em uma cópia da assinatura mapeada, gerando assim a assinatura com os valores estimados.

A primeira assinatura é gerada através da análise dos diagramas de classe e de seqüência. Na assinatura da aplicação, as informações são armazenadas em um arquivo utilizando XML (*Extensible Markup Language*) em um formato baseado em GraphML (*Graph Model Language*) (BRANDES, 2005). As informações comportamentais da aplicação são especificadas sob a forma de grafos de instruções, sendo estes grafos direcionados, onde cada nó é uma instrução simbólica e as arestas indicam o fluxo de execução entre as instruções. Cada grafo de instruções representa um diagrama de seqüência e, além dos nós e das arestas, possui um conjunto de informações que permitem identificar o diagrama de seqüência e a linha de execução à qual eles pertencem. Na assinatura há também um grafo de tarefas, onde os nós correspondem às tarefas especificadas como objetos ativos (através do estereótipo <<*SAschedRes*>> disponível no UML-SPT) nos modelos UML e as arestas representam as dependências entre tarefas. As informações sobre período e *deadline* extraídas dos modelos UML são armazenadas nestes nós. Nas arestas poderiam também ser armazenadas informações sobre o volume de comunicação entre as tarefas, mas neste trabalho a comunicação não é considerada. Além das informações comportamentais, algumas informações estruturais também estão representadas na assinatura, como os conjuntos de objetos, gerados um para cada grafo de instruções, e o conjunto de classes. A Figura 3.14 ilustra o conteúdo das informações contidas em uma assinatura.

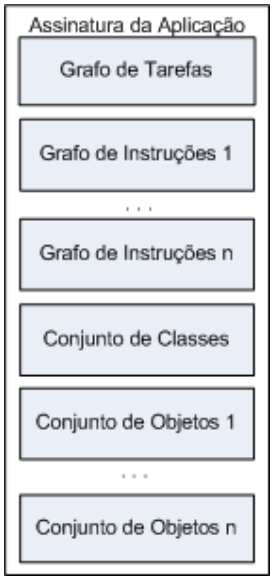

Figura 3.14: Informações contidas em uma assinatura.

A geração da assinatura depende das regras de modelagem apresentadas na Seção 3.2 e do mapeamento entre modelos UML e implementação apresentado por (WEHRMEISTER, 2005). Para gerar o grafo de tarefas, deve ser seguido o algoritmo ilustrado na Figura 3.15. Este algoritmo deve gerar um grafo de tarefas no formato GraphML semelhante ao apresentado na Figura 3.16.

```
tarefas = 0;
para cada Diagrama de Seqüência {
  identificar a linha de execução;
   se(há objeto ativo){
    tarefas = tarefas + 1;
    criar nó(nome_da_tarefa);
    se(encontrado método(run)){
      ler propriedade SATrigger.RTat;
      ler propriedade SAResponse.SAabsDeadline;
    }
    buscar interações;
    para cada interação{
      se(receiver == active){
        criar aresta(sender-receiver);
      }
    }
  }
}
se(tarefas == 0){
    inserir a tarefa "main";
}
```

Figura 3.15: Algoritmo utilizado para gerar o grafo de tarefas.

```
<graph edgedefault="directed" id="taskGraph">
  <node id="taskGraph.task1">
    <data key="name">tarefa1</data>
    <data key="period">15</data>
    <data key="deadline">3</data>
  </node>
  <node id="taskGraph.task2">
    <data key="name">tarefa2</data>
    <data key="period">10</data>
    <data key="deadline">3</data>
  </node>
  <edge id="taskGraph.comm0" source="taskGraph.task1"
        target="taskGraph.task2"/>
</graph>
```

Figura 3.16: Fragmento da assinatura apresentando um grafo de tarefas com duas tarefas, task1 e task2.

Algumas instruções simbólicas são acompanhadas de informações adicionais que devem ser extraídas dos modelos UML no momento da geração da assinatura. A Tabela 3.3 apresenta as instruções, informações adicionais e como elas são extraídas. As instruções getX e setX são referências às instruções simbólicas de leitura e escrita em um campo de objeto, como *getDynamicObjectField* e *setDynamicPrimitiveField* (ver Tabela 3.2).

Tabela 3.4: Informações adicionais associadas às instruções simbólicas.

Instruções	*Informação adicional*	*Fonte no modelo UML*
conditional	Condição de guarda	Condição de guarda disponível no operador *alt*.
interactionDynamic, interactionStatic	Fonte e alvo da invocação	Objeto que envia a mensagem e objeto que recebe mensagem.
loadParameterByReference, loadParameterByValue	Não aplicável.	Não aplicável.
loopDefined, loopUndefined	Limites inferior e superior	Restrição disponível no operador *loop*.
returnInteger, returnReference, returnVoid	Ponto de retorno	Id de uma interação
start, finish	Não aplicável.	Não aplicável.
method, service, getX, setX	Assinatura	Operação associada a uma interação no diagrama de seqüência.
Reference	Id. de um diagrama de seqüência	Referência contida no operador *ref*.

A geração dos grafos de instruções deve seguir algumas regras:

1- As instruções de um grafo de instruções devem pertencer a uma única linha de execução;

2- Cada linha de execução deve ter pelo menos um grafo de instruções;

3- Cada grafo de instruções deve ter a referência ao diagrama de seqüência através do qual ele foi gerado e o nome da tarefa à qual ele pertence;

4- Uma instrução "*start"* deve ser gerada como nó raiz para cada grafo de instruções;

5- Uma instrução "*finish"* deve ser gerada como ultima instrução, para cada grafo. Esta instrução deve possuir arestas de entradas, mas não pode conter arestas de saída.

A figura 3.17 ilustra o fragmento de um grafo de instruções obtido quando são os passos do algoritmo ilustrado na Figura 3.18 são seguidos.

```
<graph edgedefault="directed" id="ig1">
  <data key="scenario">doSomething</data>
  <data key="thread">task1</data>
  <node id="ig1.i0">
    <data key="name">service</data>
    <data key="signature">restoreContext</data>
  </node>
  <node id="ig1.i1">
    <data key="name">interactionDinamic</data>
    <data key="source">task1</data>
    <data key="target">aObject</data>
  </node>
  <node id="ig1.i2">
    <data key="name">method</data>
    <data key="signature">void methodSignature(int value)</data>
  </node>
  . . .
  <edge id="ig.e1" source="ig1.i0" target="ig1.i1"/>
  <edge id="ig.e2" source="ig1.i1" target="ig1.i2"/>
  . . .
</graph>
```

Figura 3.17: Fragmento de uma assinatura ilustrando um grafo de instruções e três instruções - *service*, *interactionDynamic* e *method*.

```
para cada Diagrama de Seqüência {
  criar grafo_de_instrucoes(nome_diagrama);
  identificar a linha_de_execução;
  grafo_de_instrucoes.nome=linha_de_execucao.nome;
  criar nó(start);
  buscar interações;
  para cada interação{
    se tipo == alt || tipo == opt
      criar nó(conditional);
      conditional.guard = condição de guarda;
    }

    se tipo == loop{
      se(há condição de parada){
        criar nó(definedLoop);
        definedLoop.lowerBond = minint;
        definedLoop.upperBond = maxint;
      }
      senão
        criar nó(undefinedLoop);
        undefinedLoop.guard = condição de parada;
        undefinedLoop.lowerBond = minint;
        undefinedLoop.upperBond = maxint;
      }
    }
    se tipo == ref{
      criar nó(reference);
      reference.sequenceDiagram=Diagrama de Seqüência atual
    }
    se tipo == message{
      se operation == static{
        criar nó(interactionStatic);
        interactionStatic.source = messageSender;
        interactionStatic.target = messageReceiver;
      }
      senão{
        criar nó(interactionDynamic);
        interactionDynamic.source = messageSender;
        interactionDynamic.target = messageReceiver;
      }
      se messageReceiver == platform{
        criar nó(service);
        service.signature = operation;
      }
      senão{
        criar nó(method);
        method.signature = operation;
      }
    }
  }
  criar nó(finish);
}
```

Figura 3.18: Algoritmo utilizado para gerar o grafo de instruções.

Um conjunto de objetos é gerado para cada diagrama de seqüência. Para obter este conjunto basta listar os objetos presentes nestes diagramas, especificando a multiplicidade e a classe à qual eles pertencem.

O conjunto de classes é a listagem das classes presentes na aplicação. Este conjunto é utilizado como referencia aos custos individuais de cada classe durante o processo de estimativa.

A assinatura gerada sofrerá transformações durante o processo de estimativa, onde novas informações são acrescentadas, como os custos das instruções simbólicas e dos serviços após o mapeamento.

3.5.2 Mapeamento da assinatura

A assinatura da aplicação gerada no passo anterior não contém informações de uma plataforma específica. Portanto, esta assinatura deve ser mapeada para o conjunto de componentes da plataforma alvo, para que seus custos sejam estimados. O mapeamento desta assinatura é realizado em duas etapas.

A primeira etapa analisa o diagrama de implantação, que contém as informações de que porção do *software* está alocada para quais unidades de processamento. Desta forma, identifica-se o conjunto de instruções simbólicas (previamente mapeado para as instruções reais) que deverá ser utilizado nas estimativas. Cada grafo de instruções simbólicas, gerado segundo o processo descrito na Seção 3.4.1, pode utilizar um conjunto de instruções diferentes, de acordo com o mapeamento utilizado no diagrama de implantação.

A segunda etapa percorre os grafos de instruções simbólicas e busca seus custos no conjunto de instruções simbólicas mapeado. De forma similar, quando instruções do tipo "*service*" são encontradas na assinatura, uma busca pelo referido serviço é realizada no repositório da plataforma, que tem seus custos anotados na assinatura para serem tratados como qualquer outra instrução simbólica.

Ao final das duas etapas, uma nova assinatura é gerada. Nessa assinatura os custos das instruções simbólicas mapeadas estão anotados e podem ser utilizados para a realização das estimativas.

3.5.3 Estimativa primária

Após o mapeamento da assinatura inicia-se o processo de estimativa. Estimativas de memória, desempenho e energia são calculados para cada cenário através do processo de estimativa primária. O SPEU calcula as estimativas de duas formas, dependendo se o Desenvolvedor do Sistema conhece ou não o melhor e pior caminho de execução.

No primeiro caso, os cenários são explicitamente modelados seguindo o melhor e o pior caso de execução. Desta forma, o SPEU calcula as estimativas apenas adicionando os custos de desempenho, energia e memória. Estes resultados são anotados na assinatura para cada grafo de instruções.

No segundo caso, a ferramenta SPEU pode calcular o melhor e pior caso de execução considerando as estimativas como um problema de otimização. A ferramenta utiliza um algoritmo de Programação Linear Inteira (*Integer Linear Programming* – ILP) para analisar os grafos de instruções, assim como na metodologia CINDERELLA. Esta estimativa é estática e, portanto, não requer simulação. Através do ILP, são realizadas estimativas de desempenho e energia. A estimativa de memória é realizada após os resultados do ILP quando o melhor e o pior caminho de execução já são conhecidos.

O melhor e o prior caso de execução são obtidos através da minimização ou maximização da expressão linear apresentada na Equação 3.1. Nesta expressão, c_i é o custo da instrução simbólica S_i, que é constante para qualquer momento em que a instrução S_i seja executada, x_i é número de execuções dessa instrução simbólica, que deve ser um número inteiro positivo, e N é o número de instruções simbólicas contidas em um grafo de instruções. O custo resultante C pode ser calculado tanto para o desempenho medido em ciclos quanto para a energia medida em atividade de chaveamento de portas.

$$C = \sum_{i=1}^{N} c_i x_i \tag{3.1}$$

Restrições lineares são utilizadas para representar restrições estruturais e funcionais que podem ser impostas sobre o grafo de instruções analisado. As restrições estruturais são extraídas de forma automática, a partir da estrutura do grafo de instruções, e não requerem informações adicionais anotadas nos modelos UML. As restrições funcionais são extraídas dos modelos UML através das informações adicionais. Estas são anotadas como restrições nos operadores disponibilizados pela UML, como o limite inferior e superior especificados em um operador *loop* ou restrições indicando informações sobre o caminho de execução em um operador *alt*. Então, as restrições especificadas nos modelos são transformadas em restrições lineares quando o problema a ser resolvido é formulado. Um adaptador Java (EBERT, 2004) para a biblioteca *lp_solve* (LP, 2005) é utilizado para formular e solucionar o problema de otimização através do método *Simplex*.

Inicialmente, as estimativas são calculadas para cada gafo de instruções referenciados em outros grafos. Após a realização das estimativas para os grafos de hierarquia mais baixa, os valores estimados são anotados nos grafos para os quais as estimativas foram realizadas e nas instruções *reference* dos grafos de hierarquia mais alta. Então, o processo é repetido novamente até que não haja mais dependências entre os grafos de instruções. Neste ponto, os grafos contêm os valores de desempenho e energia para o melhor ou pior caso de execução.

Para estimar a memória de dados são analisados os custos dos serviços reutilizados e os campos dos objetos pré-alocados segundo a especificação do diagrama de classes. Depois disso, o valor máximo de memória alocada dinamicamente para a invocação de métodos é extraído do grafo de instruções, quando o melhor e pior caminho de execução já são conhecidos, seja através do ILP ou informado pelo Desenvolvedor do Sistema. Então esse valor é adicionado ao valor de memória pré-alocada. Para estimar a memória de programa é gerada uma listagem das instruções para cada método e serviço, mas evitando incluir métodos repetidos. Após este processo os custos de memória de programa para todas as instruções e serviços são somados.

3.5.4 Estimativa secundária

Durante o processo de estimativa primária, os custos individuais de cada grafo de instruções foram calculados. Também foi calculado o caminho de execução para o pior e o melhor caso para cenários que apresentam dependência entre si. No processo de estimativa secundária, os custos dos cenários independentes são adicionados para chegar à estimativa de custo para todos os cenários selecionados pelo Desenvolvedor do Sistema. Durante este processo são realizadas comparações entre cenários, para evitar o

cálculo de informações replicadas, como objetos e serviços compartilhados por cenários diferentes.

3.5.5 Anotação dos resultados

Ao final do processo de estimativa, os resultados são anotados na assinatura mapeada, completando assim todas as informações que o processo de estimativa fornece. Porém, a assinatura é um formato interno do SPEU e os resultados das estimativas devem ser também anotados de volta nos modelos UML. Desta forma, as informações obtidas pelo SPEU podem ser utilizadas por outras ferramentas e visualizadas pelo Desenvolvedor do Sistema na ferramenta de modelagem utilizada para gerar os modelos.

O processo de anotação dos resultados nos modelos UML, do mesmo modo que o de geração da assinatura, não está automatizado. Por este motivo, o Desenvolvedor do Sistema deve manualmente inserir os resultados gerados pelo SPEU nos modelos para, assim, dar continuidade ao processo de desenvolvimento.

3.6 Processo de Exploração do Espaço de Projeto em UML

Quando modelos UML podem ser avaliados quantitativamente, novas oportunidades de otimização no nível da aplicação podem ser aproveitadas, como a distribuição de responsabilidade, as interações entre objetos, a interação com serviços da plataforma ou a forma de alocação dos dados (alocação estática ou dinâmica), aumentando, assim, o espaço de projeto a ser explorado. Mais ainda, oportunidades de otimização já conhecidas, referentes ao mapeamento entre aplicação e plataforma, como alocação de tarefas em processadores, número de processadores utilizados ou distribuição de processadores em uma estrutura de comunicação, podem ser exploradas nos estágios iniciais de desenvolvimento com relativa precisão.

O processo de exploração do espaço de projeto proposto nessa abordagem permite ao Desenvolvedor do Sistema avaliar tanto o impacto de suas decisões de modelagem, quanto como a aplicação se comportará quando executada em diferentes processadores. Inicialmente, este processo é manual, mas parte deste processo pode ser automatizada, integrando o SPEU a uma ferramenta de exploração automática do espaço de projeto.

Já nos estágios iniciais do desenvolvimento, o Desenvolvedor do Sistema deve tomar inúmeras decisões quanto ao projeto. Uma das primeiras decisões é a arquitetura de *software* utilizada, onde o Desenvolvedor do Sistema deve selecionar:

1- Quais serão as responsabilidades de cada objeto;

2- Que informações devem ser abstraídas entre os objetos – determinação do encapsulamento e delegação de responsabilidade;

3- Quantas classes e objetos devem implementar a solução;

4- Quais objetos devem ser implementados de forma estática ou dinâmica;

5- Quantas tarefas devem ser utilizadas;

6- Como os objetos devem interagir;

7- Quais os serviços disponíveis em uma plataforma precisam ser utilizados;

8- Como os objetos da aplicação irão interagir com os objetos da plataforma.

Nesta abordagem, sugere-se que para um conjunto relevante de casos de uso, as soluções alternativas sejam avaliadas. Desta forma, explorando diversos modelos, o Desenvolvedor do Sistema poderia otimizar os cenários críticos para a execução do sistema de forma que este alcance os requisitos.

Porém, as diversas possibilidades podem ser combinadas montando um vasto espaço de projeto a ser explorado. Além disso, o Desenvolvedor do Sistema deve procurar obter ganhos em atributos normalmente contraditórios, como facilidade de alterações, reusabilidade, escalabilidade, segurança, confiabilidade e maturidade, além dos já considerados desempenho e baixo consumo de energia, de memória e de outros recursos.

Para facilitar o processo de exploração, propõe-se que uma ferramenta de exploração automática integrada ao estimador SPEU, tal como disponível nos ambientes PLATUNE, ARTEMIS e DESERT, discutidos no Capítulo 2, altere o diagrama de implantação para obter novos conjuntos de soluções. As novas soluções podem ser obtidas alterando o mapeamento de tarefas às unidades de processamento e o numero e os tipos destas unidades.

4 ESTUDOS DE CASO

4.1 Visão geral

Neste capítulo é apresentado um cenário de exploração do espaço de projeto de forma manual, onde quatro modelos alternativos são propostos para um sistema de controle eletrônico do movimento de uma cadeira de rodas. Além disso, as estimativas obtidas para realizar a exploração do espaço de projeto foram comparadas aos resultados exatos obtidos através do simulador CACO-PS (BECK, 2003), um simulador de energia com precisão de ciclo. Para cada solução a ferramenta de síntese chamada SASHIMI (ITO, 2001) foi utilizada para obter a descrição do *hardware* e os *byte codes* Java para a aplicação.

4.2 Reuso de componentes da plataforma

Para implementar o controle de movimentos, diversos componentes providos pela plataforma podem ser reutilizados. Nas quatro soluções alternativas, a unidade de processamento utilizada foi o Femtojava Multiciclo RTC (WEHRMEISTER, 2005). Este é um microcontrolador baseado em uma máquina de pilha, com um conjunto reduzido de instruções que suporta a execução nativa de 68 *byte codes* Java.

Para resolver as equações utilizadas no controle de movimentos, um componente com funções matemáticas foi também utilizado nas quatro soluções alternativas. As funções utilizadas desse componente são: seno, cosseno, arco tangente, divisão de inteiros e cálculo da raiz quadrada.

Para especificar o comportamento de tempo real na primeira, terceira e quarta soluções, foi utilizado um *framework* (WEHRMEISTER, 2005) que possui uma API para especificação de tarefas e componentes de um sistema operacional de tempo real. Neste *framework*, três diferentes algoritmos de escalonamento estão disponíveis, porém os experimentos utilizaram apenas o escalonador EDF. Consequentemente, os aspectos da plataforma utilizados pelo estimador também consideram os custos relacionados ao escalonador de tempo real.

A Tabela 4.1 ilustra a caracterização de alguns serviços utilizados nos experimentos. Nesta tabela, os valores de memória são apresentados em *bytes* e o desempenho em ciclos. A energia consumida é avaliada em termos da atividade de chaveamento de portas, assim como em (TALARICO, 2005 e BECK, 2003). Os valores de desempenho, energia e memória, para os serviços apresentados nessa tabela, são constantes para o melhor e pior caso.

Tabela 4.1: Resultados parciais da caracterização de alguns serviços da plataforma utilizados no estudo de caso.

Componente	*Serviço*	*Memória de Programa (bytes)*	*Memória de Dados (bytes)*	*Desempenho (ciclos)*	*Energia (chaveamento de portas)*
Math	sin	14	4	53	73.815
EDFScheduler	restoreContext	4	0	883	8.117.388
RealtimeThread	waitForNextPeriod	243	13	5721	1.254.301
Femtojava Multiciclo RTC	int_tf0	2	32	124	187.453

4.3 Experimentos

4.3.1 Apresentação

A aplicação considerada no estudo de caso consiste em um sistema embarcado de tempo real dedicado à automação e controle de uma cadeira de rodas inteligente. Esta cadeira de rodas possui diversas funções, como controle de movimentos, detecção de possível colisão, navegação, planejamento de caminho baseado em uma referência alvo, controle de bateria, supervisão do sistema, agendamento de tarefas e movimento automático. Os experimentos foram direcionados somente aos casos de uso do controle de movimentos, principal função do sistema que incorpora restrições de tempo real. O diagrama de casos de uso simplificado está ilustrado na Figura 4.1.

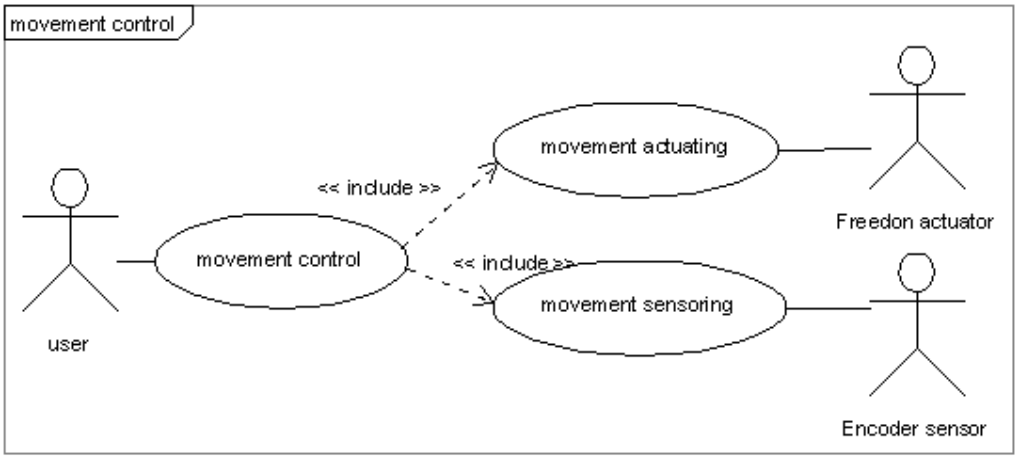

Figura 4.1: Diagrama de casos de uso da cadeira de rodas.

A aplicação, em todas as soluções candidatas, foi mapeada para uma única unidade de processamento, o Femtojava Multiciclo RTC, para assim concentrar os experimentos no espaço de projeto que pode ser explorado quando um projetista considera diferentes modelos UML para a mesma aplicação. Não são, portanto, explorados diferentes mapeamentos entre a aplicação e a plataforma.

O caso de uso *movement controller* tem duas responsabilidades: verificar o movimento, adquirindo informações sobre a velocidade e o ângulo (caso de uso

movement sensoring), e atuar nos motores da cadeira de rodas para gerar o movimento (*movement actuating*). Para obter os valores de velocidade e ângulo, dois sensores são utilizados, um para cada roda. O sensoriamento adquire os valores obtidos dos sensores e os converte em velocidade e ângulo, informações necessárias para outros componentes do sistema, como o monitor que apresenta as informações para o cadeirante e o sistema de navegação que controla automaticamente o movimento. Para atuar, o sistema recebe os valores através de um controlador manual, uma interface de comando vocal ou, ainda, de um outro sistema. Neste experimento, um controlador manual é utilizado, mas o projeto do controle de movimentos deve prever a utilização de outras interfaces.

Os valores obtidos pelo controlador manual são informações referentes a dois eixos ortogonais x e y. Estes valores podem ser escritos diretamente no atuador ou convertidos em uma forma padrão de representação. Através desta, o componente de controle de movimento pode ser utilizado junto com outros componentes, com uma interface padronizada que utilize os mesmo tipos de dados de entrada e saída, como informações de velocidade e ângulo. Essas duas alternativas foram exploradas nas diferentes soluções modeladas neste experimento. Outro ponto explorado foi a flexibilidade do modelo em termos de número de tarefas, encapsulamento e interação entre os objetos.

4.3.2 Primeira solução

No projeto da primeira solução de modelagem, optou-se por converter os valores obtidos pelo controlador manual em velocidade e ângulo. A abordagem utilizada para modelar esta solução utiliza granularidade fina para a distribuição de responsabilidades – as responsabilidades foram distribuídas entre um grande número de objetos e tarefas. Esta abordagem é flexível, facilitando o reuso de componentes e a evolução do sistema. Desta forma, é mais fácil estendê-la e inserir novos componentes. As classes utilizadas na primeira solução podem ser observadas no diagrama de classes ilustrado na Figura 4.2. As classes estão divididas em quatro pacotes:

- *userInterface*, inclui a classe *JoystickDriver*, responsável pela obtenção dos valores do controlador manual;
- *movementControl*, contém as classes: *MovementControl*, que realiza a leitura dos sensores através da classe *SensorDriver* que possui duas instâncias (sensor1 e sensor2), cada uma responsável pela leitura de um sensor instalado em cada roda; *ActuatorDriver*, responsável por escrever os valores dos eixos x e y no atuador;
- *navigation*, possui a classe *Navigator*, responsável pela apresentação dos valores de velocidade e ângulo ao cadeirante e pelos algoritmos de navegação utilizados nos outros casos de uso;
- *wheelchair*, contém as classes *Wheelchair* e *TimeObjects*, utilizadas na inicialização do sistema.

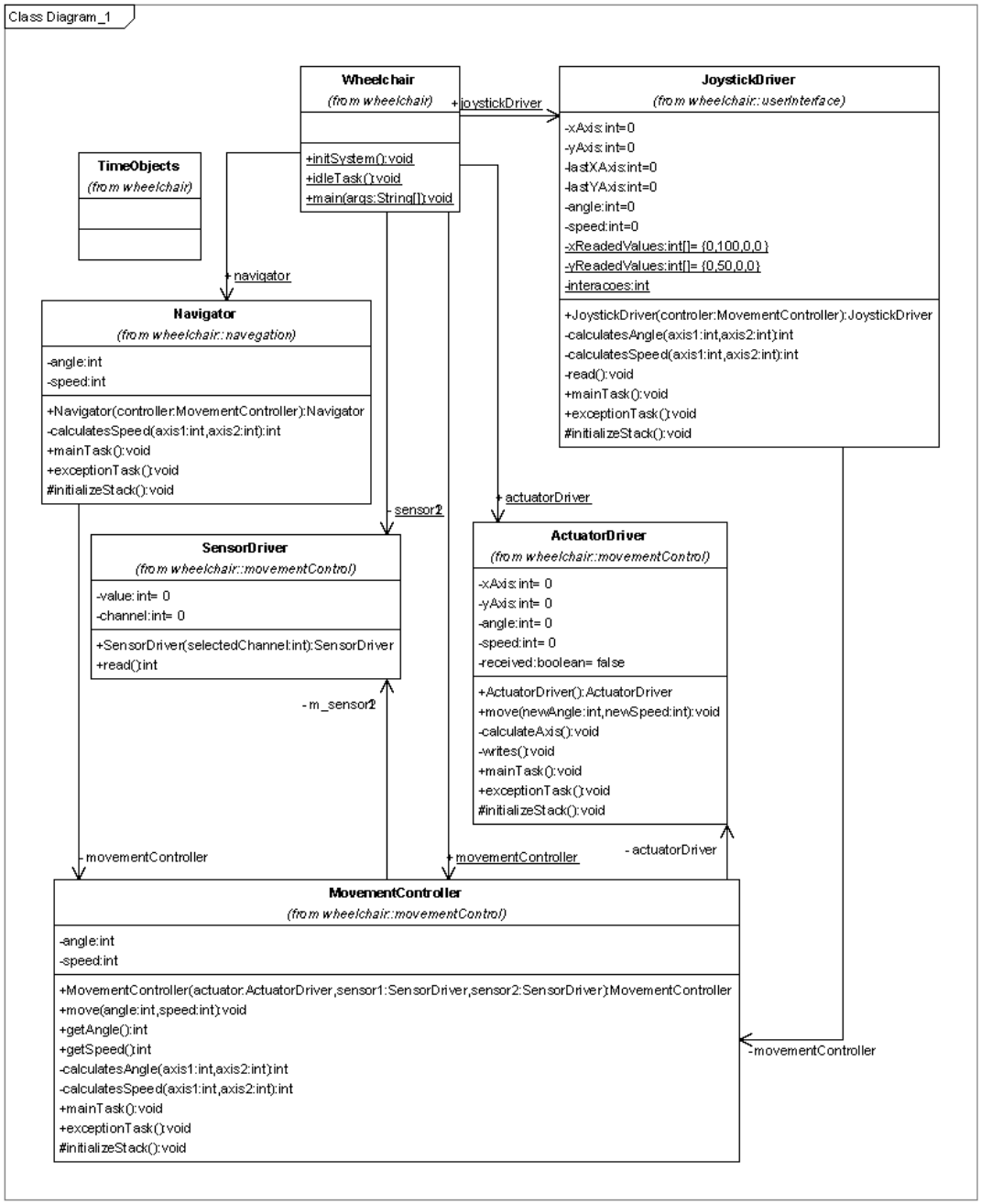

Figura 4.2: Diagrama de classes para o caso de uso *movement controller* na primeira solução.

Duas tarefas foram utilizadas para implementar o caso de uso *movement actuating*. A primeira tarefa deve ler o controlador manual, converter os valores obtidos em ângulo e velocidade e requisitar o movimento. A segunda tarefa, caso receba uma notificação de movimentos, calcula os valores dos eixos x e y e escreve na interface do atuador. O diagrama de seqüência para cada uma delas está ilustrado nas Figuras 4.3 e 4.4. Duas outras tarefas foram utilizadas para implementar o caso de uso *movement sensoring*, e o diagrama de seqüência para cada uma é mostrado nas Figuras 4.5 e 4.6. A primeira tarefa realiza a leitura dos valores dos sensores dispostos nas rodas, calcula os valores de ângulo e velocidade atuais e os armazena, permitindo que estes sejam consultados

pelos demais componentes. A segunda tarefa realiza uma consulta periodicamente aos valores disponibilizados de ângulo e velocidade.

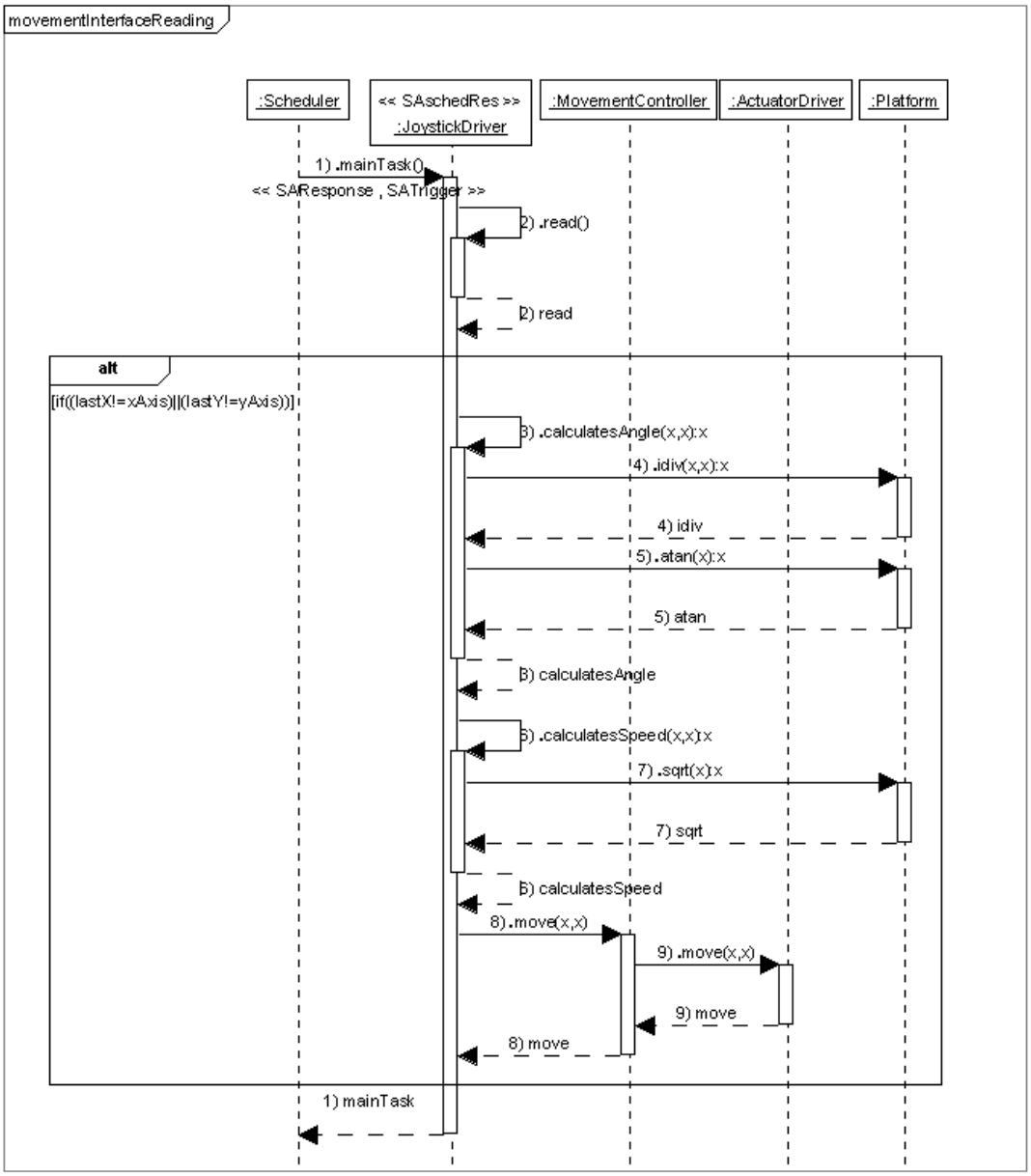

Figura 4.3: Diagrama de seqüência 1 para o caso de uso *movement actuating* na primeira solução

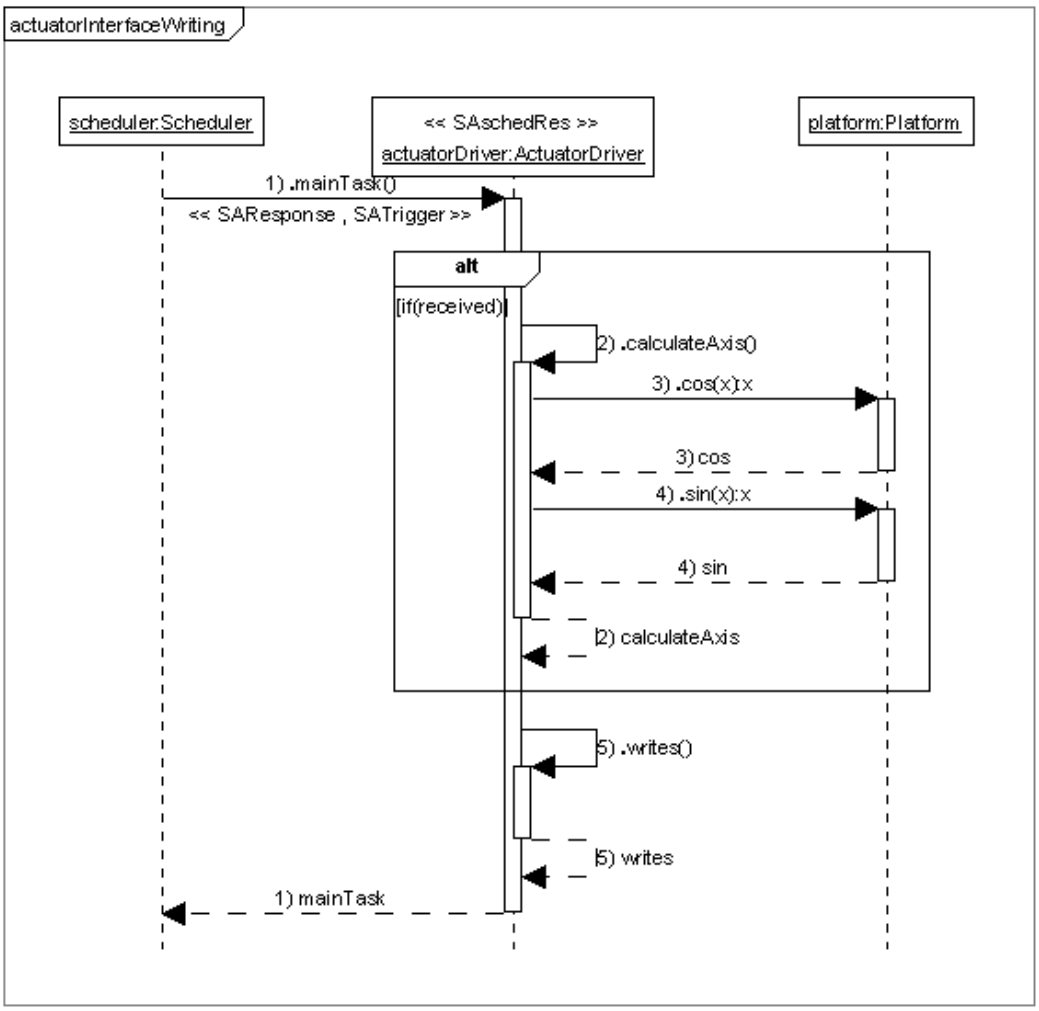

Figura 4.4: Diagrama de seqüência 2 para o caso de uso *movement actuating* na primeira solução.

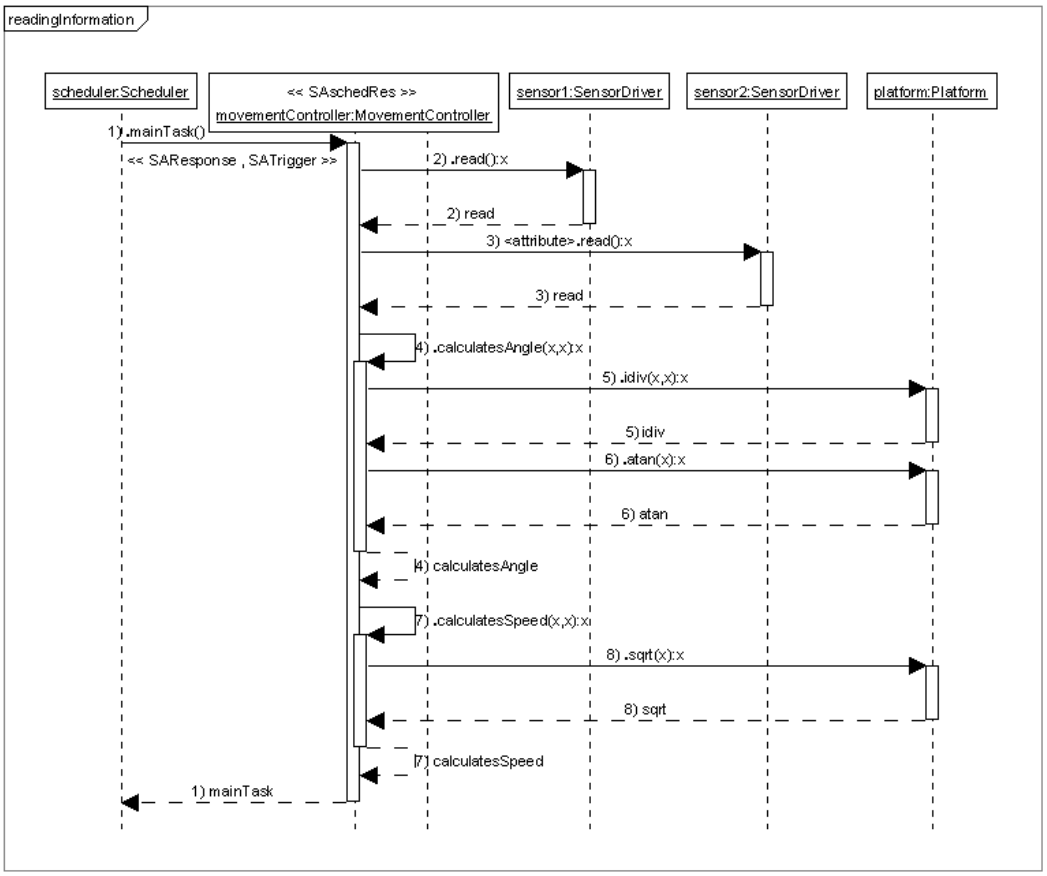

Figura 4.5: Diagrama de seqüência 1 para o caso de uso *movement sensoring* na primeira solução.

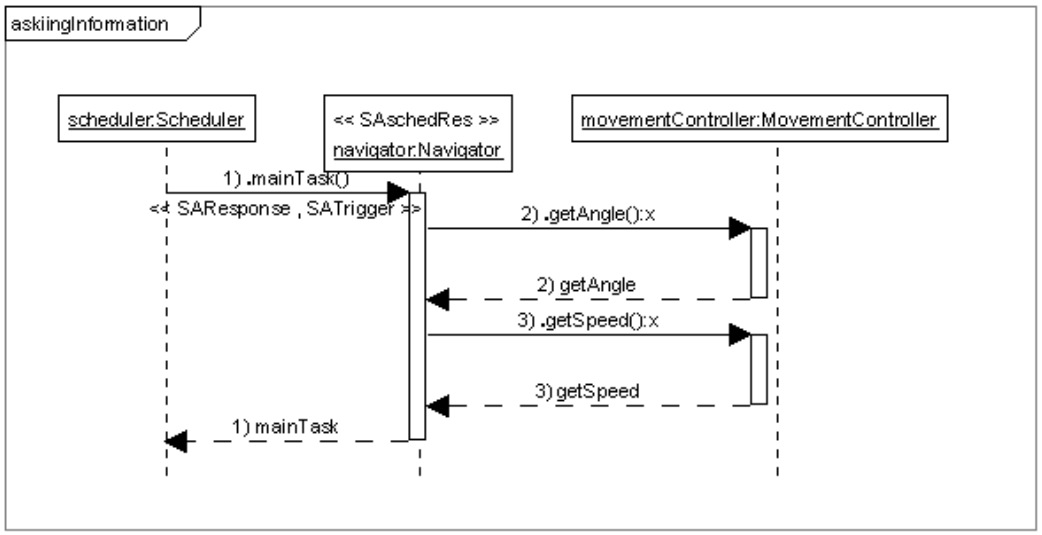

Figura 4.6: Diagrama de seqüência 2 para o caso de uso *movement sensoring* na primeira solução.

Os valores obtidos pelo SPEU para o caso de uso *movement control* são apresentados na Tabela 4.2. A tabela mostra também os valores exatos obtidos após a

síntese e a simulação, assim como os erros relativos entre os valores estimados e os exatos. Nesta tabela, os valores de memória são apresentados em *bytes* e o desempenho em ciclos. A energia consumida é avaliada em termos da atividade de chaveamento de portas, assim como em (TALARICO, 2005 e BECK, 2003). O desempenho, energia e a memória de dados foram avaliados em termos da execução de melhor caso (MC) e pior caso (PC).

Um erro de –12,37% no valor estimado de memória de dados foi encontrado, pois não é estimada a alocação dinâmica de memória para as operações na pilha de todas as instruções, nem são consideradas todas as variáveis locais alocadas na pilha de execução de um método. O MC e o PC nos valores estimados para memória de dados são os mesmos, visto que o valor máximo de memória alocada dinamicamente foi encontrado durante a execução do escalonador, que é idêntico para os dois casos. Além disso, há um erro de –12,61% no valor estimado para a memória de programa, pois partes do algoritmo de um método podem não ser capturadas pelos modelos UML. Isso ocorre quando um projetista não especifica o conteúdo do corpo de um método, pois um algoritmo complexo pode não ser facilmente expressado através de diagramas UML de seqüência. Isto também pode ocorrer para algumas instruções que implementam o método e atualmente não podem ser extraídas diretamente dos diagramas UML, como instruções de atribuição e aritméticas. Porém, é importante ressaltar que o comportamento mais relevante para a estimativa, como invocação de métodos, acessos à memória, execução condicional, execução iterativa e serviços reutilizados, são todos capturados. Como conseqüência dos erros no número de instruções executadas, há também erros na estimativa para valores de desempenho e energia.

Tabela 4.2: Resultados da estimativa para a primeira solução.

Sistema de Controle de Movimentos			
Propriedade	*SPEU*	*Exato*	*Erro (%)*
Memória de Programa	5.460	6.248	-12,61
Memória de dados - MC	510	582	-12,37
Memória de dados - PC	510	582	-12,37
Desempenho – MC	27.383	28.588	-04,21
Desempenho – PC	40.106	41.591	-03,57
Energia – MC	38.723.290	40.569.570	-04,55
Energia – PC	56.477.333	58.863.463	-04,05

4.3.3 Segunda solução

A segunda solução de modelagem foi projetada com o intuito de apresentar o maior desempenho e o menor consumo de energia e memória possíveis. Esta solução utiliza somente objetos estáticos e não utiliza tarefas escalonáveis para implementar o controle de movimentos. Ao invés destas, ela utiliza diretamente o sistema de controle de interrupções. A classe '*Navigator'* foi removida e suas responsabilidades foram incorporadas à classe principal '*MovementController'*. A classe '*SensorDriver'*, ao contrário da primeira solução, é responsável por executar a leitura dos dois sensores disponíveis nas rodas, reduzindo o número de objetos da mesma classe, e portanto

reduzindo o número de objetos. A Figura 4.7 ilustra o diagrama de classes para esta solução, onde não há relacionamento ou associação entre as classes, visto que todos os objetos são alocados estaticamente.

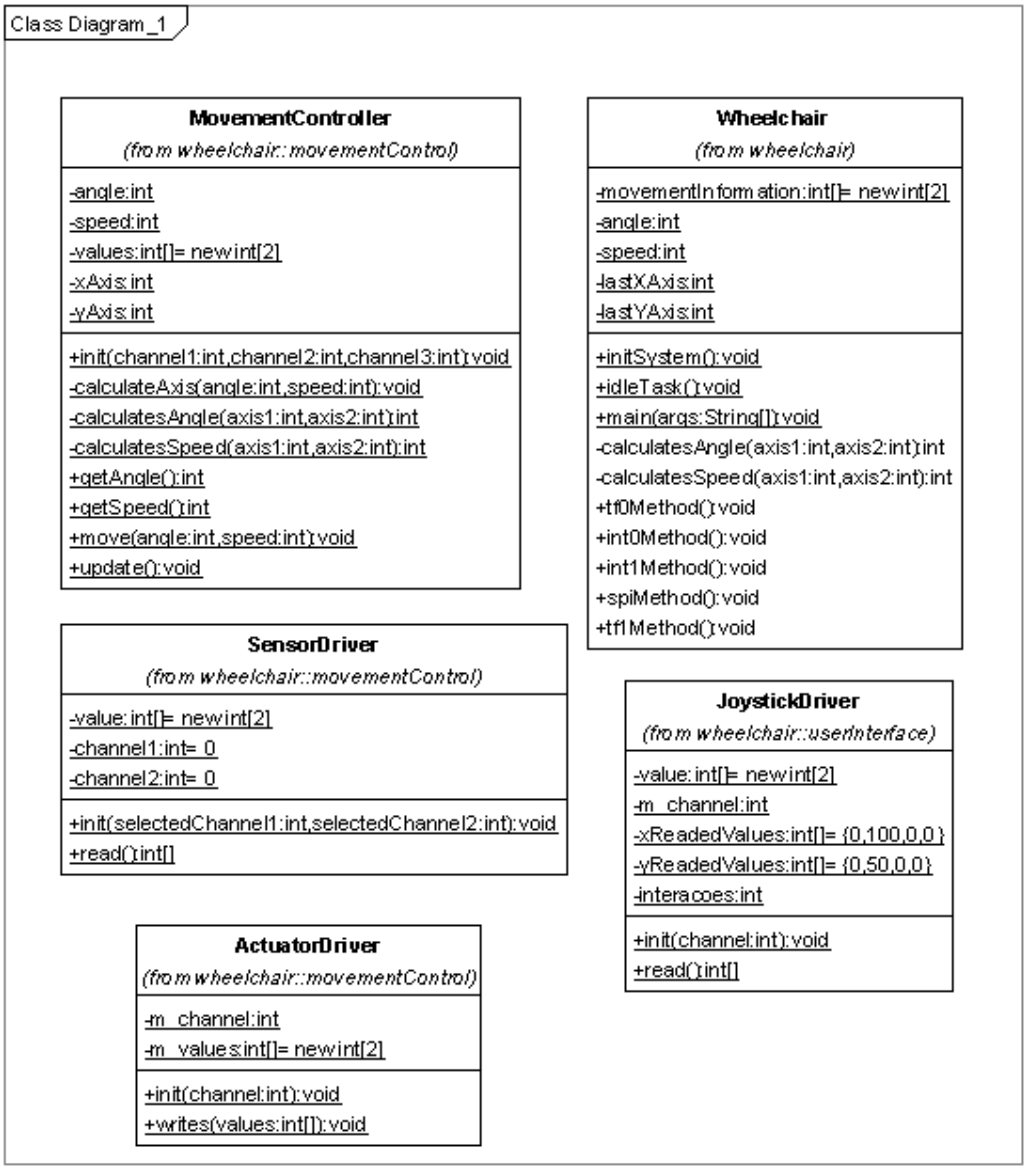

Figura 4.7: Diagrama de classes para o caso de uso *movement controller* na segunda solução.

O caso de uso *movement actuating* desta solução apresenta um encapsulamento mais fraco face à primeira solução, permitindo que os objetos interajam diretamente. Assim como na primeira solução, a segunda também converte os valores do controlador manual em valores de velocidade e ângulo. Os diagramas de seqüência para os casos de uso *movement actuating* e *movement sensoring* estão ilustrados na Figura 4.8 e 4.9.

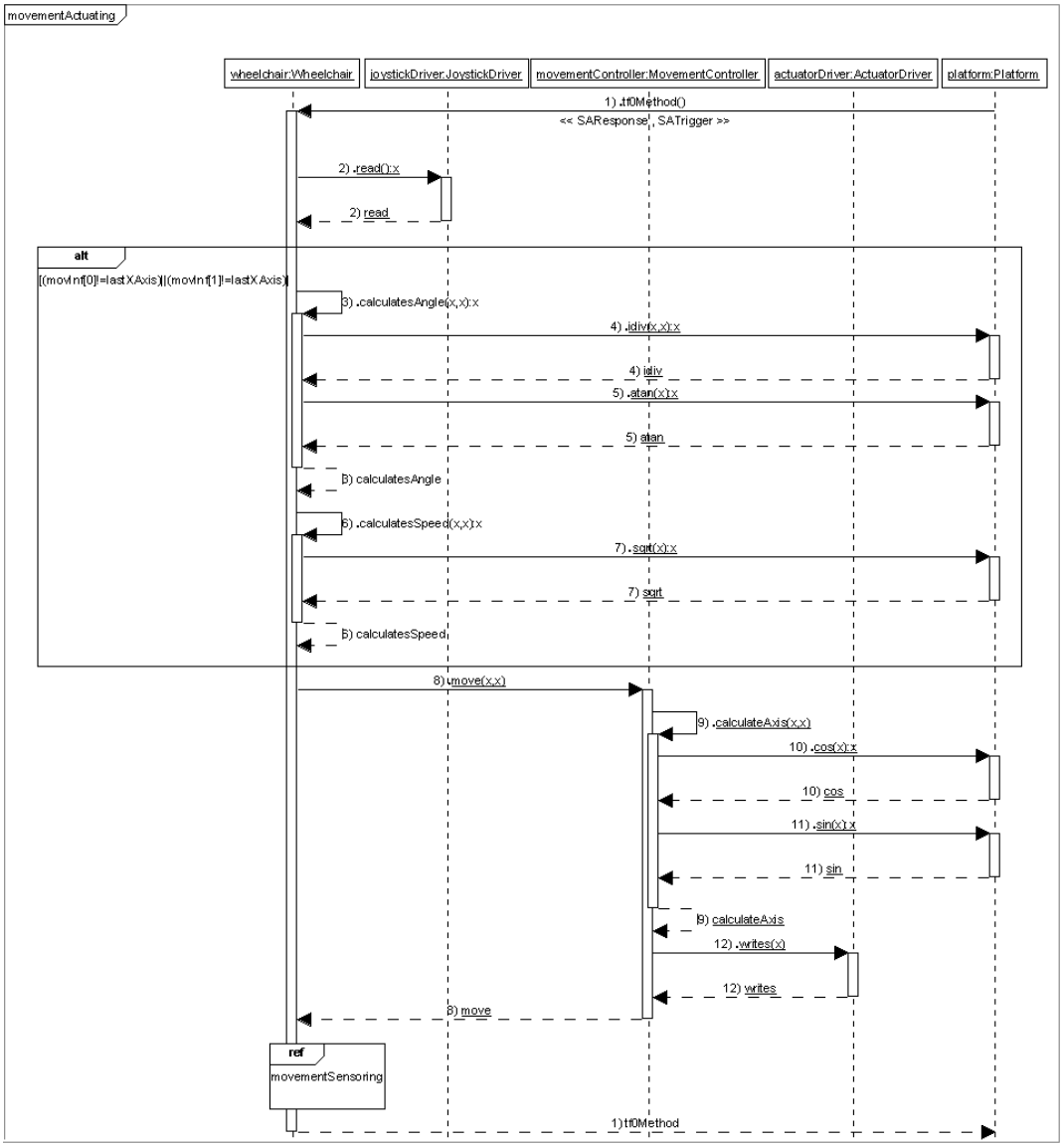

Figura 4.8: Diagrama de seqüência para o caso de uso *movement actuating* na segunda solução.

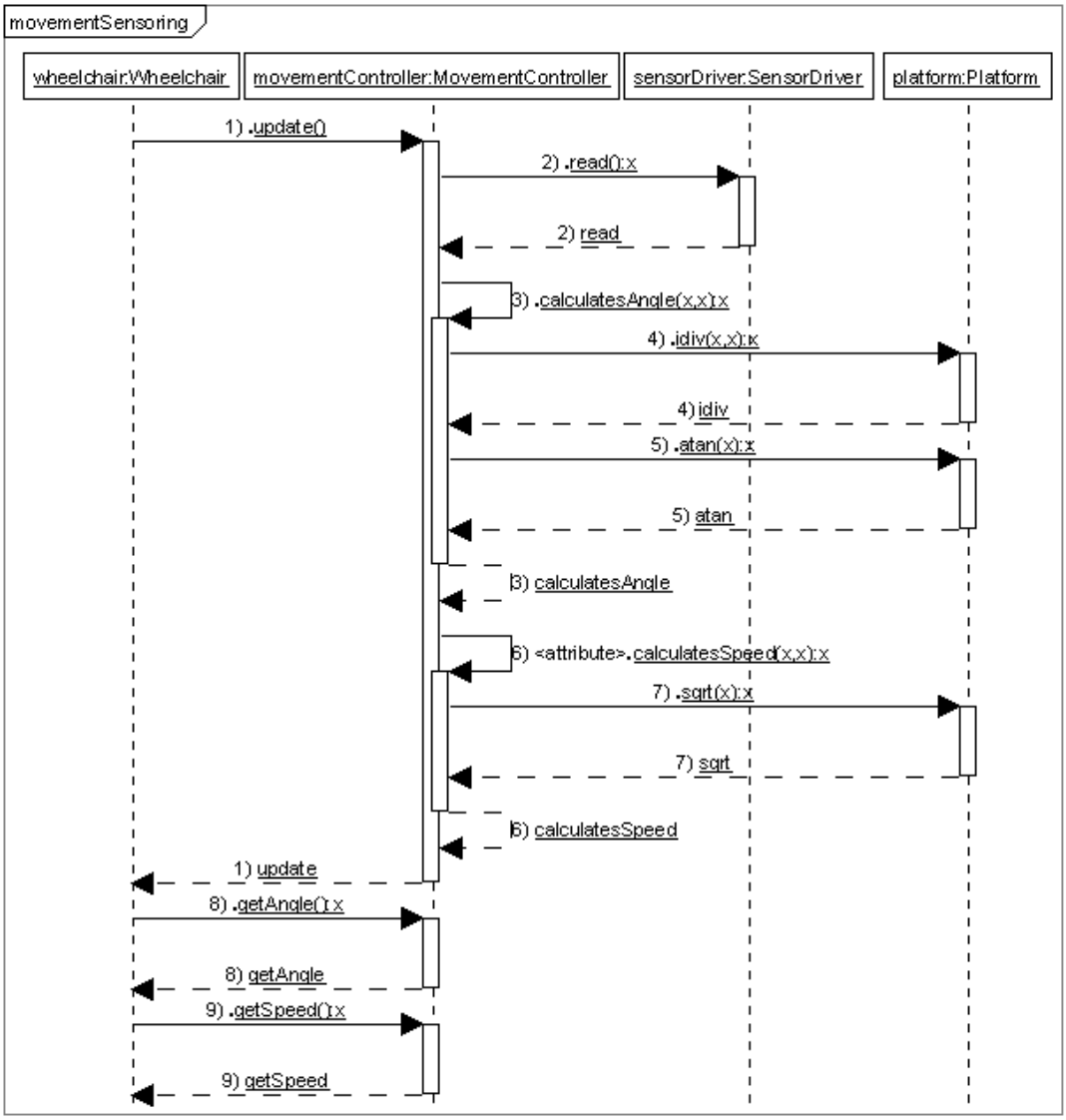

Figura 4.9: Diagrama de seqüência para o caso de uso *movement sensoring* na segunda solução.

A Tabela 4.3 apresenta os valores estimados (utilizando o SPEU) e os valores exatos (após a síntese e simulação) obtidos para a segunda solução. Já a Tabela 4.4 apresenta a diferença percentual entre a segunda e a primeira solução, para os valores estimados e exatos. A segunda solução mostra-se melhor em todas as sete métricas por uma grande diferença. Embora ela seja mais rápida, economize mais energia e apresente menores valores para o consumo de memória, ela é muito limitada em termos de flexibilidade, devido ao seu projeto estático e baixo grau de encapsulamento.

Na Tabela 4.3 podemos observar erros maiores para a memória de programa, melhor caso de desempenho e melhor caso de energia. O aumento na taxa de erro é devido ao baixo reuso de componentes por esta solução, principalmente durante a execução de melhor caso. Observando as taxas de erros alcançadas para a execução de pior caso, notamos taxas inferiores a 5%. Neste caso, durante a execução de pior caso a aplicação interage com os componentes da plataforma (componentes de funções matemáticas), os quais possuem seus custos conhecidos, reduzindo, assim, as taxas de erros.

Tabela 4.3: Resultados da estimativa para a segunda solução.

Controle de Movimentos – Segunda Solução			
Propriedade	*SPEU*	*Exato*	*Erro (%)*
Memória de Programa	1.647	2.063	-20,16
Memória de dados - MC	324	333	-02,70
Memória de dados - PC	324	333	-02,70
Desempenho - MC	1.253	1.898	-33,98
Desempenho - PC	13.706	14.423	-04,97
Energia - MC	1.485.002	2.714.132	-45,29
Energia - PC	18.874.973	20.302.894	-07,03

É importante notar na Tabela 4.4 que o SPEU foi capaz de capturar a influência das alterações presentes entre os dois modelos, o que permite estimar corretamente a diferença entre os dois modelos quando comparados e dizer qual é o ganho de um modelo em relação ao outro em uma determinada medida. Desta forma, podemos observar que o erro encontrado quando comparamos duas soluções, ou erro relativo, é o erro de interesse para a exploração do espaço de projeto. Podemos notar na Tabela 4.4 que o SPEU indicou corretamente que a segunda solução teve ganho em todas as sete medidas quando comparados os valores com a primeira solução. O mesmo comportamento pode ser observado nos outros experimentos, como será apresentado.

Tabela 4.4: Comparação dos resultados entre a primeira e a segunda solução.

Controle de Movimentos – Segunda x Primeira Solução		
Propriedade	*SPEU*	*Exato*
Memória de Programa	-69,84	-66,98
Memória de dados - MC	-36,47	-42,78
Memória de dados - PC	-36,47	-42,78
Desempenho - MC	-95,42	-93,36
Desempenho - PC	-65,82	-65,32
Energia - MC	-99,61	-93,31
Energia - PC	-66,57	-65,51

4.3.4 Terceira solução

Como a primeira solução apresenta custos elevados e a segunda solução é estática e inflexível, a terceira solução de modelagem foi projetada com o intuito de obter resultados intermediários em termos de desempenho e flexibilidade. Esta terceira solução utiliza o mesmo número de classes e objetos que a primeira solução. Porém, ela

utiliza somente uma tarefa para executar as funções de atuação e sensoriamento. O diagrama de classes para a terceira solução está ilustrado na Figura 4.10.

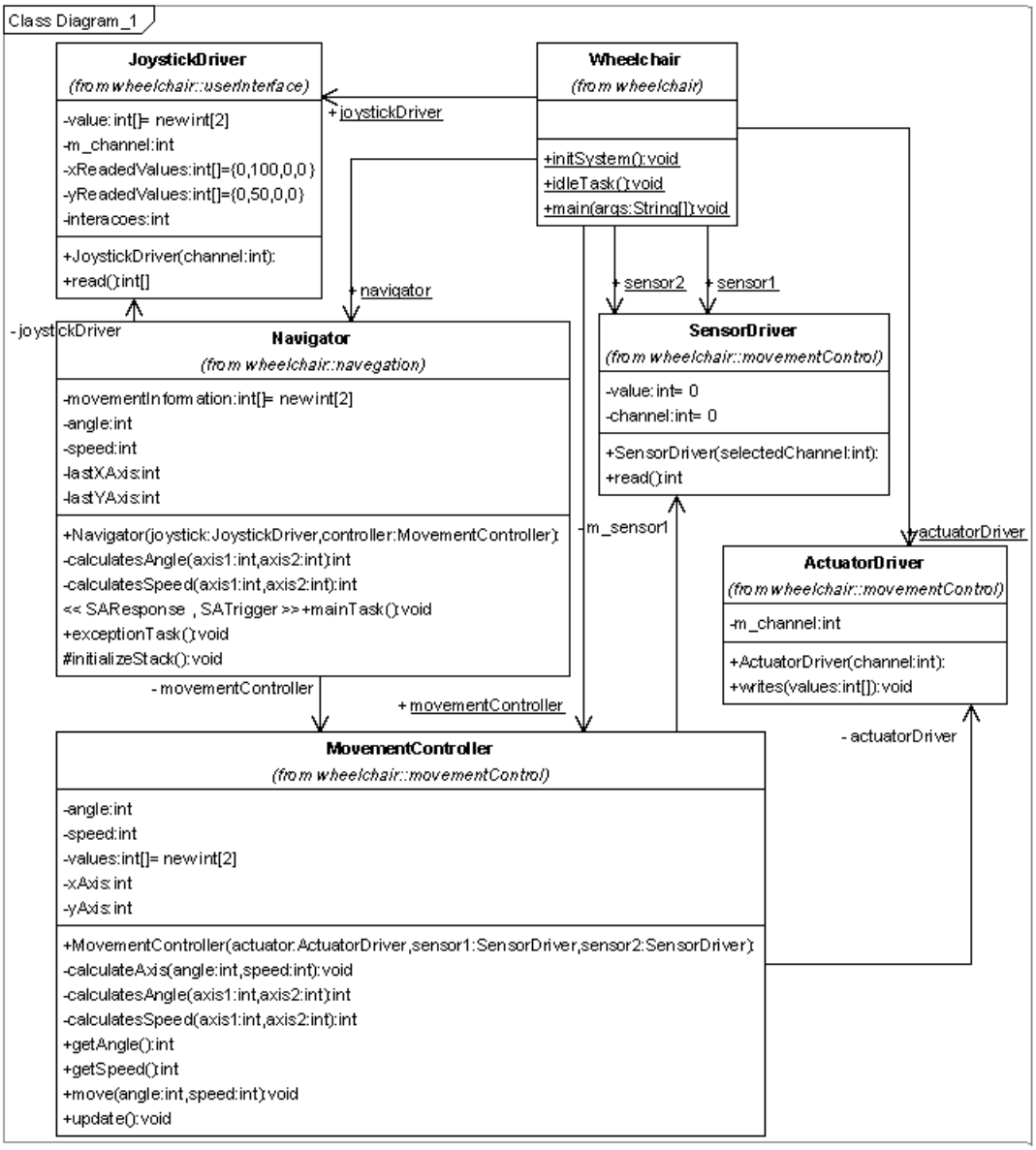

Figura 4.10: Diagrama de classes do caso de uso *movement controller* para a terceira solução.

Assim como nas outras duas soluções, foi mantida a conversão dos valores do controlador manual para velocidade e ângulo Os diagramas de seqüência para os casos de uso *movement actuating* e *movement sensoring* estão ilustrados na Figura 4.11 e 4.12.

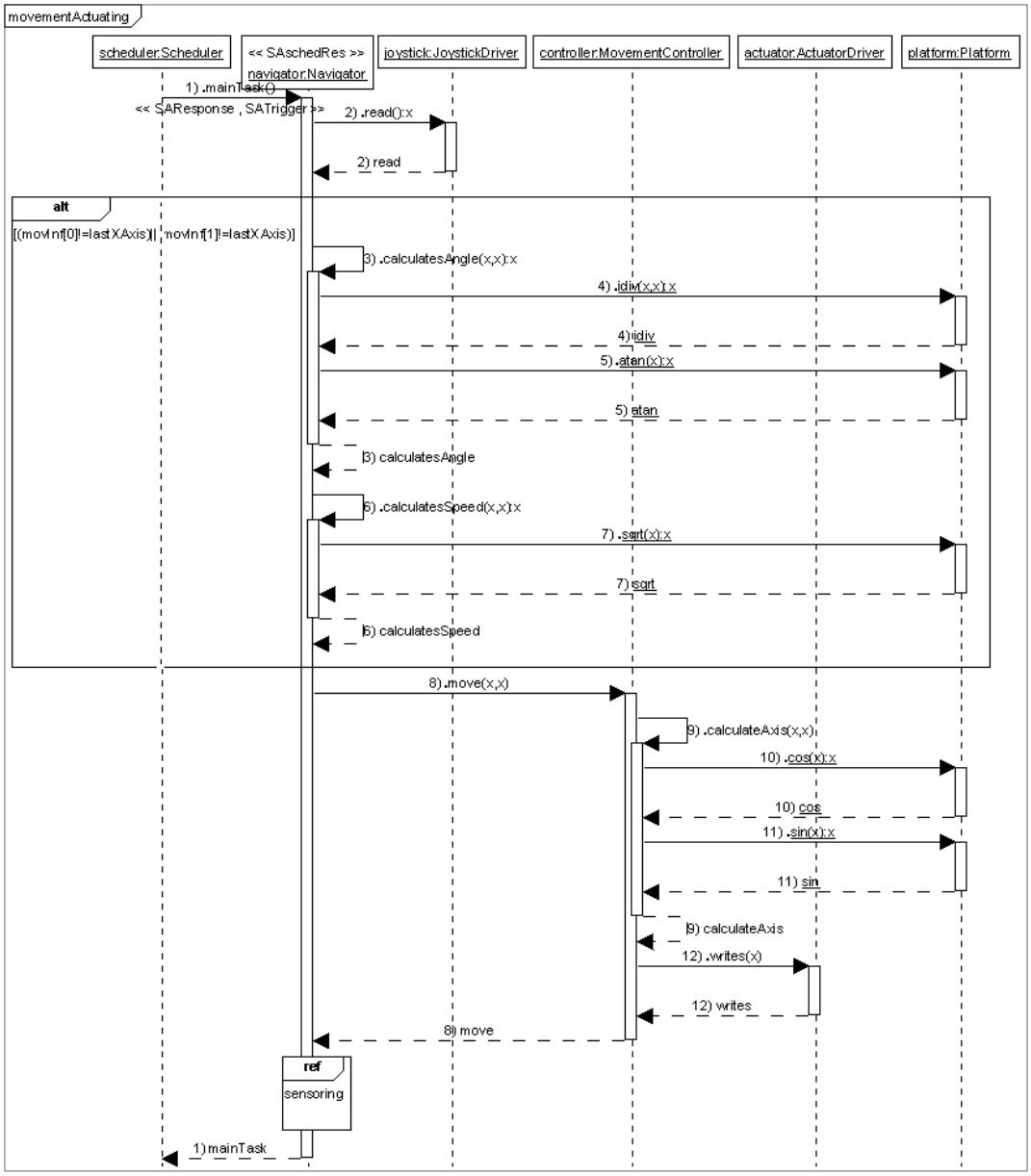

Figura 4.11: Diagrama de seqüência do caso de uso *movement actuating* para a terceira solução.

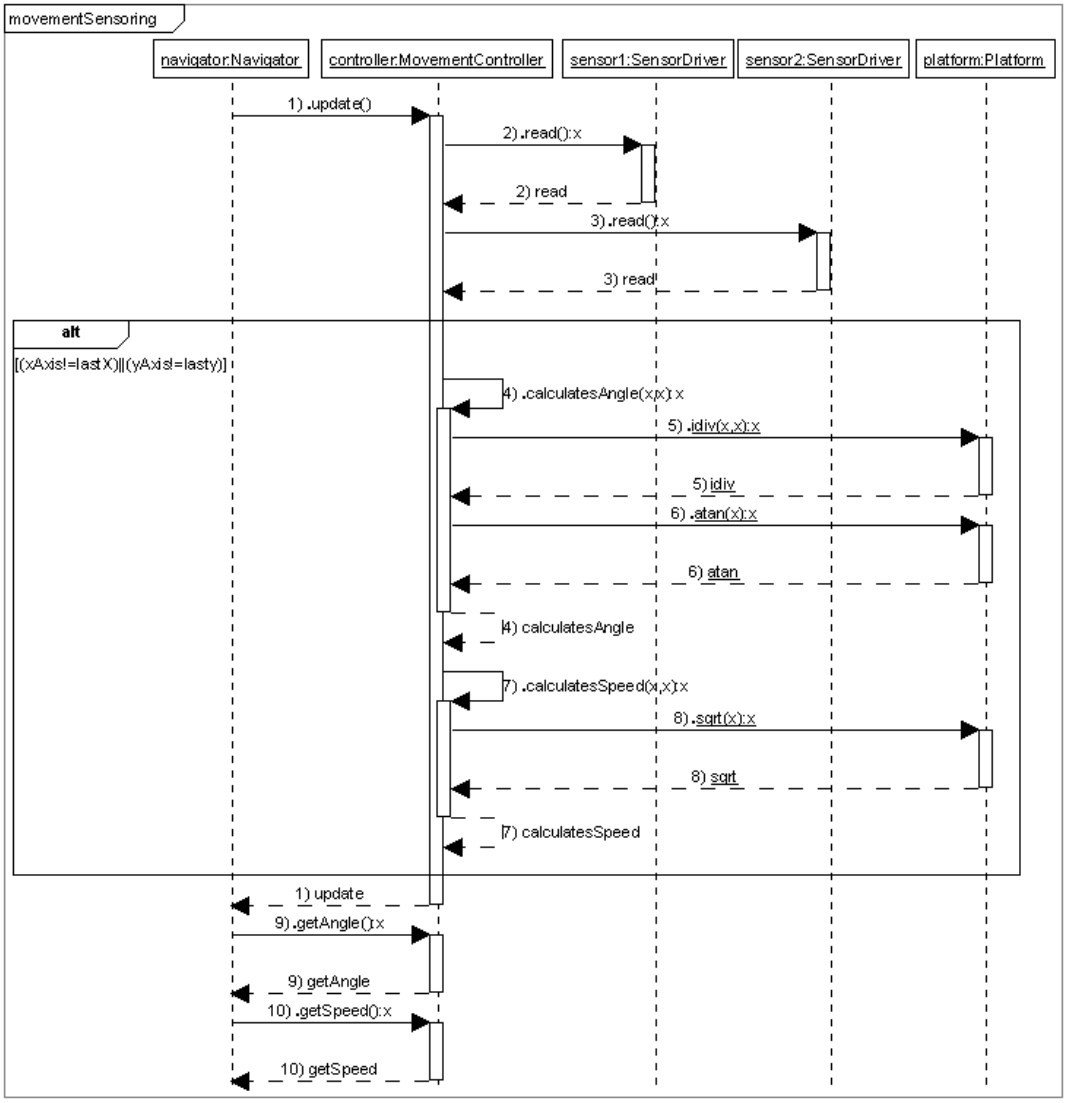

Figura 4.12: Diagrama de seqüência do caso de uso *movement sensoring* para a terceira solução.

Valendo-se da terceira solução, obteve-se resultados de desempenho melhores que na primeira solução, devido ao baixo número de tarefas e objetos, reduzindo-se assim os custos adicionais do escalonamento e da interação entre objetos. Mas a segunda solução continua muito mais rápida do que a terceira, devido à alocação dinâmica e ao custo de escalonamento. Porém, na terceira solução, novos componentes podem ser inseridos com mais facilidade do que na segunda solução, pois a utilização de alocação dinâmica e escalonador tornam mais fáceis a expansão de um sistema ou a alteração da configuração dos componentes. A Tabela 4.5 apresenta os valores estimados e exatos para a terceira solução e a Tabela 4.6 compara a terceira solução com a primeira e segunda solução.

Tabela 4.5: Resultados da estimativa para a terceira solução.

Controle de Movimentos – Terceira Solução			
Propriedade	*SPEU*	*Exato*	*Erro (%)*
Memória de Programa	4.545	5.208	-12,73
Memória de dados - MC	410	431	-04,87
Memória de dados - PC	410	431	-04,87
Desempenho – MC	7.803	9.104	-14,29
Desempenho – PC	20.230	21.673	-06,66
Energia – MC	10.839.147	12.916.022	-16,08
Energia – PC	28.276.462	30.535.449	-07,40

Tabela 4.6: Comparação dos resultados para a terceira solução.

	Terceira e Primeira Solução		*Terceira e Segunda Solução*	
Propriedade	*SPEU (%)*	*Exato (%)*	*SPEU (%)*	*Exato (%)*
Memória de Programa	-16,76	-16,65	175,96	152,45
Memória de dados - MC	-22,16	-25,95	22,53	29,43
Memória de dados - PC	-22,16	-25,95	22,53	29,43
Desempenho – MC	-71,50	-68,15	522,75	379,66
Desempenho – PC	-49,56	-47,89	47,60	50,27
Energia – MC	-72,01	-68,16	629,91	375,88
Energia – PC	-49,93	-48,12	49,81	50,40

4.3.5 Quarta solução

Nesse contexto, a quarta solução de modelagem foi projetada utilizando uma arquitetura similar à da terceira solução. Porém, diferente das três soluções anteriores, os valores do controlador manual não são convertidos em velocidade e ângulo no caso de uso *movement actuating*. A quarta solução apresenta um encapsulamento similar ao da terceira, com somente uma tarefa e a mesma classe estendida '*SensorDriver*', como na segunda solução. A Figura 4.13 ilustra o diagrama de classes para a quarta solução.

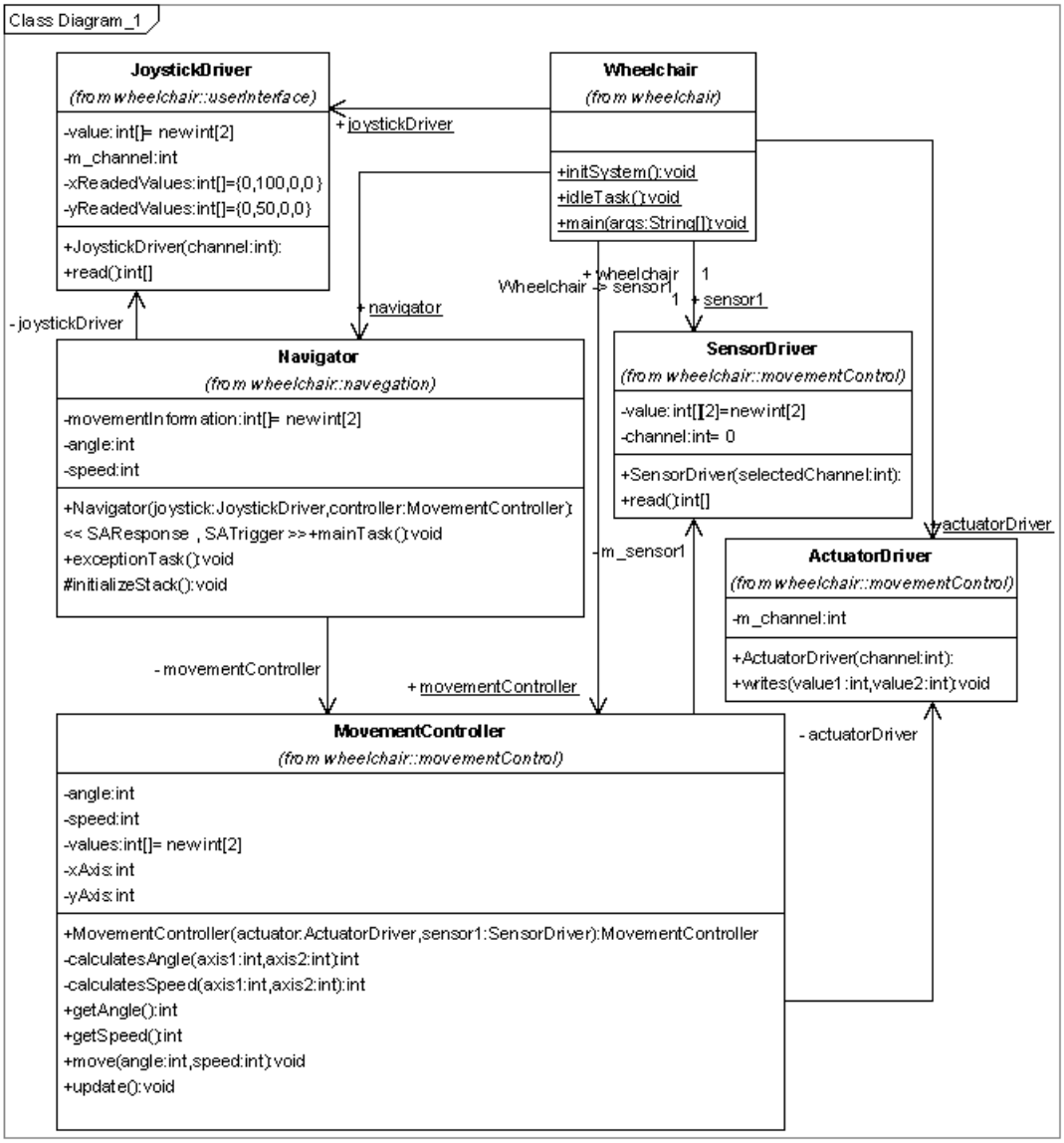

Figura 4.13: Diagrama de classes do caso de uso *movement controller* para a quarta solução.

Além disso, no caso de uso '*movement sensoring'*, um algoritmo mais eficiente foi utilizado, evitando que a velocidade e o ângulo sejam computados desnecessariamente. Com essas mudanças, algumas das flexibilidades apresentadas na primeira e terceira soluções foram perdidas, com o objetivo de alcançar um desempenho similar ao da segunda solução. As Figuras 4.14 e 4.15 ilustram os diagramas de seqüências para a quarta solução.

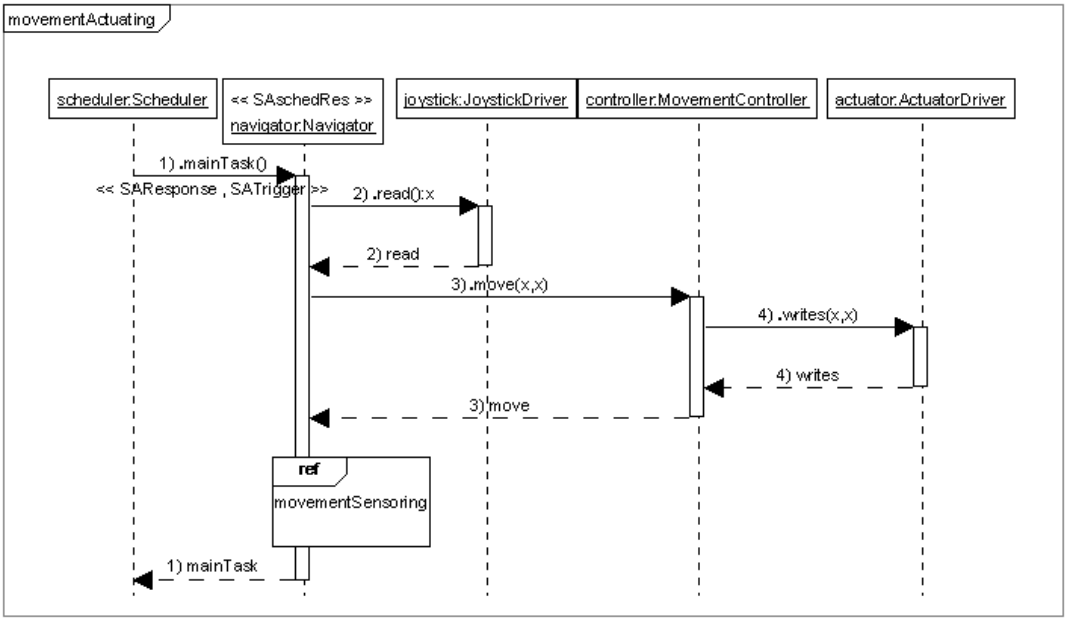

Figura 4.14: Diagrama de seqüência do caso de uso *movement actuating* para a quarta solução.

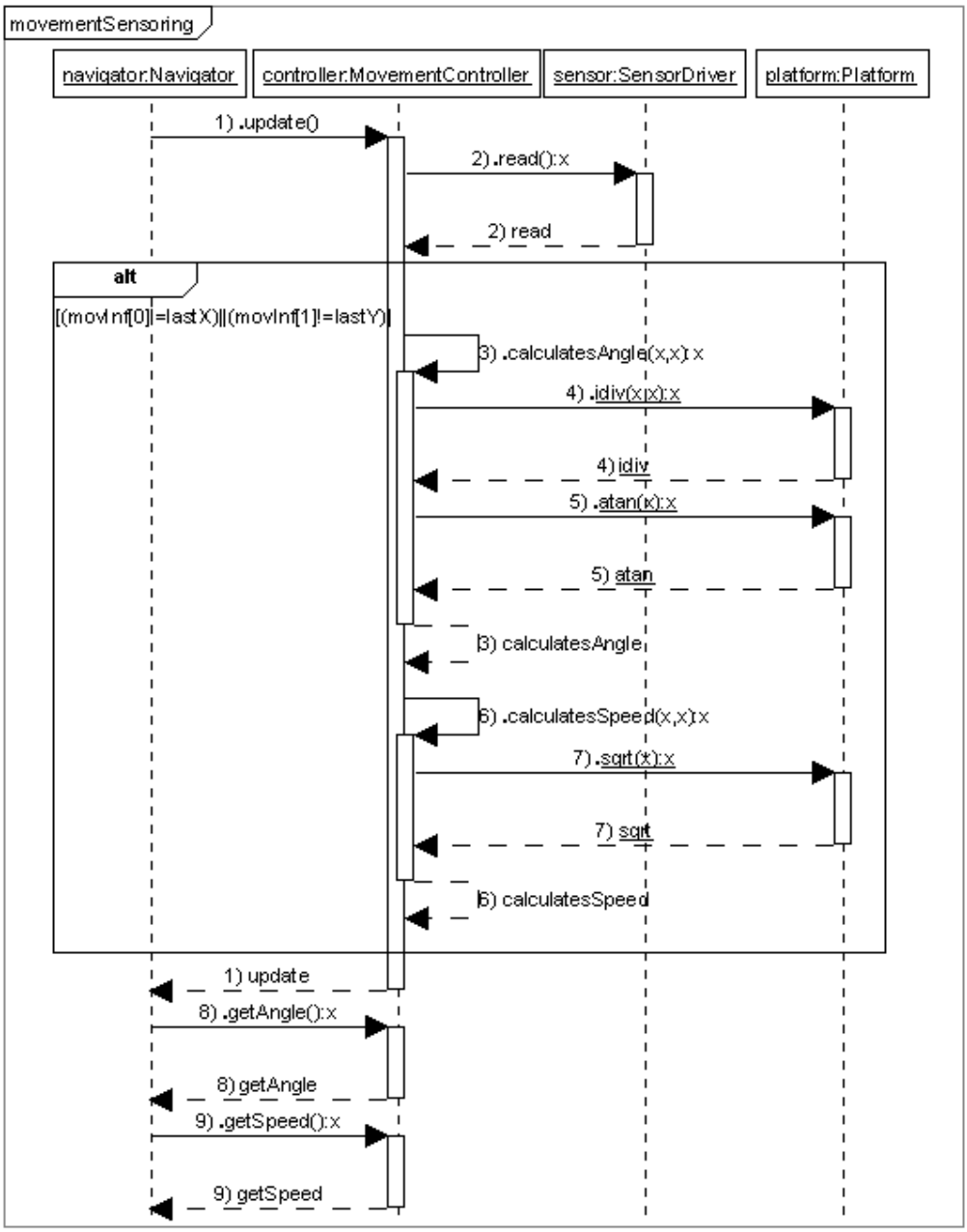

Figura 4.15: Diagrama de seqüência do caso de uso *movement controller* para a quarta solução.

A Tabela 4.7 apresenta os valores estimados e exatos para a terceira solução e a Tabela 4.8 compara a quarta solução com a segunda e terceira soluções.

Tabela 4.7: Resultados da estimativa para a quarta solução.

Controle de Movimentos – Quarta Solução			
Propriedade	*SPEU*	*Exato*	*Erro (%)*
Memória de Programa	4.387	5.094	-13,88
Memória de dados – MC	402	421	-04,51
Memória de dados – PC	402	421	-04,51
Desempenho – MC	6.878	7.776	-11,55
Desempenho – PC	13.450	14.510	-07,31
Energia – MC	9.684.235	11.026.748	-12,18
Energia – PC	18.848.311	20.511.557	-08,11

Tabela 4.8: Comparação dos resultados para a quarta solução.

	Quarta e Segunda Solução		*Quarta e Terceira Solução*	
Propriedade	*SPEU (%)*	*Exato (%)*	*SPEU (%)*	*Exato (%)*
Memória de Programa	166,36	146,92	-3,48	-2,19
Memória de dados - MC	24,07	26,43	-1,95	-2,32
Memória de dados - PC	24,07	26,43	-1,95	-2,32
Desempenho - MC	448,92	309,69	-11,85	-14,59
Desempenho - PC	-1,87	0,60	-33,51	-33,05
Energia - MC	552,14	306,27	-10,66	-14,63
Energia - PC	-0,14	1,03	-33,34	-32,83

A quarta solução apresenta melhorias em torno de 11% no melhor caso de execução e 33% no pior caso de execução, quando comparada à terceira solução. Porém, a segunda solução ainda apresenta os melhores resultados no desempenho para as execuções de melhor e pior caso, pois não apresenta custos com escalonamento.

4.4 Análise da estimativa

Nas seções anteriores foram apresentados os valores estimados para desempenho, energia e memória para quatro soluções alternativas para a mesma aplicação. Os valores estimados foram comparados a valores exatos, extraídos a partir de um simulador com precisão de ciclo e de uma ferramenta de síntese, e os erros alcançados em cada estimativa foram apresentados. Experimentos realizados em (OLIVEIRA, 2005) apresentam os resultados de estimativa para dois modelos alternativos para o Sokoban, um jogo popular de lógica. Neste outro experimento, não houve reuso de componentes da plataforma. A unidade de processamento utilizada, assim como nos experimentos das

seções anteriores, também foi o microcontrolador femtojava Multiciclo. A Tabela 4.10 compara os erros obtidos para os experimentos com dois modelos alternativos para o Sokoban e os erros obtidos para os quatro modelos alternativos para o controle de movimentos da cadeira de rodas.

Tabela 4.9: Erros apresentados nas estimativas realizadas para soluções alternativas para o Sokoban e a cadeira de rodas.

Propriedade	*Sokoban*		*Cadeira de rodas*			
	Primeira	*Segunda*	*Primeira*	*Segunda*	*Terceira*	*Quarta*
Memória de Programa	-61,86	-86,5	-12,61	-20,16	-12,73	-13,88
Memória de dados-MC	-0,90	-0,46	-12,37	-02,70	-04,87	-04,51
Memória de dados-PC	*	*	-12,37	-02,70	-04,87	-04,51
Desempenho-MC	-71,99	-80,10	-04,21	-33,98	-14,29	-11,55
Desempenho-PC	-81,20	-85,32	-03,57	-04,97	-06,66	-07,31
Energia-MC	-68,95	-77,13	-04,55	-45,29	-16,08	-12,18
Energia-PC	-79,80	-85,66	-04,05	-07,03	-07,40	-08,11

* valores não disponíveis.

Diversos fatores influenciam a estimativa e contribuem para elevação do erro:

1- *Falta de detalhes da especificação do sistema*: em modelos descritos em alto nível de abstração, informações detalhadas sobre o comportamento da aplicação não podem ser extraídas. O nível de abstração determina o volume de detalhes disponíveis, mas também o tempo necessário para descrever e analisar os modelos. Quando aumentamos a abstração, devemos estar dispostos a trabalhar com erros mais flexíveis;

2- *Imprecisão na caracterização dos componentes da plataforma*: Os componentes da plataforma podem ser caracterizados de formas diferentes. Manter um repositório com informações precisas exige que um esforço seja despendido em análises complexas para extrair informações dinâmicas e/ou relacionadas a detalhes em baixo nível de abstração. Por este motivo, são realizadas aproximações, como cálculo do pior e melhor caso, ou até mesmo do caso médio. Pode-se também realizar análises de níveis mais altos de abstração, ou utilizar a experiência dos Desenvolvedores;

3- *Imprecisão ou abstração no modelo das unidades de processamento*: Um processador que contenha memória *cache*, *pipeline* ou preditores de desvio exige modelos mais complexos de representação. Na abordagem proposta, o impacto da organização da unidade processamento é parcialmente capturado quando caracterizamos suas instruções e os componentes de software para uma unidade específica. Porém, há variações dependentes dos dados de entrada que não são capturados e impõem um erro tanto na medição das instruções como nos componentes de software;

4- *Uso de apenas informações estáticas*: Quando somente informações estáticas são utilizadas, erros relativamente altos podem ser esperados para a estimava, visto que as informações dinâmicas influenciam no caminho percorrido pela

aplicação, nas unidades funcionais da organização e no chaveamento de portas de um processador. Porém, assim como em (LI, 1997) os resultados obtidos podem ser efetivamente utilizados pelos desenvolvedores.

Outro fator que impacta no resultado da estimativa é o volume de componentes reutilizados. Quando reutilizamos componentes disponibilizados pela plataforma, reduzimos as incertezas a respeito da aplicação. Os erros apresentados pela Tabela 4.10 demonstram este fato. O Sokoban não reutiliza nenhum componente de software da plataforma, ao contrário das soluções alternativas para o controle de movimentos da cadeira de rodas, que reutilizam muitos componentes. Como conseqüência, os erros apresentados pelo estimador para os modelos do Sokoban chegam a ser 20 vezes maior do que os erros apresentados pelo estimador para os modelos do controle de movimentos da cadeira de rodas. A Figura 4.16 ilustra um diagrama de seqüência utilizado no caso de uso *movementActuating* e destaca os componentes que foram reutilizados. Neste diagrama, foram reutilizadas duas funções da biblioteca de funções matemáticas (métodos *cos* e *sin*), destacados com duas elipses na figura 4.16 e parte do framework de tempo real, destacado com dois retângulos, para especificação da tarefa *actuatorDriver* e consequentemente o escalonador e os temporizadores necessários para a tarefa.

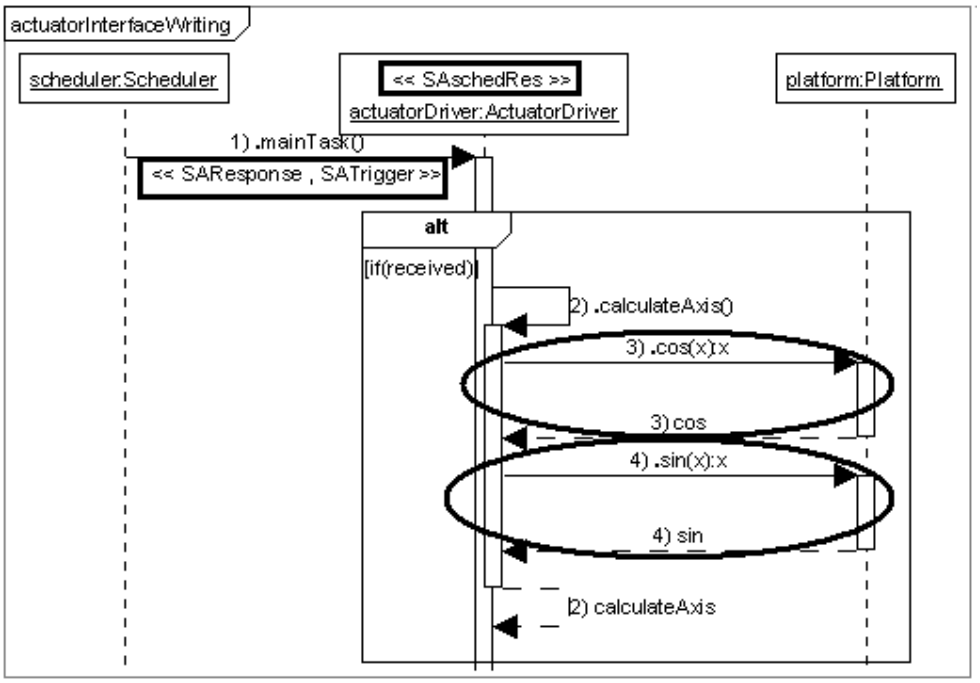

Figura 4.16: Fragmento do diagrama de seqüência 2 do caso de uso *movement actuating* para a primeira solução, ilustrando alguns dos componentes reutilizados.

Assim como nos resultados obtidos para o Sokoban, a segunda solução para o controle da cadeira de rodas também apresenta erros maiores em suas estimativas, quando comparado aos erros das outras soluções alternativas. Isso porque a solução não utiliza o *framework* para especificação de aplicações de tempo real (escalonador, temporizadores e API). Esse fato se destaca quando comparamos os erros obtidos para o pior e o melhor caso de execução. Na execução de melhor caso os erros são maiores, pois além do *framework*, durante a execução de melhor caso também os serviços da biblioteca de matemática não são requisitados.

4.5 Conclusões

Os resultados obtidos no estudo de caso demonstram que em nível alto de abstração já é possível estimar as propriedades do sistema com taxas de erros que podem chegar a 5%. As estimativas apresentam valores altamente relacionados aos valores obtidos na implementação final, demonstrando que, embora existam erros nas estimativas, estes não impedem que o Desenvolvedor do Sistema identifique a solução de modelagem mais adequada para um dado conjunto de requisitos e restrições.

Observando a comparação de resultados apresentada na Seção 4.4, podemos destacar que maior precisão pode ser alcançada quando mais componentes pré-caracterizados são reutilizados na aplicação, visto que os custos de suas funções já são conhecidos. Porém, é importante ressaltar que o erro relativo, encontrado quando comparamos duas soluções, é reduzido e identifica corretamente a variação dos valores entre os modelos. Desta forma, o erro relativo é mais importante para a exploração do espaço de projeto, pois identifica as diferenças entre os modelos. Já o erro absoluto, quando comparamos os resultados da estimativa do SPEU para uma solução com os resultados obtidos através da implementação e simulação da mesma solução, são mais altos que os erros relativos, mas de pouca importância para a exploração do espaço de projeto. Os erros absolutos apresentados pelo SPEU devem ser reduzidos para auxiliar no dimensionamento da plataforma, tarefa que exige mais precisão.

A ferramenta SPEU provê informações quantitativas sobre as métricas físicas que podem ser utilizadas por um projetista durante o processo de desenvolvimento, já nos estágio iniciais e antes da geração de código e da síntese do hardware. Observando as estimativas providas pela SPEU e os requisitos do sistema, o projetista pode explorar o espaço de projeto, utilizando as estimativas como guia para selecionar a solução de modelagem mais adequada. Visto que os valores estimados não dependem de simulação, eles podem ser utilizados como suporte a uma exploração do espaço de projeto realizada em níveis altos de abstração, rapidamente e com uma precisão aceitável.

As soluções de modelagem alternativas demonstram que há um grande espaço de projeto a ser explorado quando um projetista considera diferentes alternativas de encapsulamento e de distribuição de responsabilidades entre objetos, como o número de tarefas e a forma com que os objetos irão interagir. Portanto, a abordagem proposta, além de ser rápida e relativamente precisa, disponibiliza mais oportunidades de otimização, permitindo analisar o impacto das alternativas de modelagem da aplicação na implementação final do sistema.

Obviamente, esta mesma ferramenta de estimativa pode ser utilizada também na exploração de diferentes arquiteturas para suporte à aplicação, assim como de diferentes mapeamentos entre a aplicação e a arquitetura. Estas explorações são aquelas já tradicionais da área de co-projeto de hardware e software (GRIES, 2004).

5 CONCLUSÕES

Este trabalho apresentou uma abordagem para exploração do espaço de projeto de sistemas embarcados baseados em plataforma através de estimativas extraídas de modelos UML. A abordagem proposta especificou cinco aspectos para permitir esta abordagem de exploração do espaço de projeto:

- Um conjunto de regras de modelagem em UML;
- Um modelo que represente uma plataforma, formando um repositório que contenha informações sobre os componentes disponíveis para reuso e que será consultado durante o processo de estimativa;
- Mecanismos para mapeamento entre o modelo de alto nível da aplicação em UML e a plataforma utilizada, e as características de baixo nível necessárias para que os custos de um sistema possam ser avaliados;
- Um processo de estimativas que permita a avaliação de modelos UML;
- Um processo de exploração do espaço de projeto que, através do estimador, gere um espaço de soluções candidatas e selecione a melhor solução para o sistema.

O conjunto de regras para geração de modelos UML permitiu a extração das informações relevantes à avaliação quantitativa dos modelos. O conjunto de regras apresentado é simples, com pouca interferência no estilo geral de modelagem que um Desenvolvedor de Sistema poderia utilizar. Essa característica facilita a integração da fase de desenvolvimento com a fase de avaliação, não exige o aprendizado de novas e complexas tecnologias e facilita a aceitação da abordagem pelos possíveis usuários.

Através de um modelo de plataforma, a abordagem proposta pode representar as informações necessárias para realizar as estimativas, reutilizando o conhecimento disponível sobre componentes pré-caracterizados que são utilizados em um ambiente de desenvolvimento baseado em plataformas. Esta abordagem aumenta a precisão do processo de estimativa, pois reduz a incerteza acerca dos componentes do sistema. Este modelo é fortemente baseado nos modelos presentes no perfil UML-SPT. Além disso, baseia-se também em outras especificações padrão, como UML, MOF e XMI/XML, facilitando a extensão e integração do modelo a modelos mais complexos, que podem ser utilizados tanto para estimativa quanto para síntese.

Os mecanismos de mapeamento propostos se apresentam como uma ponte entre os modelos abstratos gerados em UML e as informações de baixo nível de abstração, que a abordagem pretende estimar. Utilizando diagramas de implantação, o Desenvolvedor do Sistema pode alterar rapidamente a alocação dos componentes de *software* entre as unidades de processamento disponíveis na plataforma permitindo, assim, que o *software*

em desenvolvimento seja avaliado em diferentes processadores. As informações sobre os componentes pré-caracterizados presentes no modelo da plataforma também são consideradas como um mecanismo de mapeamento, pois estas apresentam implicitamente o mapeamento utilizado. Por fim, um conjunto de instruções simbólicas representa as características relevantes do comportamento do sistema. Uma ferramenta de mapeamento, em conjunto com o repositório da plataforma, auxilia na geração de um conjunto de instruções mapeadas para as instruções reais das unidades de processamento disponíveis na plataforma, permitindo que, durante o processo de exploração, a aplicação seja avaliada sobre diferentes unidades de processamento. Os três mecanismos apresentados permitem a estimativa rápida e auxiliam no processo de exploração do espaço de projeto.

Através da especificação dos aspectos comentados anteriormente, foi possível propor um processo que permite a avaliação quantitativa de modelos gerados em UML. Esse processo extrai as informações relevantes acerca do sistema, montando uma assinatura que contém informações independentes de plataforma. Esse fato permite a reutilização da assinatura com diferentes mapeamentos de plataforma, sem a necessidade de avaliar novamente a aplicação. Após a geração da assinatura, um processo de mapeamento da assinatura gerada permite que a ferramenta de estimativa SPEU seja facilmente integrada a uma ferramenta de exploração automática. De posse da assinatura mapeada, um processo analítico, utilizando métodos de programação linear inteira, permite identificar as execuções de melhor e pior caso obtendo, então, valores estimados de desempenho, energia, memória de dados e de programa para ambos os casos de execução. Os resultados obtidos podem ser anotados novamente em modelos UML, permitindo que o Desenvolvedor do Sistema visualize os resultados e que a solução proposta possa ser avaliada por uma ferramenta de exploração automática.

A partir da abordagem para avaliação quantitativa de modelos UML, novas oportunidades de exploração do espaço de projeto podem ser aproveitadas, permitindo ao Desenvolvedor do Sistema avaliar o impacto de suas decisões de modelagem sobre as propriedades físicas do sistema. Em um primeiro momento, esta abordagem permite o Desenvolvedor explorar manualmente diferentes soluções de modelagem, alterando a arquitetura de *software* no que diz respeito à distribuição de responsabilidades, à interação entre objetos, à especificação de tarefas e ao número de objetos. Outras oportunidades de exploração já utilizadas nos processos de desenvolvimento também podem ser exploradas, como a seleção de processadores e particionamento de tarefas entre processadores. Devido à independência do estimador, aos modelos da aplicação e ao uso de tecnologias padrão, em um segundo momento a ferramenta de estimativa pode ser integrada a uma ferramenta de exploração automática do espaço de projeto.

Experimentos com aplicações embarcadas demonstram que, em nível alto de abstração, já é possível estimar as propriedades do sistema, com taxas de erros que podem ser inferiores a 5%. Os valores estimados são altamente relacionados aos valores obtidos por simulação com precisão de ciclos após a implementação final do sistema. Como os valores estimados não dependem de simulação, eles podem ser utilizados como suporte para explorar o espaço de projeto rapidamente em níveis altos de abstração, com uma precisão aceitável.

Os experimentos também demonstram que, observando as estimativas providas pela ferramenta SPEU e os requisitos do sistema, o projetista pode explorar o espaço de projeto utilizando as estimativas como guia para selecionar a solução de modelagem

mais adequada. Resultados obtidos após as estimativas mostram um grande espaço de projeto a ser explorado, quando um projetista considera soluções alternativas nos modelos UML. Portanto, a abordagem de estimativa suporta uma exploração rápida e relativamente precisa do impacto das diferentes alternativas na implementação final do sistema.

5.1 Trabalhos Futuros

As regras apresentadas para modelagem são simples e acrescentam pouco formalismo a UML. Como o processo de geração de assinatura não foi automatizado, pode ser necessário estender o conjunto de regras para permitir que o processo seja automatizado. Do mesmo modo, novas regras podem permitir extração de informações que ainda não são obtidas dos modelos, como a identificação de comunicação entre tarefas e a avaliação das condições de guarda. Além disso, outros diagramas UML podem ser incorporados à abordagem, como os diagramas de estados e de atividades, permitindo capturar comportamentos ainda mais complexos.

O modelo da plataforma proposto baseia-se nos modelos estáticos disponíveis no UML-SPT. Esse modelo pode ser estendido utilizando os conceitos de modelagem de execução dinâmica, o que pode aumentar o número de serviços que podem ser representados pelo modelo proposto. Outras extensões podem ser propostas para capturar informações sobre comunicação em uma estrutura de comunicação, como barramentos ou uma NoC (*Network-on-Chip*).

Naturalmente, os passos do processo de estimativa que ainda requerem intervenção do Desenvolvedor do Sistema poderiam ser automatizados. Esses passos são: geração de assinatura, estimativa secundaria e anotação dos resultados nos modelos UML. O processo de estimativa também poderia ser estendido, para que, após as estimativas propostas neste trabalho, análises mais detalhadas, como na proposta de (WILLIAMS, 2002), possam contribuir para o processo de desenvolvimento. O estimador SPEU também poderia ser integrado a uma ferramenta de exploração automática que, através de alguma heurística, avalie diferentes opções de alocação de tarefas e de mapeamento.

Adicionalmente, novos experimentos podem ser realizados. Em especial, experimentar a abordagem proposta com novas aplicações, que apresentem diferentes características, como *data-flow* ou *control-flow*, que permitiria avaliar a precisão, a robustez e a generalidade da abordagem. Nesses experimentos, pode ser identificada a necessidade de alterações nas regras de modelagem para expressar de forma adequada o comportamento de aplicações do tipo *data-flow*. A adoção de um perfil UML específico para aplicações desse tipo e/ou incorporação de outros diagramas UML também podem ser necessárias. Simultaneamente, a representação interna poderia ser alterada, objetivando representar as possíveis alterações nos modelos, de forma que esse seja capaz de capturar as novas características. Outros experimentos podem procurar por novas oportunidades de exploração do espaço de projeto, através das estimativas extraídas de modelos UML.

Finalmente, a abordagem proposta para estimativa e exploração apresentada nesta dissertação poderia ser incorporada a um processo de desenvolvimento baseado em modelos, como em uma abordagem de desenvolvimento MDA, onde os passos de transformação dos modelos sejam guiados por resultados de estimativa e exploração obtidos através da abordagem proposta.

REFERÊNCIAS

BECK FILHO, A.C. S. et. al. CACO-PS: a General Purpose Cycle-Accurate Configurable Power Simulator. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 2003. **Proceedings...** Los Alamitos: IEEE Computer Society Press, 2003. p.349-354.

BERNARDI, S.; DONATELLI, S.; MERSEGUER, J. From UML Diagrams and Statecharts to analyzable Petri Nets. In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, WOSOP, 2002. **Proceedings…** Roma: ACM Press, 2002. p. 35-45.

BONTEMPI, G.; KRUIJTZER, W. A Data Analysis Method for Software Performance Prediction. In: DESIGN, AUTOMATION AND TEST IN EUROPE CONFERENCE, DATE, 2002, Paris. **Proceedings…** Paris: [s.n.], 2002. p. 971–976.

BRANDES, U.; EIGLSPERGER, M.; LERNER, J. **GraphML Primer**. Graph Drawing. [S.l.]:Disponível em: <http://graphml.graphdrawing.org/primer/graphml-primer.html>. Acesso em: out. 2005

DOUGLASS, B. **Real-Time Design Patterns:** Robust Scalable Architecture for Real-Time Systems. Boston: Addison Wesley, 2003.

ERBAS, C.; ERBAS, S.E.; PIMENTEL, A.D. A Multi objective Optimization Model for Exploring Multiprocessor Mappings of Process Networks. In: INTERNATIONAL WORKSHOP ON HARDWARE/SOFTWARE CODESIGN, CODES-ISSS, 2003, Newport Beach. **Proceedings…** [S.l.]: ACM Press, 2003.

EBERT, J. LP Solve Java Wrapper. LP SOLVE. Disponível em: <http://lpsolve.sourceforge.net/5.5/> Acesso em: 14 out. 2005.

GEILEN M. C. W.; et. al. Modeling and Specification Using SHE. **Journal of Computer Languages**, [S.l.], v.27, n.3, p.19-38, Dec. 2001.

GIVARGIS, T.; VHID, F. Platune: a tuning framework for system-on-a-chip platforms. **IEEE Transaction on computer-aided design of integrated circuits and systems**, [S.l.], v.21, n.11, p.1317 - 1327, Nov. 2002.

GIVARGIS, T.; VHID, F.; HENKEL, J. System-level Exploration for Pareto-optimal Configurations in Parameterized System-on-a-chip. **IEEE Transactions on Very Large Scale Integration Systems**, [S.l.], v.10, n.4, p. 579-592, Aug. 2002.

GRAAF, B.; LORMANS, M.; TOETENEL, H. Embedded software engineering: the state of the practice. **IEEE Software**, [S.l.], v. 20, n. 6, p. 61 – 69, Nov. – Dec. 2003.

GRIES, M. Method for Evaluating and Covering the Design Space During Early Design Development. **Integration, the VLSI Journal**, [S.l.], v. 38, n. 2, p.131-183, 2004.

GU, G. P.; PETRIU, D. C. Early evaluation of *software* performance based on the UML performance profile. In: CONFERENCE OF THE CENTRE FOR ADVANCED STUDIES ON COLLABORATIVE RESEARCH, 2003. **Proceedings…** Toronto:[s.n.], 2003. p.66-79.

HOEBEN, F. Using UML Models for Performance Calculation. In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, WOSP, 2., 2000. **Proceedings…** Ontário: ACM Press, 2000. p. 77-82.

ITO, S.; CARRO, L.; JACOBI, R. Making Java Work for Microcontroller Applications. **IEEE Design & Test of Computers,** [S.l.], v.18, n.5, Sept.-Oct. 2001.

KAHN, G. The Semantics of a Simple Language for Parallel Programming. In: IFIP CONGRESS, 1974, Stockholm. **Information Processing 74**: Proceedings of IFIP Congress 74. Amsterdam: North-Holland, 1974. p. 471-475.

LAVAGNO, L.; MARTIN, G.; SELIC, B. (Ed.). **UML for Real**: Design of Embedded Real Time Systems. Dordrecht: Kluwer Academic Publishers, 2003.

LEDECZI, A. et al. Modeling Methodology for Integrated Simulation of Embedded Systems. **ACM Transactions on Modeling and Computer Simulation**, [S.l.], v. 13, n. 1, p. 82-103, Jan. 2003.

LI, Y. S.; MALIK, S. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In: DESIGN AUTOMATION CONFERENCE, DAC, 1995. **Proceedings…** [S.l.]: ACM press, 1995. p. 456-461.

LIEVERSE, P. A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems. **Journal of VLSI Signal**, [S.l.], v. 29, n.3, p. 197-207, 2001.

LP SOLVE. Disponível em: <http://lpsolve.sourceforge.net/5.5/> Acesso em: 14 out. 2005.

MRAIDHA, C. et al. MDA Platform for Complex embedded System Development. In: IFIP WORKING CONFERENCE ON DISTRIBUTED AND PARALLEL EMBEDDED SYSTEMS, DIPES, 2004, Toulouse. **Proceedings...** Boston: Kluwer Academic Publishers, 2004. p.1-10.

MARTIN, G.; MÜLLER, W. (Org.). **UML for SoC Design.** Dordrecht: Kluwer Academic Publishers, 2005.

MARWEDEL, P**. Embedded System Design.** Dordrecht: Kluwer Academic Publishers, 2003.

MATTOS, J. C. B. et al. Design Space Exploration with Automatic Generation of IP-based Embedded Software. In: IFIP WORKING CONFERENCE ON DISTRIBUTED AND PARALLEL EMBEDDED SYSTEMS, DIPES, 2004, Toulouse. **Proceedings...** Boston: Kluwer Academic Publishers, 2004. p.237-246.

NEEMA, S.; SZTIPANOVITS, J.; KARSAI, G. Constraint-Based Design-Space Exploration and Model Synthesis. In: INTERNATIONAL CONFERENCE ON EMBEDDED SOFTWARE, EMSOFT, 2003, Philadelphia. **Proceedings…** [S.l.]: ACM Press, 2003. p. 290-305.

OLIVEIRA, M. F. S. et al. An Embedded SW Design Exploration Approach Based on UML Estimation Tools. In: IEEE EMBEDDED SYSTEMS SYMPOSIUM, 2005, Manuaus. **Proceedings…** New York: IFIP, 2005. p. 45-54.

OMG – OBJECT MANAGEMENT GROUP. **Meta Object Facility.** 2002. Disponível em: <http://www.omg.org/mof>. Acesso em: 14 jan. 2005.

OMG – OBJECT MANAGEMENT GROUP. **Model Driven Architecture Guide.** 2004. Disponível em: <http://www.omg.org/mda>. Acesso em: 01 fev. 2005.

OMG – OBJECT MANAGEMENT GROUP. UML - **Unified Modeling Language.** 2005. Disponível em: <http://www.omg.org/uml>. Acesso em: 14 jan. 2005.

OMG – OBJECT MANAGEMENT GROUP. **UML Profile for Schedulability, Performance and Time Specification.** 2002. Disponível em: <http://www.omg.org/cgi-bin/doc?ptc/02-03-03>. Acesso em: 14 jan. 2005.

OMG – OBJECT MANAGEMENT GROUP. **XML Metadata Interchange.** 2002. Disponível em: <http://www.omg.org/xmi>. Acesso em: 01 fev. 2005.

PIMENTEL, A.D.; ERBAS, C.; POLSTRA, S. A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels. **IEEE Transactions on Computers,** [S.l.], v. 55, n. 2, p. 99-112, Feb. 2006.

PIMENTEL, A.D. et al. Exploring Embedded-Systems Architectures with Artemis. **IEEE Transactions on Computers,** [S.l.], v. 34, n. 11, p. 57-63, Nov. 2001.

PETRIU, D.C.; WOODSIDE, C. M. Performance analysis with UML: layered queueing models from the performance profile. In: LAVAGNO, L.; MARTIN, G.; SELIC, B.; **UML for Real:** Design of Embedded Real Time Systems. Dordrecht: Kluwer Academic Publishers, 2003.

RUSSELL, J. T.; JACOME, M. F. Architecture-level Performance Evaluation of Component Based Embedded Systems; In: DESIGN AUTOMATION CONFERENCE, DAC, 2003, Anaheim. **Proceedings…** [S.l.]: ACM Press, 2003. p. 396-401.

SANGIOVANNI-VICENTELLI, A.; MARTIN, G. Platform-based Design and Software Design Methodology for Embedded Systems. **IEEE Design and Test for Computer**, [S.l.], v. 18, n. 6, p. 23-33, 2001.

SANGIOVANNI-VICENTELLI, A. et al. Benefits and Challenges for Platform Based. In: DESIGN AUTOMATION CONFERENCE, DAC, San Diego, 2004. **Proceedings…** [S.l.]: ACM Press, 2004. p. 409-414.

SHANDLE, J.; MARTIN, G. Making Embedded Software reusable for SoCs. **EEDesing**, [S.l.], Mar. 2002. Disponível em: <http://www.eedesign.com/story/OEG20020301S0104>. Acesso em: 10 nov. 2005.

SMITH, C. U.; WILLIAMS, L. G. Performance Engineering Evaluation of Object-Oriented Systems with SPE*ED. In: INTERNATIONAL CONFERENCE ON MODELLING TECHNIQUES AND TOOLS FOR COMPUTER PERFORMANCE EVALUATION, 1997. **Computer Performance Evaluation:** Modeling Techniques and Tools: Proceedings. Berlin: Springer-Verlag, 1997. p.135-154. (Lecture Notes in Computer Science, v. 1245).

SMITH, C. U.; WILLIAMS, L. G. Software Performance Engineering. In: LAVAGNO, L.; MARTIN, G.; SELIC, B. **UML for Real:** Design of Embedded Real Time Systems. Dordrecht: Kluwer Academic Publishers, 2003.

TALARICO, C. et al. A New Framework for Power Estimation of Embedded Systems. **IEEE Computer,** [S.l.], v. 38, n. 2, p.71-78, 2005.

THEELEN, B.D.; PUTTEN, P.H.A. van der; VOETEN, J.P.M. Using the SHE method for UML-based performance modeling. In: SYSTEM SPECIFICATION AND DESIGN LANGUAGES, 2003. **Proceedings…** Dordrecht: Kluwer Academic Publishers, 2003. p.143-160.

WANDELER, E.; THIELE, L. Abstracting Functionality for Modular Performance Analysis of Hard Real-Time Systems. In: ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 2005. **Proceedings…** Shanghai: [s.n.], 2005. p. 697-702.

WEHRMEISTER, M. A**. Framework Orientado a Objetos para Projeto de *Hardware* e *Software* Embarcados para Sistemas de Tempo Real**. 2005. 104 f. Dissertação (Mestrado em Ciências da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

WILLIAMS, L. G.; SMITH, C. U. Performance evaluation of *software* architectures. In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, 1. 1998. **Proceedings…**Santa Fe: ACM Press, 1998. p.164-177.

WILLIAMS, L. G.; SMITH, C. U. PASASM: a method for the performance assessment of software architectures, In: INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE, 3., 2002. **Proceedings…** Roma: ACM Press, 2002. p. 179-189.