UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Alex de Magalhães Machado

UM ESTUDO SOBRE A MODELAGEM QUANTITATIVA DE REQUISITOS DE SISTEMAS EMBARCADOS VISANDO A EXPLORAÇÃO DO ESPAÇO DE PROJETO

Florianópolis

2012

Alex de Magalhães Machado

UM ESTUDO SOBRE A MODELAGEM QUANTITATIVA DE REQUISITOS DE SISTEMAS EMBARCADOS VISANDO A EXPLORAÇÃO DO ESPAÇO DE PROJETO

Dissertação submetida ao Programa de Pós Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Antônio Augusto Fröhlich, Dr.

Florianópolis

2012

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

Alex de Magalhães Machado

UM ESTUDO SOBRE A MODELAGEM QUANTITATIVA DE REQUISITOS DE SISTEMAS EMBARCADOS VISANDO A EXPLORAÇÃO DO ESPAÇO DE PROJETO

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 30 de setembro 2012.

Prof. Ronaldo dos Santos Mello, Dr.
Coordenador

Prof. Antônio Augusto Fröhlich, Dr.
Orientador

Banca Examinadora:

Rivalino Matias Júnior
Universidade Federal de Uberlândia

Prof. Dr. Marco Aurélio Wehrmeister
Universidade Estatual de Santa Catarina

A todos aqueles que sentiram como se essa conquista também fosse deles.

AGRADECIMENTOS

De acordo com Cícero, "a gratidão não é somente a maior de todas as virtudes, mas também a mãe de todas as outras". Reservo então este espaço para agradecer a todos que me apoiaram durante todo o processo. Agradeço aos meus pais por todas as portas que apareceram no meu caminho. Agradeço à Universidade Federal de Santa Catarina, ao Programa de Pós-Graduação em Ciência da Computação e ao Laboratório de Integração de Software/Hardware por todas as oportunidades. Agradeço também ao Prof. Dr. Antônio Augusto Medeiros Fröhlich por abrir as portas do Mestrado para mim e por ter acreditado desde o início no meu potencial. Finalmente, agradeço à família LISHA por ter me recebido de braços abertos, principalmente ao Alexandre, Arliones, Cancian, Djulyan, Hugo, Giovani, Mateus, Nicole, Rita, Rodrigo, Tiago e Wesley.

Também agradeço a todos que acreditaram em mim até mesmo nas horas mais escuras, quando até mesmo eu estava duvidando do meu potencial. Em especial, agradeço ao Prof. Dr. Mario Antônio Ribeiro Dantas por apostar em mim. Agradeço à Barbara, à Jay e à Mariana pelo conforto e pelas risadas quando eu precisava desestressar. Agradeço ao Arthur, à Cecilia, à Janaina, ao Luis e ao meu irmão Mateus pelas palavras sábias e acolhedoras e pelo apoio singular. Agradeço ao Ivo e à Malu por todo o amor e carinho, por todas as broncas e palavras de incentivo, pelas risadas e pelas lágrimas, e por estarem ao meu lado durante cada passo que eu dei desde que me conheceram. Não sei o que seria de mim sem vocês.

It is always darkest just before the day dawneth.

Thomas Fuller

RESUMO

Sistemas embarcados geralmente executam uma ou poucas funções dedicadas e possuem requisitos funcionais e não-funcionais altamente especializados. Dessa forma, cada sistema embarcado pode ser muito diferente dos outros, dependendo da tarefa que ele deverá executar e das restrições impostas para essa execução. Além disso, muitos desses requisitos são conflitantes. Reduzir o consumo de energia de um sistema embarcado pode ocasionar uma perda de desempenho ou um maior uso de memória, ou até mesmo um maior custo total para o sistema. Encontrar um meio termo que satisfaça todos os requisitos do sistema pode ser uma tarefa inviável de se realizar sem a utilização de ferramental apropriado.

Sistemas muito específicos e com diversos requisitos conflitantes acabam exigindo um amplo planejamento de custos que capture todas as especificidades e também satisfaça todos os requisitos. Para facilitar esse planejamento, nós propomos a modelagem do custo de um sistema embarcado através de equações matemáticas. Essas equações abstraem os principais requisitos do sistema e podem ser aplicadas para um grande número de sistemas embarcados, permitindo que os projetistas modelem os principais custos do sistema apenas com base nessas equações. Através de estudo de caso, confirmamos a adequabilidade das equações para o planejamento de sistemas embarcados e mostramos as vantagens em relação a outras formas de modelagem.

Palavras-chave: Sistemas Embarcados. Exploração do espaço de projeto. Modelagem de requisitos.

ABSTRACT

Embedded systems usually perform one or a few dedicated tasks and have highly focused functional and non-functional requirements. This way, every embedded system may be very different from the others, depending on the tasks it is supposed to execute and the inherent restrictions of that execution. Besides, most of these requirements are conflictive. Reducing power consumption of a embedded system might cause loss of performance, higher memory usage, or even make the final system more expensive. Finding an intermediate solution that satisfies all system requirements may be unfeasible to achieve without using proper tools.

Systems with very specific purposes and several conflicting requirements end up also requiring specific and vast cost planning. This planning must capture all the system's specificities and also satisfy all requirements. To ease this task, we propose an embedded system cost modeling based on mathematical functions. These functions abstract the system requirements and can be used with many types of embedded systems, allowing designers to model all the system's costs only based on these equations. Through a study case, we have been able to ensure the suitability of our model to embedded system design and cost planning, and we have shown its advantages compared to other modeling techniques.

Keywords: Embedded systems. Design space exploration. Requirement modeling.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASIC Application-specific Integrated Circuit

ASIP Application-specific Instruction-set Processor

CPN Coloured Petri Nets

CPU Central Processing Unit

DPM Dynamic Power Management

DSE Design Space Exploration

DSP Digital Signal Processor

DVS Dynamic Voltage Scaling

EDG Execution Dependency Graph

FPGA Field-programmable Gate Array

HAP Heterogeneous Assignment with Probability

ILP Integer Linear Programming

IP Intellectual Property

IPI Inter-Prefetch Interval

JVM Java Virtual Machine

P2P Peer-to-Peer

PLD Programmable Logic Device

RAM Random Access Memory

ROM Read-Only Memory

RTL Register-Transfer Level

SAN Stochastic Automata Network

SoC System on Chip

TLB Translation Lookaside Buffer

TTM Time To Market

UART Universal Asynchronous Receiver/Transmitter

WCET Worst-Case Execution Time

WSN Wireless Sensor Network

SUMÁRIO

1 INTRODUÇÃO

Um sistema embarcado é um sistema computacional projetado para executar um conjunto específico de tarefas. É comum também que esses sistemas fiquem embutidos dentro de outros, e essas tarefas acabem contribuindo para o funcionamento de um ou mais sistemas maiores. Essas tarefas podem variar muito, visto que há uma gama enorme de aplicações para sistemas embarcados no nosso cotidiano. Devido a essa variabilidade, há poucas técnicas e metodologias de projeto de sistemas que possam ser utilizadas para qualquer tipo de sistema embarcado. Além disso, muitas dessas técnicas e metodologias foram desenvolvidas inicialmente para sistemas de propósito geral, e portanto precisam ser repensadas quando aplicadas a sistemas embarcados (SIMON, 1999).

A utilização dos sistemas embarcados está cada vez mais presente na vida das pessoas. Eles estão presentes em relógios digitais, impressoras, televisores, semáforos, lombadas eletrônicas, tocadores de MP3, sistemas de controle de tráfego áereo, entre muitos outros. E ao mesmo tempo em que a utilização desses sistemas cresce, o mercado também passa a investir mais nessa área e a competição entre fabricantes se acentua. É importante não apenas fornecer um produto, mas principalmente desenvolvê-lo no menor tempo possível e fabricá-lo a baixos custos (WOLF, 2003).

O projeto de um sistema embarcado deve então tratar diversas questões referentes a requisitos funcionais e não funcionais do sistema, de forma mais crítica do que ocorre em sistemas de propósito geral. Ao mesmo tempo, devido à diversidade de finalidades para esses sistemas, algumas técnicas e metodologias podem ser menos eficazes ou até mesmo não serem aplicáveis a algumas categorias de sistemas (SIMON, 1999).

A primeira etapa do projeto de um sistema embarcado é então a definição de requisitos funcionais, os quais variam amplamente entre diferentes projetos. Sistemas embarcados se encontram por toda parte, e cada um possui sua própria finalidade, com seu próprio conjunto de funcionalidades. Para que haja reuso no planejamento desses diferentes sistemas, é desejável que se use uma única forma de representação de requisitos, a qual deve capturar as diferentes funcionalidades (MARWEDEL, 2003).

O custo financeiro de um sistema embarcado é o requisito não funcional mais simples. Sempre busca-se um sistema final que seja o

mais barato possível. Porém, por possuírem finalidades variadas, cada sistema embarcado também possui seus próprios requisitos não funcionais, e esses também podem variar muito. Alguns sistemas, por serem operados por baterias, devem ficar ligados funcionando por longos períodos de tempo, e por isso precisam consumir pouca energia. Há sistemas que precisam responder a eventos externos sem atrasos, e outros que devem utilizar pouca memória ou ocupar espaços muitos pequenos (BOEHM et al., 1995).

Sistemas embarcados geralmente executam uma ou poucas funções dedicadas e possuem requisitos funcionais e não funcionais altamente focados. Um fator agravante desse cenário é que muitos desses requisitos são conflitantes. Reduzir o consumo de energia de um sistema pode ocasionar uma perda de desempenho ou um maior uso de memória, ou até mesmo um maior custo total para o sistema. A maior parte dos projetistas, quando se depara com esses problemas, tende a reduzir o problema, diminuindo a quantidade de requisitos, ou tornando-os mais flexíveis. Com menos requisitos, fica mais fácil também representá-los utilizando uma abordagem pouco abrangente, específica do projeto. Uma abordagem assim, porém, prejudica a reusabilidade das técnicas empregadas e dos próprios sistemas (SIMON, 1999).

Por outro lado, se os requisitos não puderem ser simplificados, é preciso encontrar uma solução intermediária que satisfaça todos os requisitos do sistema, o que pode ser uma tarefa inviável sem a utilização de ferramentas e modelos bem definidos. Um dos maiores desafios é representar todos os diferentes requisitos de uma forma homogênea e reusável. Cada projeto costuma usar sua própria modelagem, geralmente representando cada requisito de uma forma diferente (WOLF, 2003).

Porém, mesmo com modelos definidos, os projetistas podem enfrentar dificuldades durante a tomada de decisões, visto que há muitos fatores a respeito do funcionamento do sistema final que são apenas estimativas durante as etapas iniciais do projeto, e decisões tomadas erroneamente nessas etapas acabam influenciando muito todo o restante do projeto. Dessa forma, uma grande pesquisa nessa área tem sido motivada por todas as dificuldades inerentes dessas etapas iniciais do projeto de um sistema embarcado e também pela importância que essas etapas têm para o sucesso de todo o projeto (CHULANI; BOEHM; STEECE, 1999).

É essencial que seja realizado um planejamento amplo e que ao mesmo tempo considere diversas especificidades desses sistemas embarcados, de forma a satisfazer todos os requisitos. Esse planejamento é

realizado geralmente através da utilização de modelos de custo, que consideram os diversos requisitos do sistema e tentam balanceá-los de forma a realizar uma exploração do espaço de projeto. Porém, de forma geral, há poucos casos no mundo em que essa exploração de espaço de projeto é realizada levando-se em consideração tantos fatores diferentes, como consumo de energia, área, peso, utilização de memória, desempenho, preço do sistema, etc. Geralmente leva-se em consideração apenas um ou dois aspectos, ou então apenas um pequeno nicho de aplicações alvo. Os modelos existentes, então, não são suficientes para todos os sistemas embarcados, prejudicando o reuso dos modelos (MARWEDEL, 2003).

Para resolver esse problema propõe-se a modelagem dos custos de um sistema embarcado através de equações matemáticas amplas e ao mesmo tempo que tratem especificidades. Essas equações abstraem os requisitos do sistema, tratam as incertezas inerentes a eles, e podem ser aplicadas para qualquer tipo de sistema embarcado, permitindo que os projetistas modelem todos os requisitos do sistema apenas com base nessas equações.

1.1 OBJETIVOS

Esta dissertação tem como hipótese de pesquisa que é possível modelar os principais custos de um sistema embarcado visando a exploração do espaço de projeto através de uma única linguagem. Desta forma, o principal objetivo desta dissertação é criar um único modelo que possa ser utilizado para modelar os principais custos de um sistema, de forma a possibilitar uma exploração do espaço de projeto. Não faz parte, porém, do espaço deste trabalho realizar essa exploração do espaço de projeto. Para mostrar isso, estudou-se diferentes modelagens existentes para sistemas embarcados e suas limitações. Estudou-se também como trabalhos relacionados costumam modelar os principais requisitos de um sistema embarcado, e como essa modelagem pode ser estendida para qualquer tipo de requisito.

Este trabalho de pesquisa possui os seguintes objetivos específicos:

- Identificar os principais custos de sistema que a comunidade científica busca modelar.
- Identificar que tipos de modelagens a comunidade científica usa para representar os principais custos de sistema.
- Demonstrar que é possível representar todos esses principais cus-

tos usando uma única linguagem.

- Demonstrar que a linguagem matemática através de equações matemáticas é adequada para a modelagem de custos de sistemas embarcados.

- Identificar quais tipos de sistemas embarcados e quais tipos de custos de sistema a modelagem proposta suporta.

- Demonstrar que é possível, através unicamente da modelagem proposta, modelar os todos os principais custos de um sistema embarcado dentre os suportados pelo modelo.

- Demonstrar que a modelagem proposta é adequada para a utilização na exploração do espaço de projeto.

- Demonstrar as possibilidades de trabalhos futuros e de extensões do trabalho realizado, abrindo portas para a continuidade da pesquisa.

A principal contribuição desse trabalho consiste então em apresentar um modelo reusável para a modelagem dos principais custos de um sistema embarcado, que leve em consideração aspectos específicos de cada requisito do sistema ou do projeto.

1.2 METODOLOGIA

A metodologia utilizada para demonstrar a adequabilidade dos modelos propostos foi a criação de um estudo de caso onde os principais custos de um sistema embarcado são modelados com base em um repositório de componentes com dados obtidos por estimativas dos projetistas e também de tabelas fornecidas pela indústria. Nesse estudo de caso considerou-se 5 diferentes nodos sensores: Mica-2, Mica-Z, Telos, EposMote e ZigBit (KRÄMER; GERALDY, 2006; POLASTRE; SZEWCZYK; CULLER, 2005; LISHA, 2012). Para os custos referentes a componentes de software, considerou-se uma implementação completa do sistema no sistema operacional EPOS.

Pode-se dividir a metodologia utilizada neste trabalho nos seguintes tópicos:

- Identificar os principais custos de sistemas embarcados e como a comunidade científica costuma modelá-los.

- Propor um modelo que possa ser usado para todos esses principais custos, e que possa vantagens em relação aos existentes, ao mesmo tempo que facilite a exploração do espaço de projeto.

- Demonstrar que é possivel modelar os principais custos de um sistema embarcado com base no modelo proposto.

Dessa forma, propõe-se uma modelagem matemática para sistemas embarcados, abstraindo características dos seus diferentes componentes e classificando-as em diferentes indicadores de qualidade. Esses indicadores podem ser consumo de energia, utilização de memória, desempenho, custo, disponibilidade no mercado, experiência da equipe, entre outras. O resultado disso são equações matemáticas que abstraem um sistema embarcado completo e podem ser usadas para avaliar a qualidade de uma solução de sistema embarcado.

1.3 ORGANIZAÇÃO DO TEXTO

De forma a descrever com clareza todas as etapas do trabalho desenvolvido, o texto foi organizado da seguinte forma. O capítulo 2 tem como objetivo descrever conceitos básicos a respeito de sistemas embarcados, quais são os diferentes tipos de sistemas embarcados e suas peculiaridades. Serão discutidas características que diferenciam os sistemas embarcados dos demais sistemas computacionais. Esses conceitos serão importantes para o compreendimento das dificuldades encontradas durante o projeto desses sistemas embarcados.

É descrito então em detalhes como é o projeto de um sistema embarcado e que preocupações os projetistas devem ter ao iniciar um novo projeto. Será mostrado também como é o processo de definição de requisitos, dando exemplos reais de diversas aplicações cotidianas, além de mostrar os vários conflitos que podem aparecer durante essa etapa de definição de requisitos e como os projetistas precisam tomar decisões importantes nessas fases iniciais do projeto. Será descrito como esses requisitos podem ser modelados de forma que essa tomada de decisões seja facilitada. Diferentes formas de modelagem serão apresentadas, e suas peculiaridades serão discutidas. Por fim, será mostrado como os modelos podem ser usados para realizar uma exploração de espaço do projeto, a fim de se identificar soluções satisfatórias para o projeto.

O capítulo 3 tem como objetivo apresentar uma especialidade da modelagem de requisitos, a modelagem matemática, através de equações matemáticas. Primeiramente será mostrado como os diversos

requisitos funcionais e não funcionais de um sistema embarcado podem ser expressos matematicamente, e também como custos diversos podem ser modelados através de equações. Serão apresentados meios de se realizar um tratamento de incertezas, visto que é comum que os projetistas não tenham certeza a respeito dos dados de várias variáveis. É comum também que algumas variáveis dependam de outras, então será mostrado como essa dependência pode ser modelada nas equações. Por fim, será descrito como a literatura existente modela os requisitos mais comuns de sistemas embarcados, como baixo consumo de energia, alto desempenho, pouco uso de memória, entre outros.

No capítulo 4 finalmente será apresentada em detalhes toda a modelagem proposta para o projeto de sistemas embarcados. Todas as equações serão apresentadas e cada termo de cada equação será descrito, justificando seu uso com exemplos reais. Também será demonstrado como é possível criar várias novas equações a partir de diferentes requisitos, e como as equações podem ser usadas em conjunto para identificar-se os melhores componentes de um sistema embarcado.

Tendo cada equação descrita e justificada, no capítulo 5 será mostrado como o modelo proposto pode ser usado na prática, através de um estudo de caso real e ao mesmo tempo didático, onde várias decisões de projeto puderam ser tomadas utilizando as equações apresentadas. O capítulo 6 finaliza o trabalho de pesquisa dessa dissertação, sumarizando os resultados obtidos e discutindo extensões do trabalho desenvolvido.

2 MODELAGEM DE REQUISITOS E A EXPLORAÇÃO DO ESPAÇO DE PROJETO

Este capítulo faz uma revisão das principais metodologias de projeto de sistemas embarcados. Serão apresentados conceitos básicos e peculiaridades que os tornam diferentes dos outros sistemas computacionais. Depois será discutido o projeto de sistemas embarcados, desde as etapas iniciais de definição de requisitos. Será mostrado também os vários conflitos que podem aparecer durante essa etapa, e como os projetistas precisam tomar decisões importantes nessas fases iniciais do projeto. Será apresentado como esses requisitos podem ser modelados de forma que essa tomada de decisões seja facilitada, que diferentes formas de modelagem existem, e quais as suas peculiaridades. Por fim, será mostrado como os modelos podem ser usados para realizar uma exploração de espaço do projeto, a fim de se identificar soluções satisfatórias para o projeto.

2.1 PRINCÍPIOS DE SISTEMAS EMBARCADOS

Sistemas embarcados são sistemas que realizam um conjunto específico de tarefas definido no início do projeto do sistema, ao contrário de sistemas de propósito geral. Um sistema embarcado geralmente não é o todo, e sim um componente de um sistema maior. Esse componente é composto por hardware e software que juntos realizam as tarefas específicas definidas pelos projetistas do sistema. A forma com que esses sistemas se comunicam com o meio ou até mesmo com usuários humanos é bastante específica das aplicações do sistema (SIMON, 1999).

Por exemplo, relógios digitais possuem geralmente alguns botões que são utilizados pelo usuário, e também possuem um simples visor onde informações como horário atual e data são exibidos. Lombadas eletrônicas medem a velocidade dos veículos e se comunicam com os usuários informando a velocidade medida e também se esse valor está dentro do permitido ou não. Porém, sistemas de trânsito como lombadas eletrônicas e semáforos não permitem nenhuma forma de comunicação por parte do usuário. Há também sistemas que se comunicam apenas com outros sistemas, e jamais com seres humanos, como é o caso de sensores de temperatura, que após detectarem um aumento significativo da temperatura ativam um outro sistema responsável pela refrigeração do ambiente. Um navio de cruzeiros pode possuir diversos

sistemas comunicando-se entre si e também com a tripulação, de forma a detectar exatamente como está a situação do navio e permitir atitudes rápidas e eficazes por parte do comandante em caso de emergência (SIMON, 1999).

Assim como semáforos e relógios digitais, inúmeros outros sistemas embarcados estão ao redor das pessoas a todo momento, cada vez mais presentes na vida de cada um. O número de sistemas embarcados já superou o número de habitantes no planeta, e esse número continua crescendo em ritmo acelerado (POP, 2005). Dessa forma, a área de sistemas embarcados acaba desempenhando um papel muito importante na economia mundial, com um mercado competitivo que só cresce.

A complexidade dos sistemas embarcados é outro fator que vem crescendo constantemente. A maioria dos sistemas embarcados possui um projeto integrado de software e hardware, e graças aos avanços nas tecnologias de semicondutores, aplicações cada vez mais complexas vem surgindo. Ao mesmo tempo que se tornam mais complexas, essas aplicações também possuem requisitos cada vez mais rigorosos para que possam ser inseridas com sucesso no mercado. Essas restrições tornam o projeto de sistemas embarcados cada vez mais complexo e cada vez mais importante (WOLF, 2003).

As tecnologias usadas no projeto desses sistemas também variam bastante, e diversas arquiteturas distintas podem ser usadas, como um microcontrolador, um Digital Signal Processor (DSP), um Programmable Logic Device (PLD)/Field-programmable Gate Array (FPGA), um Application-specific Integrated Circuit (ASIC), ou até mesmo um System on Chip (SoC), onde todo o projeto é implementado numa única pastilha de silício, aumentando ainda mais o grau de integração e a complexidade do sistema. Com essas tecnologias e aplicações complexas, a área de sistemas embarcados acaba contemplando sistemas muito diferentes uns dos outros, cada um com seu conjunto próprio de requisitos (WOLF, 2003). A seguir essas características de sistemas embarcados serão mostradas com três exemplos reais.

O primeiro exemplo é um leitor de código de barras sem fio, que pode ser utilizado em supermercados. Os requisitos funcionais do sistema são simples: quando um usuário apertar um botão, o leitor de código de barras deve ativar seu laser para ler o código de barras. Além disso, quando um código de barras é lido, ele deve ser enviado para algum computador ou máquina registradora. Uma característica importante desse sistema é que ele não possui fios, logo sua única fonte de energia deve ser sua bateria. Além disso, como o usuário deve segurar o leitor de código de barras com uma mão, o peso da bateria é

limitado pelo peso que um usuário normal consegue segurar confortavelmente. Uma primeira questão a respeito da bateria é quanto tempo os projetistas desejam que ela dure. Pode-se definir um requisito não funcional do sistema como sendo então que a bateria do sistema dure pelo menos um turno de 8 horas, ou o tempo que o supermercado permanecer aberto. Após esse período, a bateria do leitor deve ser recarregada (SIMON, 1999).

O segundo exemplo é uma impressora laser. Uma impressora possui mais requisitos funcionais do que o leitor de código de barras. Ela deve ser capaz de coletar dados das diversas portas de comunicação que possui, deve possuir diversos botões no painel de controle para o usuário se comunicar com ela, deve apresentar mensagens ao usuário no visor do painel de controle, deve perceber quando alguma folha de papel fica presa no dispositivo, deve realizar a recuperação da impressão de forma apropriada, deve perceber quando ela fica sem folhas disponíveis, deve realizar a impressão de novas folhas, etc. Um requisito não funcional desse sistema é que as impressões sejam rápidas e durem poucos segundos, e isso pode exigir que partes do sistema sejam desenvolvidas especialmente para cumprir esse requisito (SIMON, 1999).

O terceiro exemplo é um nodo sensor de uma Wireless Sensor Network (WSN), que vem a ser uma sub-área de pesquisa de sistemas embarcados e redes de computadores. Uma WSN consiste em nodos sensores autônomos distribuídos num espaço com o objetivo de monitorar condições físicas do ambiente, como temperatura, som, pressão, vibração, movimento, entre outras, e repassar essas informações para um computador central. Uma dessas redes pode possuir centenas ou milhares de nodos espalhados, e cada um deles pode possuir um ou mais sensores. Os requisitos funcionais de um nodo consistem em utilizar seus sensores para medir as condições desejadas do ambiente, reportar a um controle central os dados coletados e ativar algum sistema específico quando os dados coletados atingirem valores críticos. Um requisito não funcional é que ele funcione sem interrupções durante um longo período de tempo, por exemplo um ano, apenas usando uma bateria como fonte de energia, pois ele ficará longe do contato com seres humanos. Um outro requisito não funcional que pode fazer parte do projeto desse sistema é que as tarefas que ele executa não possam atrasar, ou seja, a execução das tarefas possui um limite de tempo predeterminado e considera-se um erro quando o sistema demora mais do que o limite para executar uma tarefa. Esse tipo de sistema é chamado de sistema de tempo real (MARWEDEL, 2003).

Os três sistemas embarcados descritos acima possuem requisitos

muito diferentes. Devido a essa variabilidade, as técnicas e metodologias para o projeto de sistemas embarcados têm sido objeto de estudo da comunidade científica. Através desses requisitos se inicia o projeto, com a modelagem do sistema. Porém, os projetistas precisam tratar vários problemas e conflitos causados pela definição de requisitos. Esses desafios serão discutidos nas próximas seções (WOLF, 2003).

2.2 COMPONENTES DE UM SISTEMA EMBARCADO

Pode-se representar um sistema embarcado de formas diferentes e essa representação está diretamente relacionada ao processo de desenvolvimento do sistema. Tanto hardware quanto software podem se beneficiar enormemente através de um projeto baseado em componentes. Esse tipo de projeto consiste em se dividir o sistema todo em pequenas partes chamadas componentes, e então projetá-las separadamente, como unidades independentes. Finalmente, na geração do sistema final, as partes são acopladas e o sistema é criado. A maior vantagem dessa abordagem é a reusabilidade que ela traz para o projeto. Componentes, por serem unidades independentes, podem ser reusados em outros projetos, sem que haja retrabalho no projeto do componente. Se utilizada corretamente, essa abordagem traz diversos benefícios, como redução no tempo de desenvolvimento dos sistemas. Acredita-se que essa abordagem é a mais indicada para o projeto de sistemas embarcados, e por isso todo o desenvolvimento deste trabalho terá como foco o desenvolvimento baseado em componentes (SAMETINGER, 1997).

Para um observador externo, as diferenças mais visíveis entre os três sistemas descritos na seção anterior são o tamanho, o formato e os tipos de componentes mecânicos usados. O leitor de código de barras é um aparelho que um usuário irá segurar com uma mão, logo ele não pode ser muito grande nem pesado. Além disso, ele possui um leitor a laser para fazer leituras de código de barras e não possui fios. A impressora laser, por sua vez, deve ser maior e pesada, devido ao seu mecanismo de impressão e o armazenamento de folhas de papel. Ela está ligada à rede elétrica, e portanto não precisa de uma bateria. O nodo sensor usado em redes de sensores sem fio como o que descrevemos acima não possui nenhuma forma de comunicação com usuários, não possui mecanismos para leitura ou impressão, não possui fios e deve ser muito menor. Um nodo sensor pode ter 2 metros de altura, por exemplo, ou menos de um milímetro (HART; MARTINEZ, 2006). Consideramos que o sensor descrito acima como exemplo possui cerca de

10 cm de comprimento. O formato do nodo é apenas uma cápsula que engloba todos os componentes de hardware e alguns sensores para medir condições do ambiente, que não estavam presentes nem no leitor de código de barras e nem na impressora. Além disso, o sistema do nodo pode ser de tempo real, exigindo que as suas funcionalidades sejam realizadas dentro de um tempo predeterminado.

Com base na descrição dos sistemas acima, pode-se já deduzir alguns componentes de hardware presentes nos sistemas. Uma unidade de processamento, seja um microprocessador, ou um Application-specific Instruction-set Processor (ASIP), ou uma FPGA, é o principal componente desses sistemas embarcados e está presente em todos, embora possua características distintas em cada um (SIMON, 1999). Por exemplo, o microprocessador utilizado pela impressora deve ser mais poderoso do que os utilizados no leitor de código de barras e no nodo sensor. Ele deve ser capaz de controlar diversos outros componentes e também deve possuir um alto poder de processamento, visto que algumas tarefas podem exigir bastante de sua capacidade. De fato, a maioria das impressoras laser possui vários microprocessadores embarcados nelas. Os microprocessadores dos outros exemplos podem ter um poder menor de processamento, além de terem outras preocupações, como baixo consumo de energia. Um sistema embarcado precisa também de memória por duas razões: armazenar seu programa, seus componentes de software, e armazenar seus dados. Dessa forma, os sistemas embarcados descritos possuem também Read-Only Memory (ROM) e Random Access Memory (RAM).

Além de um processador e da memória, os componentes de cada sistema embarcado variam bastante. Muitos sistemas embarcados possuem também um rádio. No leitor de código de barras ele é usado para transferir o código lido para uma máquina registradora. No caso do nodo sensor, o rádio desempenha um papel muito importante pois é através dele que o nodo sensor integra a rede de sensores. É através desse rádio que as informações medidas pelos sensores do nodo são enviadas para um computador central.

Na seção anterior descreveu-se a necessidade do leitor de código de barras e do nodo sensor de possuírem uma bateria. Características que devem ser consideradas na escolha de uma bateria é tempo de uso e peso. No caso do leitor de código de barras, a bateria precisa durar pelo menos 8 horas, e deve ser leve. No caso do nodo sensor, por outro lado, a bateria deve durar um ano, mas pode não ter requisitos de peso. Nos dois casos, porém, só é possível manter um processador, memórias, um rádio, entre outros periféricos executando pelo tempo

desejado com a utilização de uma grande bateria. Desse problema surge uma das maiores preocupações em sistemas embarcados alimentados por bateria: como consumir pouca energia para que a bateria dure o tempo desejado (WOLF, 2003). Uma solução simples, mas que para muitos casos é insuficiente, consiste em se desligar um componente sempre que ele não estiver sendo usado pelo sistema, e religá-lo quando for necessário.

Além desses componentes, o nodo sensor também possui seus sensores, que podem medir as condições desejadas do ambiente, e também pode possuir um atuador, como um cooler externo, que é ativado quando alguma condição medida, como temperatura, atinge um nível crítico. No caso do leitor de código de barras, ele precisa de um botão para que o usuário ative o laser leitor. Já a impressora possui vários botões para o usuário controlar a impressão e também possui um visor para mostrar mensagens ao usuário.

Junto com componentes mecânicos e de hardware, há os componentes de software, armazenados na memória ROM dos dispositivos e executados pelos processadores quando colocados em funcionamento. O software do sistema pode ser dividido em duas partes: a aplicação e as rotinas de sistema que dão suporte à aplicação. Como a aplicação depende diretamente dos requisitos funcionais, o software dos três sistemas descritos varia tanto quanto os requisitos funcionais desses sistemas.

A parte referente à aplicação consiste em uma ou mais rotinas de software para cada requisito funcional do sistema. No exemplo do leitor de código de barras, duas rotinas estão presentes: uma que é executada quando o usuário aperta o botão de leitura, e outra que envia para a máquina registradora o código lido. No caso do nodo sensor, três rotinas compõem a aplicação: uma rotina monitora a condição do ambiente desejada, uma rotina ativa a rotina de recuperação do sistema quando a condição medida chega a um estado crítico e a rotina de recuperação ativa o atuador para recuperar o sistema após o estado crítico ser alcançado. Já o exemplo da impressora laser contém diversas rotinas, devido a sua maior complexidade.

A outra parte do software do sistema, composta pelas rotinas de sistema que dão suporte à aplicação, é na verdade a parte mais complexa do software embarcado. As rotinas de sistema podem variar muito em complexidade, dependendo da necessidade. A mais simples solução é a utilização de um loop infinito que executa todas as rotinas de aplicação do sistema sequencialmente ininterruptamente. Por outro lado, a solução mais complexa é a utilização de um sistema operacional

de tempo real. A utilização de apenas um loop infinito é interessante quando existem poucos requisitos funcionais. São essas rotinas de sistema, então, que controlam o cumprimento de todos esses requisitos. Na hora de escolher que tipo de rotinas usar, recomenda-se escolher as mais simples possíveis que sejam capazes de cumprir todos os requisitos do sistema (SIMON, 1999).

No caso do leitor de código de barras, o software é simples. As rotinas de aplicação podem permanecer num loop infinito. Não há a necessidade então da utilização de um sistema operacional. Quando o botão de leitura é apertado, as rotinas são liberadas para execução, o código é lido e então enviado para a máquina registradora. Para reduzir bastante o consumo de energia do sistema, ao invés de bloquear o loop infinito por software, o sistema poderia desligar todos os componentes de hardware, deixando de consumir energia. O sistema inteiro seria religado quando o botão do leitor fosse apertado pelo usuário, executando então as rotinas de aplicação novamente. Isso exigiria apenas que os projetistas de hardware permitissem que os componentes de hardware fossem desligados através de software e religados pelo usuário através de um botão (SIMON, 1999).

Os outros exemplos não são tão simples. O nodo sensor não pode ser totalmente desligado para consumir menos energia. Como uma alternativa a isso, ele pode regular os estados dos seus componentes de hardware durante a execução. Por exemplo, enquanto o rádio não estiver recebendo ou enviando dados, ele pode permanecer num estado de escuta onde consome menos energia. Além disso, algumas rotinas do nodo sensor possuem prioridades e períodos de execução diferentes das outras. Com a utilização de um sistema operacional, quando uma rotina de baixa prioridade estiver executando e uma rotina de alta prioridade solicitar execução, a rotina de baixa prioridade será interrompida e o sistema iniciará a execução da rotina de maior prioridade. Esse mecanismo seria inviável de se implementar com um único loop infinito. No caso da impressora laser, sua grande variedade de rotinas também exige a utilização de rotinas de sistema mais complexas, fornecidas por um sistema operacional.

O próprio sistema operacional usado, por sua vez, também é dividido em componentes de software, permitindo uma maior integração. Um componente que sistemas operacionais possuem é um escalonador. Se for um escalonador preemptivo, ele será o responsável por decidir qual rotina executará em qual momento, interrompendo a execução de qualquer rotina em prol de outra sempre que necessário. Como cada rotina possui seu próprio contexto, com seus dados e seu código, é inte-

ressante implementar uma abstração de *threads* ou processos e modelar cada rotina como uma thread. Nesse caso, a implementação de threads do sistema seria mais um componente. Semáforos e *signals* são componentes de software que podem ser usados para enviar sinais entre threads, sincronizar suas execuções e controlar apropriadamente o uso de dados compartilhados. No nodo sensor, por exemplo, a thread de monitoração pode utilizar um semáforo para ativar a rotina de recuperação do sistema quando a condição que ela estava monitorando chega a um estado crítico. Além desses componentes descritos, vários outros componentes de software podem estar presentes em sistemas operacionais para sistemas embarcados. Quanto maior o número, mais situações diversas o sistema consegue tratar, porém, mais código e dados ele usa, de forma que a memória precise armazenar mais dados e o processador precise executar mais instruções (MARWEDEL, 2003).

2.3 PROJETANDO UM SISTEMA EMBARCADO

Tendo sido apresentados alguns conceitos a respeito de sistemas embarcados, será discutido nesta seção como se realiza o projeto de um novo sistema embarcado e suas peculiaridades, visto que o sistema deve atender a requisitos bem específicos. Este trabalho terá como foco o projeto de sistemas embarcados dirigido à aplicação final, levando em consideração os requisitos funcionais e não funcionais definidos pelos projetistas, como consumo de energia, desempenho, custo, entre outros. O cumprimento desses requisitos deve ser planejado e executado através de um projeto integrado de hardware e software (MARWEDEL, 2003).

A Figura (1) mostra uma visão geral e básica dos passos realizados no projeto de um sistema embarcado. A figura mostra as principais etapas, mas não especifica as subetapas inerentes a cada uma. São listadas apenas as principais subetapas em relação ao escopo deste trabalho, omitindo assim etapas como verificação e validação, por exemplo.

Os projetistas inicialmente criam uma especificação inicial do sistema a ser projetado, expondo nela suas ideias. Essa especificação diz todas as funcionalidades que os projetistas desejam que o sistema possua, ou seja, todos os seus requisitos funcionais. Se o sistema não possuirá fios, a especificação deve informar isso, e deve especificar também o uso de uma bateria como fonte de alimentação. Se o sistema se comunica com outros sistemas, a natureza dessa comunicação já deve ser estabelecida nessa especificação.

Também já podem ser considerados na especificação os requisi-

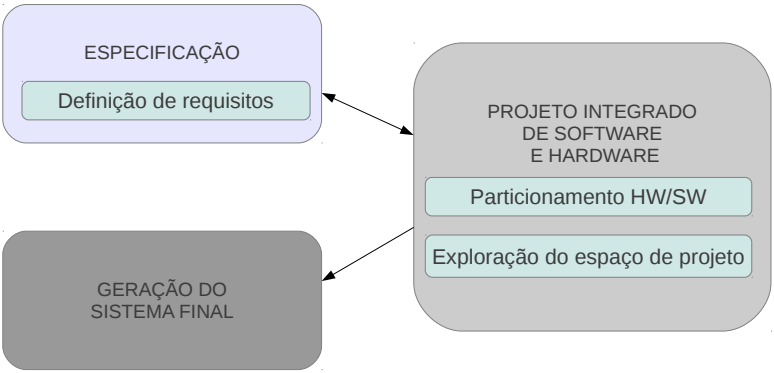

Figura 1: Visão geral das fases do projeto de um sistema embarcado.

tos não funcionais do sistema. Se o sistema irá possuir uma bateria, já pode-se informar quanto tempo a bateria deverá durar sem que uma recarga ou substituição seja feita. E um requisito não funcional do sistema deve ser então consumir pouca energia a ponto dessa bateria conseguir cumprir seu período de funcionamento. No caso do nodo sensor, dependendo da emergência do estado crítico, a thread de recuperação do sistema pode ser obrigada a executar seu código num curto período de tempo, fazendo com que o sistema possua também requisitos de desempenho ou tempo real. Além disso, pode ser que os projetistas queiram colocar o nodo sensor funcionando num ambiente hostil, com pouco espaço disponível. Dessa forma, a especificação do sistema deve contemplar requisitos de tamanho do dispositivo. O leitor de código de barras apresentado anteriormente também possui um requisito de peso, visto que o usuário irá segurá-lo durante várias horas. A lista de requisitos funcionais e não funcionais de um sistema embarcado pode crescer muito mais, e esta etapa do projeto será o foco das próximas seções do texto (SIMON, 1999).

Após a fase de especificação do sistema, inicia-se o mapeamento dessa especificação em cenários de implementação. Todas as funcionalidades especificadas são mapeadas em tarefas do sistema, que podem ser implementadas em hardware, software ou até mesmo com componentes mecânicos. Ao realizar esse mapeamento, porém, os projetistas se deparam com inúmeras possibilidades. Num cenário, uma tarefa pode ser implementada em hardware num componente dedicado, e em outro cenário a mesma tarefa pode ser implementada como threads sendo

executadas por uma ou mais unidades de processamento. Um conjunto de soluções possíveis para a especificação inicial é gerado ao final dessa etapa, e então é realizada uma exploração dessas soluções, em busca de uma que os projetistas definam como a mais apropriada para o projeto. Essa exploração é chamada de Design Space Exploration (DSE) (ZITZLER; THIELE, 1999).

O objetivo dessa DSE é identificar o cenário de mapeamento de componentes e configurações que melhor realiza as funcionalidades do sistema ao mesmo tempo que cumpre de forma mais satisfatória todos os requisitos não funcionais da especificação. Porém, pode ocorrer de nenhum cenário explorado ser capaz de realizar todas essas funcionalidades e cumprir todos os requisitos especificados. É possível então que se retorne à etapa inicial de especificação do sistema para que as especificações sejam revistas. Após a fase de DSE ser finalizada, a geração do sistema final é realizada (MARWEDEL, 2003).

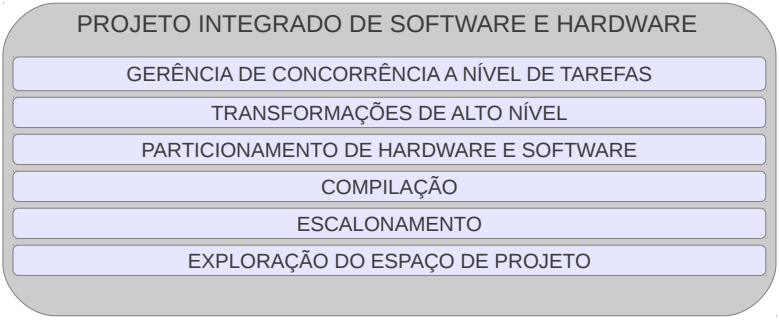

Figura 2: Visão mais específica da fase do projeto integrado de software e hardware (MARWEDEL, 2003).

A Figura (2) mostra diversas atividades realizadas na fase do projeto integrado de software e hardware. Realiza-se nessa fase, por exemplo, transformações de alto nível, que consistem em se realizar na especificação algumas otimizações que os compiladores não são capazes de executar. Tendo uma especificação pronta, a gerência de concorrência a nível de tarefa serve para determinar exatamente quais tarefas executarão no sistema final, o que pode variar em relação às tarefas descritas na especificação. Uma das principais partes desse projeto integrado é o particionamento de hardware e software, que é responsável por definir que funcionalidades do sistema serão executadas em hard-

ware, e em qual tipo de hardware, e quais serão executadas em software. Esse mapeamento determina que componentes poderão ser usados no sistema. Como foi visto anteriormente, diversos mapeamentos podem ser possíveis, criando várias soluções. A etapa então de exploração do espaço de projeto consiste em se analisar o conjunto de soluções possíveis e escolher uma. Em seguida, os componentes de software escolhidos são compilados, e depois é realizado o escalonamento das tarefas, onde é determinado em que momento cada tarefa começará a executar (MARWEDEL, 2003).

Uma forma de se reduzir bastante o tempo de desenvolvimento de um sistema embarcado é utilizar uma metodologia de desenvolvimento baseada em componentes, justificando sua utilização nesta dissertação. Através da utilização de componentes, reforça-se o conceito de reusabilidade. Novos projetos podem utilizar componentes já projetados, implementados e validados. O projeto do sistema é guiado então através da composição de componentes reutilizáveis, a fim de melhorar o processo de desenvolvimento. Cada componente é independente dos outros e, através de uma interface amigável, eles podem ser acoplados para a geração de sistemas maiores. Essa interface amigável é possível através da criação de uma infraestrutura ágil que defina regras para a composição de componentes. Essas regras estabelecem como será a comunicação entre os componentes, de forma que eles possam colaborar entre si na execução das funcionalidades especificadas e de forma que eles possam ser agregados de forma fácil e com o mínimo de modificações necessárias entre eles (JACOBSON, 1992).

Devido a sua grande utilidade para sistemas embarcados, a pesquisa a respeito do desenvolvimento de componentes reutilizáveis tem sido objeto de estudo da comunidade científica. Em estudos dessa natureza, avanços foram atingidos no contexto do hardware com o desenvolvimento e utilização de Intellectual Property (IP) (GAJSKI, 1999), o que permitiu uma grande redução na complexidade do projeto desses sistemas. Após a implementação das funcionalidades através da metodologia de desenvolvimento baseada em componentes, obtém-se um conjunto de componentes de software e também um projeto de hardware que combinados formam o sistema final. Na etapa seguinte, de implantação, o sistema final é montado e disponibilizado para utilização.

2.4 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Nesta seção e nas seguintes o texto terá como foco a especificação dos requisitos funcionas e não funcionais do sistema e também requisitos inerentes ao projeto, como por exemplo a experiência dos projetistas em relação aos diferentes componentes do sistema. Essa especificação será objeto de estudo ao longo da dissertação e possui relação direta com os modelos matemáticos propostos. Como foi visto anteriormente, sistemas embarcados são usados para as mais diversas finalidades, e são esses requisitos funcionais que motivam a grande variedade de requisitos não funcionais. Além disso, outras restrições no projeto são impostas por fatores externos, como mercado e o ambiente onde o sistema será instalado.

A especificação dos requisitos de um sistema embarcado geralmente é realizada sem formalidades, usualmente em linguagem natural, gerando um resultado às vezes confuso (SIMON, 1999). Porém, independente da forma com que esses requisitos são especificados, após os projetistas definirem em detalhes todas as funcionalidades do sistema, a lista de requisitos funcionais está completa. Essa lista não deve apenas conter uma descrição simples do que o sistema fará, ela também deve informar como o sistema realizará as funcionalidades. No caso do nodo sensor, usado como exemplo anteriormente, detalhes a respeito de comunicação e segurança devem ser especificados junto com as funcionalidades. Projetistas podem querer que seus nodos não apenas se comuniquem com outros nodos, mas que essa comunicação seja feita através de uma infraestrutura confiável e segura (FRÖHLICH; STEINER; RUFINO, 2011), e isso afeta diretamente a implementação do sistema. Ao fim da etapa de definição dos requisitos funcionais, todos os componentes que serão necessários para realizar tudo o que os projetistas desejam já foram definidos. A próxima etapa é definir os requisitos não funcionais, que serão essenciais durante a fase de DSE.

É comum que sistemas embarcados sejam instalados em locais sem acesso a energia elétrica, e também que eles funcionem ininterruptamente por longos períodos de tempo. Por essa razão, possuir um baixo consumo de energia é um dos requisitos não funcionais mais comuns no projeto de um sistema embarcado. Todos os componentes mecânicos e de hardware, controlados pelos componentes de software, consomem energia. Um rádio, por exemplo, possui um consumo de energia diferente para cada estado de funcionamento. Naturalmente, ele consome mais energia quando está transmitindo ou recebendo dados, e menos energia quando está apenas aguardando a chegada de dados.

Através de tecnologias modernas como Dynamic Voltage Scaling (DVS), é possível reduzir a voltagem dos microprocessadores em momentos em que eles não estão sendo tão exigidos, reduzindo o consumo de energia (CHEDE; KULAT, 2008).

Sistemas embarcados também geralmente possuem requisitos de desempenho. Dos exemplos vistos anteriormente, o leitor de código de barras é talvez o que menos necessite de alto desempenho. O tempo entre uma leitura de código de barras e a próxima não precisa ser muito curto, pois depende da velocidade do usuário. A impressora laser, entretanto, pode não possuir muito tempo para realizar algumas funcionalidades, uma vez que seus usuários possuem cada vez menos tolerância para esperar uma página ser impressa, por exemplo. Outro exemplo comum são aplicações de multimídia, que precisam mostrar vídeos em telas numa velocidade de processamento de dezenas de quadros por segundo. Usuários também possuem pouca tolerância a atrasos nesse processamento, pois ele pode gerar a perda de quadros, reduzindo a qualidade do vídeo.

Em todos esses sistemas, porém, tolera-se que raros e eventuais atrasos ocorram no processamento, pois os usuários não irão perceber muita diferença se uma página demorar 1 décimo de segundo a mais para ser impressa, ou se uma quantidade pequena de quadros for perdida. Existe, porém, uma variação do requisito de desempenho que se aplica a tarefas de tempo real críticas, onde o requisito é expresso em termos do tempo máximo de execução das funcionalidades especificadas. Nessas tarefas, os requisitos de desempenho são rigorosos, e executar uma funcionalidade num tempo maior do que o máximo especificado é considerado um erro do sistema.

Redes de sensores sem fio podem possuir restrições de tempo real nas suas aplicações. No exemplo do nodo sensor, ele e os nodos adjacentes podem estar monitorando a saúde de pacientes e cidadãos idosos. Na eventual ocorrência de uma condição crítica ser atingida, o sistema deve alertar o pessoal encarregado para que medidas sejam tomadas. Esses nodos seriam inúteis se médicos não fossem alertados cedo o bastante para salvar a vida dos pacientes. Outra aplicação possível é o monitoramento de locais de acesso proibido. Se algum movimento de pessoas for detectado no local, uma ação, como ligar uma câmera, deve ser executada rapidamente, antes que algum intruso saia da área onde o movimento foi detectado. Nesses casos, é inaceitável que o sistema demore mais do que o permitido para executar as tarefas (PRABH, 2007).

Por ser um recurso caro, pouco uso de memória também costuma

ser um requisito não funcional em sistemas embarcados, principalmente em sistemas onde muitas tarefas são implementadas usando componentes de software. Isso motivou muita pesquisa científica nas últimas décadas, a ponto de hoje sistemas operacionais inteiros poderem ocupar menos de 32 KB de memória (DUNKELS; GRONVALL; VOIGT, 2004; BHATTI et al., 2005). Aplicações complexas, porém, costumam possuir muito código, e ao mesmo tempo também precisam utilizar mais serviços fornecidos pelo sistema operacional, e, sem o devido cuidado, o sistema final facilmente ocupará mais espaço do que os projetistas desejavam. Nesses casos, os projetistas irão preferir sistemas operacionais que executem suas rotinas utilizando menos memória. Técnicas avançadas de engenharia de software podem ser utilizadas no desenvolvimento do software de forma a diminuir o uso de memória.

Área, peso e volume também podem ser definidos na especificação do sistema como requisitos não funcionais, principalmente em cenários onde se usam muitos componentes de hardware. No exemplo do leitor de código de barras sem fio, peso e volume são requisitos importantes, uma vez que o usuário irá segurar com uma mão o aparelho e usá-lo por horas. As possíveis aplicações para redes de sensores sem fio são tantas que é comum nessa área de pesquisa que os nodos sensores possuam restrições de área e peso também, já que são muitas vezes instalados em locais hostis. Um outro exemplo de sistema embarcado onde requisitos desse tipo se fazem necessários é no projeto de besouros ciborgues. Besouros ciborgues são besouros que possuem um sistema embarcado acoplado a eles. Esse sistema é capaz de estimular com pulsos elétricos áreas relativamente grandes do circuito neuromuscular desses insetos, a fim de controlar para onde e como os besouros voam (MAHARBIZ; SATO, 2010). Naturalmente, o sistema precisa ser pequeno o bastante para poder ser acoplado ao inseto e também leve o bastante para o besouro conseguir carregá-lo.

Esses requisitos não funcionais descritos, além de outros, compõem um grupo de requisitos que devem ser cumpridos durante o projeto do sistema. Porém, existe mais um requisito que engloba todos esses e que está sempre presente no desenvolvimento de um sistema embarcado: o requisito de custo financeiro do sistema. É natural que projetistas tenham como objetivo alcançar um sistema final que custe o menos possível, e isso se reflete em todos os outros requisitos não funcionais do sistema. Esse custo financeiro passa a ser requisito a partir do momento em que os projetistas traçam como meta inserir o sistema num mercado competitivo, onde o preço para o usuário final já está preestabelecido pelos fabricantes competidores. Esse custo financeiro pode

ser dividido em dois custos diferentes: o custo inicial de projetar o sistema, que é pago apenas uma vez, e o custo de fabricar cada cópia do sistema. Se o preço para o usuário final ultrapassar o limite imposto na especificação do sistema, o produto irá fracassar no mercado (PHAM; ZHANG, 1999).

Junto com esse custo, outra característica importante quando se trata de um mercado cada vez mais competitivo é o conceito de Time To Market (TTM), que é o tempo que um produto leva para chegar ao mercado. Para atingir sucesso no mercado, sistemas embarcados não precisam apenas custar cada vez menos, eles precisam também ser disponibilizados para o mercado em cada vez menos tempo, ou seja, devem cumprir metas de TTM cada vez mais rígidas (DEBARDELABEN; MADISETTI; GADIENT, 1997; BUTT; SAYYAH; LAVAGNO, 2011).

Alguns outros requisitos podem ser definidos pelos projetistas, embora a comunidade científica foque sua pesquisa apenas nos requisitos descritos acima. Porém, é muito importante modelar a maior quantidade possível de requisitos no projeto de um sistema embarcado. A modelagem proposta que será apresentada nos capítulos seguintes suporta a definição de variados requisitos no projeto. De fato, a modelagem é extensível, e isso se apresenta como uma das vantagens da abordagem que será apresentada.

Requisitos mais subjetivos também podem ser definidos na especificação de um sistema. Como é possível comprar componentes de terceiros ao invés de usar componentes criados e validados localmente, há vários requisitos que exploram esses detalhes. Por exemplo, projetistas podem querer que o sistema use o máximo possível de componentes que eles já conheçam e já tenham usado em projetos anteriores. Dessa forma, a experiência da equipe de desenvolvimento é um fator relevante na escolha dos componentes, assim como o grau de reusabilidade de cada componente. Uma equipe que já conhece em detalhes os componentes que serão acoplados pode acelerar o processo de desenvolvimento e reduzir o número de erros do sistema, aumentando assim sua confiabilidade.

A disponibilidade dos componentes no mercado também pode ser modelada como um requisito não funcional. Os projetistas podem preferir componentes que estejam disponíveis com mais facilidade no mercado, não só por questões de custo mas também porque comprar um componente localizado a uma grande distância exige que os projetistas esperem alguns dias ou semanas para o componente chegar, possivelmente atrasando o desenvolvimento do projeto. Por fim, o estágio de desenvolvimento dos componentes é outro fator relevante. Alguns com-

ponentes podem se encontrar em estágio de implementação, e outros já podem ter sido testados e validados há anos. Esses detalhes podem influenciar a escolha de componentes para o sistema, junto com os outros requisitos descritos.

2.5 CONFLITO ENTRE REQUISITOS E TOMADA DE DECISÕES

Com todos os requisitos apresentados na seção anterior, é natural imaginar que diversos conflitos aparecerão durante a DSE. Vários dos requisitos que foram especificados são conflitantes. Essa seção pretende apresentar os conflitos mais comuns que a fase de DSE precisa resolver. Tendo esses conflitos bem descritos, na próxima seção será mostrado como a comunidade científica vem modelando esses requisitos conflitantes e quais desafios têm encontrado.

O principal conflito que aparece após os requisitos terem sido definidos é em relação ao custo financeiro do sistema. Esse custo acaba colidindo com todos os outros requisitos, pois um componente que possui alto desempenho e consome pouca energia, se existe, provavelmente não será barato. Da mesma forma, muitos dos problemas de custo financeiro seriam resolvidos se escolhêssemos os componentes mais baratos, mas esses por sua vez provavelmente não cumprem os outros requisitos. O requisito de uso de memória possui uma relação direta com esse custo: quanto mais memória o sistema tiver, mais caro ele será.

Os usuários também estão ficando cada vez mais exigentes. Eles querem sistemas cada vez mais sofisticados, sem que exista aumento de preços. E isso se reflete no mercado. Como sistemas embarcados são cada vez mais usados, eles já são parte importante da economia mundial, e o mercado está repleto de companhias projetando seus próprios sistemas embarcados e tentando se inserir no mercado. O custo financeiro total de um sistema é então um requisito importante, mas que entra em conflito com basicamente todos os outros. Durante a DSE os projetistas precisam balancear os requisitos, permitindo que o sistema custe um pouco mais, ou então que ele consuma um pouco mais de energia e seja um pouco mais lento, caso não exista uma solução ideal para o sistema.

Pela sua importância e por afetar tanto componentes de hardware como de software, o consumo de energia também é fonte de vários conflitos. Fazer um sistema consumir pouca energia é possível utilizando-se componentes de hardware de baixo desempenho. Um mi-

croprocessador de baixa frequência pode consumir menos energia, mas o código executado também será mais lento. No caso do leitor de código de barras, isso não é um problema, pois os algoritmos do sistema são simples. Por outro lado, sistemas de tempo real não podem aceitar perdas de desempenho; é nesses sistemas que o consumo de energia colide mais fortemente com desempenho. Os avanços em monitoramento do consumo de energia e DVS têm colaborado para que os conflitos entre esses requisitos diminuam (CHEDE; KULAT, 2008; JUNIOR; FRÖHLICH, 2011), mas ainda é um problema para a DSE. Para sistemas baseados em bateria, o preço da bateria aumenta conforme aumenta também a capacidade dela. Dessa forma, se os projetistas permitirem que o sistema consuma um pouco mais de energia, pode ser que eles precisem também utilizar uma bateria maior e mais cara.

O desempenho em sistemas embarcados está diretamente relacionado com o microprocessador e os barramentos de comunicação. Um microprocessador mais potente alcançará um melhor desempenho, mas será mais caro e consumirá mais energia. Isso se acentua em sistemas embarcados de tempo real alimentados por baterias. Além desse conflito, o desempenho também é influenciado pela quantidade de memória do sistema. Técnicas de programação que permitem que o sistema cumpra seus requisitos de tempo de execução podem fazer com que mais código seja usado para o sistema, e consequentemente o uso de memória aumente (LEE et al., 2008).

Além disso, em alguns sistemas, companhias usam o desempenho do sistema como estratégia de marketing para vender mais produtos, como é o caso da impressora laser que foi usada como exemplo nesse capítulo. Empresas utilizam métricas como quantidade de páginas impressas por minuto para comparar seus produtos com o de concorrentes. Dessa forma, impressoras precisam cumprir requisitos de tempo cada vez mais rigorosos, mesmo que o usuário pudesse esperar um ou dois segundos a mais para que todos os cálculos complexos de impressão fossem realizados e a impressão fosse feita usando componentes mais baratos.

Requisitos como disponibilidade no mercado, experiência dos projetistas e estágio de desenvolvimento dos componentes também podem influenciar nos outros requisitos e gerar conflitos. Os projetistas podem ter grande experiência num componente que está pouco disponível no mercado, ou que tenha sido implementado há pouco tempo e não tenha passado por muitas etapas de testes. Da mesma forma, um componente que está disponível com relativa facilidade no mercado pode ser totalmente desconhecido por parte dos projetistas ou estar

ainda em fase de implementação. É possível que os projetistas não conheçam nada a respeito do componente que consuma menos energia e tenha o melhor desempenho, ou que ele esteja pouco disponível no mercado: vendido apenas no mercado asiático, por exemplo.

Esses requisitos também influenciam o TTM do sistema. Projetistas com experiência nos componentes escolhidos implementam o sistema em menos tempo. Componentes com maior disponibilidade no mercado ficam disponíveis mais cedo para serem usados na implementação do sistema. E componentes testados e validados permitem que um período menor de tempo seja usado para testes do sistema final. A união de todos esses casos permite uma grande redução no TTM do sistema.

2.6 MODELANDO REQUISITOS

Todos os requisitos discutidos nas seções anteriores são reais. Uns são mais comuns do que outros, mas todos podem estar presentes no projeto de um sistema embarcado. E com todos os conflitos gerados por esses requisitos, a etapa de DSE torna-se inviável sem a utilização de ferramental apropriado. Ao mesmo tempo, é uma das etapas mais importantes do projeto. Durante a DSE, a especificação completa do sistema é explorada e as melhores soluções para a implementação são identificadas, porém não há consenso a respeito da forma como essa especificação deve ser modelada para o projeto integrado de software e hardware e principalmente para a DSE. Além disso, grande parte da comunidade científica modela as especificações com foco em no máximo dois requisitos não funcionais como os que foram descritos anteriormente, o que torna os modelos muito simplistas e diminui o potencial da DSE. Nesta seção serão apresentadas algumas modelagens existentes e suas vantagens e limitações serão discutidas.

Nogueira e outros propõem uma estratégia de modelagem de eventos baseada em Coloured Petri Nets (CPN), com o objetivo de avaliar o consumo de energia e o desempenho de sistemas embarcados. Eles modelam esses requisitos usando um modelo a nível de arquitetura, ou seja, a nível de circuito, Register-Transfer Level (RTL) e nível de microestrutura. Durante a modelagem foram criados diversos blocos de modelos em CPN, seguindo uma abordagem crescente para a representação de comportamentos sofisticados. Através de uma simulação das CPN as estimativas de desempenho e energia são então obtidas. A abordagem proposta, porém, possui diversas limitações. Os autores

tentam avaliar durante a fase de DSE apenas aspectos de desempenho e consumo de energia em sistemas alimentados por bateria. A abordagem também possui um espaço de projeto limitado, visto que a escolha da plataforma de hardware foi feita a priori pelos projetistas, não fazendo parte da exploração (NOGUEIRA et al., 2011).

Em uma outra abordagem semelhante usou-se o conceito de Stochastic Automata Network (SAN) para realizar a modelagem (ZAMORA; HU; MARCULESCU, 2007). Diversas SAN são criadas para diferentes partes do sistema, como a Central Processing Unit (CPU), caches, memória, rotinas de escalonamento e barramentos. Com base nessas redes, um mapeamento de arquitetura e aplicação é realizado e uma ferramenta chamada Metropolis realiza um balanceamento entre desempenho e consumo de energia em variadas aplicações embarcadas através de simulações. Embora seja uma abordagem também limitada a desempenho e consumo de energia, todos os modelos são independentes de arquitetura. Foram criados modelos separados para aplicação e arquitetura, possibilitando o reuso de ambos.

O conjunto de ferramentas Artemis foi proposto para realizar a modelagem e simulação de um sistema embarcado através de sua especificação e também realizar DSE, porém a abordagem explora apenas desempenho durante a etapa de DSE (PIMENTEL, 2008). São várias ferramentas que permitem uma avaliação de desempenho e exploração do sistema eficientes. A abordagem também é independente de arquitetura e modela aplicação e arquitetura de forma separada, embora seu maior foco sejam sistemas embarcados multimídia.

Fummi e outros propõem um framework de cossimulação que também explora apenas requisitos de desempenho, embora possua uma abordagem diferente (FUMMI et al., 2009). O framework de cossimulação é usado para verificar o projeto integrado do sistema embarcado e avaliar seu desempenho. Para a utilização no framework, a especificação do sistema é modelada em SystemC.

Qiu e Sha propõem uma abordagem para sistemas de tempo real. Para saber quais tarefas atribuir a cada unidade funcional é preciso saber os tempos de execução das tarefas. Como o tempo de execução de algumas tarefas não é fixo, os autores modelam cada tempo de execução variado como uma variável probabilística aleatória e em seguida resolvem o problema de Heterogeneous Assignment with Probability (HAP). A solução do problema é um mapeamento completo de todas as tarefas em unidades funcionais. Experimentos mostraram que os algoritmos foram capazes de alcançar uma significante redução no consumo de energia e forneceram mais soluções de projeto que minimizem o custo

total do sistema, ao mesmo tempo que cumpre os requisitos de tempo. A abordagem funciona com sistemas de tempo real críticos e não críticos (QIU; SHA, 2009).

Uma outra abordagem propõe a utilização de algoritmos modelados e validados com Simulink e ferramentas comerciais para realizar uma DSE (BUTT; SAYYAH; LAVAGNO, 2011). Essa abordagem tem como objetivo explorar requisitos de consumo de energia, desempenho e custo de hardware, com foco em redes de sensores sem fio. De todas as modelagens vistas nesta seção, é a que leva em consideração mais requisitos, embora explore apenas componentes de hardware e com foco em redes de sensores.

Ueda e outros apresentam uma outra abordagem que modela o sistema embarcado através do conceito de Execution Dependency Graph (EDG). Os autores exploram o desempenho do sistema através de uma análise do grafo. A vantagem é que a abordagem permite uma avaliação de desempenho mais rápida do que outros trabalhos relacionados, porém também está limitada a requisitos de desempenho (UEDA et al., 2004).

De modo geral, todas as abordagens apresentadas nessa seção possuem várias limitações. Os modelos funcionam bem para o que se propõem, mas dificilmente poderiam ser estendidos para considerar a gama de possibilidades que a abordagem apresentada nessa dissertação considera. Por exemplo, quase todas as abordagens acima são focadas num nicho específico de aplicações, como tempo real ou sistemas multimídia, o que prejudica o pleno reuso da modelagem. Todas as abordagens acima também tratam poucos requisitos não funcionais, e usam ferramentas e técnicas que nem sempre serão capazes de modelar outros tipos de requisitos, o que deixa os modelos pouco extensíveis.

A modelagem que esta dissertação propõe é baseada num repositório de componentes e um conjunto de funções de custo expressas matematicamente. Há várias outras abordagens existentes que seguem a mesma linha. Elas exploram o uso de modelos matemáticos no projeto de sistemas embarcados, e elas serão discutidas em detalhes no próximo capítulo, onde o restante do estado da arte será apresentado.

3 MODELANDO REQUISITOS MATEMATICAMENTE

Nesta dissertação propõe-se uma modelagem que engloba requisitos quantificáveis que o sistema possa ter e que pode ser utilizada por ferramentas de DSE sem grandes esforços. Neste capítulo serão apresentadas diversas outras abordagens semelhantes, e suas vantagens e desvantagens serão discutidas.

A utilização de equações matemáticas para descrever relações entre componentes e configurações de um sistema embarcado possibilita que os modelos sejam facilmente verificados, validados e utilizados. Mecanismos simples e metodologias de DSE conseguiriam usar as equações no processo sem muitas dificuldades. A maior dificuldade desses modelos, porém, é conseguir representar todos os requisitos de um sistema, mantendo a correção do modelo.

Os trabalhos na área costumam separar requisitos por equações, criando uma equação para cada um. Essas equações são então analisadas em conjunto para a identificação dos melhores componentes para o sistema. As equações que representam cada requisito devem conter o maior número possível de fatores que influenciam aquele requisito no sistema, e também devem levar em consideração fatores como dependências entre variáveis. Outra preocupação deve ser eventuais incertezas em relação a algumas variáveis das equações.

Nas próximas seções será demonstrado como é possível desmantelar um requisito em um conjunto de variáveis e como expressar essas variáveis numa equação matemática. Será mostrado então como descrever requisito por requisito, como é possível modelá-los matematicamente e como a comunidade científica tem feito nos últimos anos. Também será discutido como os pesquisadores costumam tratar incertezas e dependências entre variáveis.

3.1 EXPRESSANDO REQUISITOS MATEMATICAMENTE

Inicialmente, ao criar uma nova equação, define-se qual será a semântica do resultado da equação. Uma equação de custo financeiro pode ter como resultado o valor monetário que o sistema irá custar, assim como também pode possuir um valor relativo dentro de um intervalo de valores. Tendo isso definido, é importante estabelecer como a especificação do sistema será avaliada de forma que o resultado da equação nos diga o custo financeiro total do sistema, ou o consumo de

energia de sistema, ou quanto de memória o sistema usará, etc.

É importante que as equações possam, de alguma forma, ser integradas posteriormente para uma análise conjunta. Para isso, é importante que as equações abordem a especificação do sistema de forma consistente. Por exemplo, o sistema embarcado pode ser visto como um conjunto de componentes de hardware e software, ou como um conjunto de tarefas executando numa plataforma de hardware, ou como um conjunto de instruções sendo executadas por um microprocessador. Se alguma equação tratar o sistema como um conjunto de tarefas e outra tratar o sistema como um conjunto de componentes, a integração entre elas ficará prejudicada, reduzindo o potencial da exploração.

Independente da forma como as equações capturam aspectos do sistema, o sistema é visto como um conjunto de artefatos. Por exemplo, se analisarmos o sistema como um conjunto de componentes, um desses artefatos pode ser um rádio. Se analisarmos o sistema como um conjunto de tarefas, um artefato pode ser a tarefa de transmissão do código de barras lido para uma máquina registradora. Para cada artefato do sistema, existe um conjunto grande de características ou variáveis a respeito daquele artefato que descrevem seu funcionamento.

O foco desta dissertação é o projeto de sistemas embarcados baseado em componentes, logo iremos tratar o sistema sempre como um conjunto de componentes, ou seja, cada artefato é um componente. Um possível componente do sistema é o rádio. O rádio então possui uma série de características que descrevem seu funcionamento no sistema. O rádio possui diversos estados de funcionamento, e cada estado possui um *duty cycle*, que é quanto tempo o rádio permanece naquele estado em relação ao tempo total de funcionamento do rádio. Cada estado também possui o quanto de corrente elétrica ele consome quando está naquele estado. Um rádio também possui informações como taxa de transmissão máxima e outras informações referentes à tecnologia usada.

Um framework de componente define os relacionamentos entre diferentes componentes através de suas interfaces. A Figura (3) ilustra em alto nível o projeto de um sistema baseado em componentes. Com base em um repositório com diversos tipos de componentes e diversas implementações de cada tipo, os componentes são selecionados e acoplados em um framework já definido, onde as relações entre os componentes também já estão definidas, formando então o sistema final. Todo o projeto integrado de software e hardware fica no meio desse processo.

Cada um desses componentes possui várias características que o descrevem e o identificam. Essas características é que serão levadas em

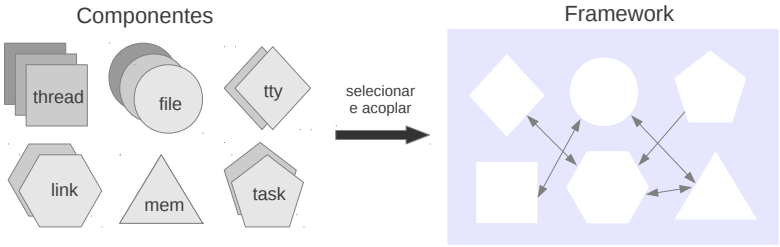

Figura 3: Um framework de componente (FRÖHLICH, 2001).

consideração na escolha do componente e também na etapa de DSE. A Figura (4) mostra como é a interface de um componente, com suas características de funcionamento. No caso, é mostrada a interface de uma implementação do componente de tipo thread. Uma thread que segue essa interface possui um ponto de entrada, um estado, uma prioridade de execução, e uma série de funções que descrevem seu funcionamento.

Para cada componente, um conjunto de características descreve seu funcionamento. Analisar o sistema embarcado em relação a algum requisito consiste então em percorrer o conjunto de componentes de acordo com a representação escolhida. Para cada componente analisado, dentre as suas características de funcionamento, coleta-se então as que influenciam no requisito em questão. A relação entre elas é modelada e representada equacionalmente, e o resultado dessa equação local é obtido. Sabendo o resultado da equação para aquele componente, unimos os resultados de todos os componentes e chegamos ao resultado da equação global, que pode ser o consumo de energia total do sistema, ou a quantidade de memória usada pelo sistema, ou o tempo de execução do sistema, ou o custo do sistema, etc.

3.2 MODELANDO CUSTOS

O custo, como métrica financeira, é um requisito genérico que pode ser aplicado a qualquer etapa do desenvolvimento de qualquer tipo de sistema computacional. Independente do que o sistema final fará, métricas de custo podem ser impostas ao projeto. Vários fatores podem influenciar custos do sistema, e isso permite que equações de custo possam adquirir formas muito diferentes, dependendo do objeto de análise delas. Nesta seção serão apresentados diferentes tipos de

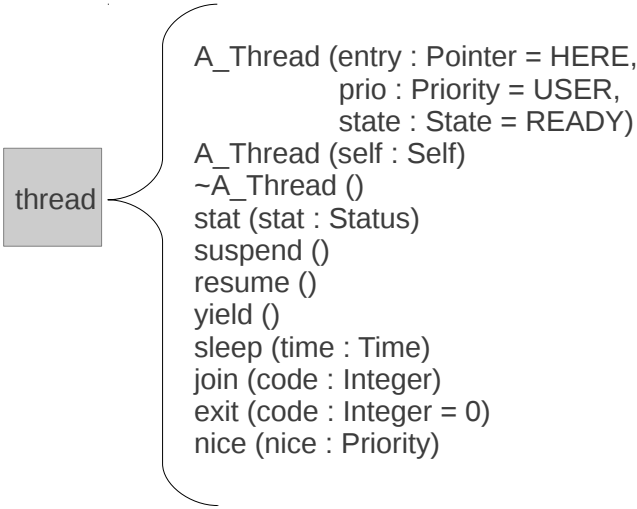

Figura 4: A interface de uma implementação do componente de tipo thread (FRÖHLICH, 2001).

equações de custo para o desenvolvimento de projetos.

Vários trabalhos propõem o uso de equações no desenvolvimento de sistemas computacionais para o cálculo do custo de desenvolvimento, esse custo geralmente tem como resultado um valor monetáro. Debardelaben e outros propõem a metodologia de projeto dirigido a custo para projetos de sistemas embarcados. Eles ampliam o conceito de custo e propõem equações para diferentes tipos de custos como esforço em homens-mês para desenvolver o software, tempo em meses para desenvolver o software e o esforço em homens-mês para fornecer manutenção do software. Como variáveis dessas equações, os autores usam a quantidade de linhas de código do software, requisitos de tempo de execução, quantidade de memória disponível, e também levam em consideração estimativas do efeito causado no custo por variáveis como equipe, ferramentas e atributos de projeto (DEBARDELABEN; MADISETTI; GADIENT, 1997). Dessa forma, características de hardware, de software e da equipe de desenvolvimento acabam todas afetando os custos do sistema.

Ragan e outros apresentam a ferramenta Ghost, para o desenvolvimento de hardware e software. Na abordagem deles o custo é

expresso através de uma única grande equação, e representa o custo, em dólares, de se desenvolver, fabricar e testar todo o sistema. A abordagem é semelhante à descrita acima, principalmente em termos dos custos de software. Os custos do hardware dependem de características como quantidade de portas lógicas, custo de projeto do chip, fabricação de wafer, geração de testes automáticos e manuais, etc (RAGAN; SANDBORN; STOAKS, 2002).

Vários trabalhos também propõem técnicas, metodologias e ferramentas para a estimativa de custos no desenvolvimento e manutenção de software. Boehm e outros utilizam a ferramenta COCOMO. A ferramenta foi criada na década de 80 e passa por eventuais atualizações. Os autores informam que cada vez mais técnicas e metodologias novas estão sendo usadas no projeto de software, o que faz com que ferramentas antigas percam eficiência (BOEHM et al., 1995). Diferentes técnicas vêm sendo usadas para melhorar o tratamento do custo de desenvolvimento de software e também deixá-lo mais flexível. Chulani e outros apresentam um avanço sobre o trabalho de Boehm no sentido de introduzir o uso de inferências Bayesianas para resolver alguns dos problemas enfrentados pela modelagem. Através de inferências Bayesianas é possível fazer inferências consistentes utilizando não apenas características do sistema sendo projetado, como também informações de especialistas a respeito de projetos passados (CHULANI; BOEHM; STEECE, 1999).

Pham e Zhang também propõem avanços nesse sentido com o tratamento de riscos e garantias nos modelos de custo. Embora o custo em si seja financeiro, os modelos podem ser usados para custos mais genéricos. Eles utilizam um modelo de confiabilidade baseado num processo de Poisson não homogêneo (PHAM; ZHANG, 1999). As equações desses trabalhos avaliam o custo não apenas para desenvolver todos os componentes de hardware e software, mas também o custo para manter uma equipe de desenvolvimento, o custo para realizar testes, manutenção do software, correção de possíveis falhas encontradas após a distribuição do software, etc.

Kujawski e outros propõem uma metodologia para a estimativa do custo de um projeto qualquer usando uma análise de custo probabilística que modela a estratégia de alocação de recursos. Os autores apresentam um modelo de análise de custo que leva em consideração riscos, incertezas e dependências entre variáveis. Como esses fatores costumam ser estimados pelos projetistas, os autores incorporam ao modelo as várias influências psicológicas que os projetistas podem sofrer durante a estimativa desses valores (KUJAWSKI; ALVARO; EDWARDS,

2004). Essa abordagem é interessante pois leva em consideração vários fatores que outras abordagens não consideram, o que aumenta a confiança dos projetistas nos resultados obtidos.

Algumas abordagens foram propostas com foco específico em sistemas embarcados. Axelsson propõe modelos de custo que tratem incertezas usando distribuições de probabilidade, e também propõe o uso de simulações para analisar os riscos de não se alcançar as métricas de custo (AXELSSON, 2006). O modelo proposto é bastante extenso e considera diversas variáveis nas suas equações. Em termos do custo financeiro do produto final, são usadas características como número esperado de unidades que serão fabricadas, custo total de cada cópia do sistema até a entrega, custo de produção de mais uma cópia do sistema, custo para desenvolver o sistema, esforço em homens-mês para desenvolver o software e o hardware e o custo de um homem-mês. Em termos de memória, por exemplo, usa-se informações como custo para se produzir uma unidade da memória, área do circuito eletrônico, quantidade de bytes de informação que a memória consegue armazenar, percentual da capacidade que não é usada e o custo de 1 byte de memória.

Persson, por sua vez, apresenta algumas equações semelhantes num estudo que identifica as fontes de utilização de recursos no projeto de sistemas embarcados e discute o impacto que várias decisões de projeto têm sobre o consumo de recursos. O autor então propõe a análise de envolvimento de dados para avaliar a eficiência da utilização de recursos na implementação do software (PERSSON, 2008).

Chen e outros também apresentam uma abordagem semelhante à de Axelsson, criando uma função de eficiência de custo para o projeto integrado de hardware e software. Os autores, porém, utilizam menos variáveis nos cálculos, considerando principalmente o custo dos componentes de hardware e o esforço em homens-mês para realizar a programação do software em linguagens como C e C++ (CHEN et al., 2011).

Por fim, algumas análises de custo têm sido feitas numa área bem específica: o projeto de circuitos integrados 3D. Dong e Xie propõem um estudo a respeito do método de estimativa usado nas primeiras etapas do projeto de circuitos integrados 3D. Através desse estudo, eles propõem a utilização de um modelo de análise de custo para identificar os efeitos causados pela utilização de circuitos 3D nos custos do projeto (DONG; XIE, 2009). A utilização desses circuitos traz vantagens, mas costuma aumentar os custos do projeto, uma vez que novas etapas são inseridas no desenvolvimento do projeto e novos componentes são usados para o acoplamento de circuitos, o que justifica o modelo proposto.

Weerasekera e outros estendem o trabalho de Ragan no cálculo dos custos de hardware, adicionando as variáveis referentes ao projeto 3D do circuito. Junto com custo financeiro, os autores também modelam desempenho (WEERASEKERA et al., 2009).

A maioria das abordagens vistas nessa seção tratam o custo como um requisito financeiro, ou seja, o resultado da equação é um valor monetário que será gasto pelos projetistas no desenvolvimento do sistema. Porém, a maioria dos modelos pode ser estendida para custos mais genéricos. Além disso, as abordagens descritas acima costumam considerar custos de desenvolvimento de componentes, partindo do princípio que os componentes não existem e precisam ser criados. Embora nesta dissertação as equações não sejam muito diferentes, a abordagem usada aqui considera que os componentes já existem, simplificando a questão do custo. Nesse caso, os projetistas compram os componentes para criar o sistema, ou utilizam componentes já disponíveis, como será visto mais à frente em detalhes.

3.3 MODELANDO CONSUMO DE ENERGIA

Até agora foram apresentados nesse capítulo trabalhos relacionados que utilizam equações de custo. Agora serão apresentados diversos trabalhos relacionados que mostram como a comunidade científica vem modelando o consumo de energia de sistemas. Essa modelagem é usada quando o consumo de energia é uma preocupação dos projetistas. Há formas variadas de se calcular o consumo de energia de um sistema, e serão mostradas algumas abordagens que diferem bastante e também outras que foram usadas como base para a que será proposta nesta dissertação.

Lajolo e outros fazem uma DSE tendo consumo de energia como um requisito não funcional. Os autores utilizam um framework de cossimulação do consumo de energia que implementa várias técnicas de aceleração para fazer estimativas de energia rapidamente (LAJOLO et al., 2002). Os autores apresentam uma modelagem matemática do consumo de energia dos barramentos e da integração dos componentes, utilizando variáveis como voltagem, frequência e capacitância. A análise completa do consumo de energia é realizada, porém, através de ferramentas. Weng e outros também apresentam algumas equações baseadas em voltagem, frequência e capacitância para o consumo de energia de um sistema, embora apenas de forma ilustrativa, para detectar quais as fontes de consumo de energia de um sistema e assim

poder regulá-las em tempo de execução, uma técnica conhecida como Dynamic Power Management (DPM) (WENG; WANG; LIU, 2003).

Junior e Fröhlich propõem uma abordagem para a medição precisa de dados referentes a consumo de energia em sistemas embarcados alimentados por baterias. Esse consumo de energia pode então ser adequado para melhorar as métricas de desempenho e consumo de energia em sistemas de tempo real críticos (JUNIOR; FRÖHLICH, 2011). Algumas equações simples que demonstram como funciona a abordagem foram apresentadas.

Niar e Inglart apresentam comparações entre 3 simuladores que executam DSE. A modelagem de desempenho e consumo de energia em cada um dos simuladores é semelhante, embora cada uma possua suas peculiaridades. A forma como cada uma obtém suas estimativas dos valores também costuma ser diferente. Os autores apresentam as equações usadas internamente pelos simuladores. Desempenho e consumo de energia utilizam nos modelos variáveis como número de instruções por ciclo, energia consumida por ciclo e número de instruções do programa (NIAR; INGLART, 2006).

Há alguns grupos de pesquisa desenvolvendo estimativas de consumo de energia para serem usadas em tempo de execução. Dunkels e outros modelam matematicamente o consumo de energia do ponto de vista de um sistema em execução (DUNKELS et al., 2007). O objetivo é detectar de que forma a energia vem sendo consumida, para, em tempo de execução, regular o sistema e aumentar o tempo de vida do sistema ou da rede na qual ele está inserido (YANG; TOH; XIE, 2007). Mu e Lysecky, por sua vez, modelam o consumo de energia e o desempenho do sistema em tempo execução com outra finalidade: auxiliar na reconfiguração dinâmica de dispositivos com FPGA (MU; LYSECKY, 2011). Eles apresentam modelos complexos de consumo de energia e desempenho, mas que possuem foco em FPGA e suas peculiaridades.

Lee e outros propõem uma modelagem simples de consumo de energia, tamanho de código e desempenho, com o objetivo de realizar uma simples exploração e balancear esses requisitos durante as etapas iniciais do projeto. A equação de consumo de energia usada pelos autores trata o sistema como um conjunto de tarefas, e usa como variáveis das suas equações fatores como frequência da CPU, o valor de pior caso do número de ciclos para executar uma tarefa e o mínimo múltiplo comum dos períodos de todas as tarefas (LEE et al., 2008). Chede e Kulat apresentam modelos matemáticos para tamanho de código e consumo de energia que são implementados em um algoritmo. A abordagem também trata o sistema como um conjunto de tarefas e define o con-

sumo de energia de uma forma bem parecida (CHEDE; KULAT, 2008). No trabalho proposto aqui o sistema é tratado como um conjunto de componentes, logo, as equações usadas nesses trabalhos não são facilmente comparáveis com as vistas aqui.

Zhao e outros propõem uma abordagem para estimar e otimizar o consumo de energia em sistemas operacionais embarcados. Os autores calculam o consumo de energia do sistema a nível de instrução, simulando a execução do sistema para saber aproximadamente quais instruções são executadas e estimando o consumo de energia de cada uma. Dessa forma, o sistema embarcado é visto como um conjunto de instruções sendo executadas por um processador. O consumo de energia de uma instrução passando por um pipeline depende do consumo de energia do datapath, da cache, da Translation Lookaside Buffer (TLB), dos circuitos de clock e das diversas unidades lógicas (ZHAO et al., 2008).

Kim e outros propõem um modelo de consumo de energia semelhante, baseado em Inter-Prefetch Interval (IPI). Esse tipo de modelo detecta o consumo de energia do intervalo entre duas operações de prefetch do processador. Dessa forma, ele é visto também como um modelo de consumo de energia a nível de instrução. Em cada IPI existem 3 consumos de energia: o consumo de energia das instruções atuais passando pelo IPI, o consumo de energia de instruções iniciadas no IPI anterior e o consumo de energia causado por eventuais stalls na execução de uma instrução (KIM et al., 2007).

Azizi e outros também propõem uma abordagem semelhante para modelo de consumo de energia, com o objetivo de usá-la num framework de otimização que tenta balancear consumo de energia e desempenho em diferentes arquiteturas de processadores e circuitos. A abordagem dos autores também modela o consumo de energia a nível de instrução, embora considere fatores diferentes na equação. As equações levam em consideração a energia gasta em blocos lógicos, em registradores de pipeline e nos circuitos de clock (AZIZI et al., 2010).

Jayaseelan e outros, por sua vez, propõem uma técnica de análise estática para estimar o pior caso do consumo de energia de um sistema embarcado. Nesse caso, o sistema é tratado como um conjunto de blocos básicos de execução, e o consumo de energia de um bloco básico é definido como

$$energy_{BB} = dynamic_{BB} + switchoff_{BB} + leakage_{BB} + clock_{BB} \tag{3.1}$$

Eles informam que o consumo de energia pode ser dividido em energia consumida com instruções e energia consumida com pipeline.

Na equação acima, *dynamic* representa a energia consumida com instruções. O consumo de energia de pipeline, representado pelas variáveis $switchoff$, *leakeage* e *clock*, é proporcional ao tempo de execução do sistema, logo pode ser estimado através de variáveis de desempenho. Os autores então propõem um modelo de consumo de energia com várias equações derivadas desta e que estende modelos a nível de instrução e calcula o consumo de energia de instruções, de pipeline e também da rede de circuitos de clock. Para se calcular o pior caso do consumo de energia do sistema primeiramente calcula-se o consumo de energia de todos os blocos básicos do sistema separadamente. Então, usa-se uma formulação semelhante à de Integer Linear Programming (ILP), usada geralmente para cálculo do tempo de execução de pior caso de um sistema (JAYASEELAN; MITRA; LI, 2006).

Brandolese propõe uma abordagem híbrida para a modelagem do consumo de energia e do tempo de execução de um sistema. A abordagem une um modelo de consumo de energia a nível de instrução, que geralmente é utilizado num simulador, com um modelo de consumo de energia a nível de código fonte, usado geralmente numa análise estática. A primeira abordagem é lenta, e a segunda não contém detalhes arquiteturais que influenciam no consumo de energia. Através da abordagem híbrida proposta, é possível se estimar consumo de energia a nível de código fonte considerando-se também detalhes arquiteturais (BRANDOLESE, 2008).

Dhouib e outros apresentam uma abordagem onde eles tratam o sistema embarcado como um conjunto de tarefas e separam o sistema em 6 camadas distintas, somando o consumo de energia delas, uma de cada vez. Inicialmente, na primeira camada calcula-se o consumo de energia de todos os componentes, desconsiderando sistema operacional, quando não estão executando nenhuma aplicação. Então, nas camadas seguintes, soma-se o consumo de energia de cada tarefa considerando overhead específico relacionado a cada periférico e serviço de sistema operacional (DHOUIB et al., 2009). Essa abordagem difere das anteriores no sentido de tratar o sistema como um conjunto de tarefas, e não de instruções ou de blocos de código fonte.

Luo e outros utilizam duas abordagens para modelagem do consumo de energia de um sistema embarcado: um modelo a nível de microestruturas e um modelo a nível de circuito. A nível de microestruturas considera-se no cálculo do consumo de energia variáveis como o consumo de energia do processador quando ativo e quando não está executando instruções, o consumo da memória, da Universal Asynchronous Receiver/Transmitter (UART) e de outros periféricos. A nível

de circuito consideram-se variáveis como voltagem, frequência e capacitância (LUO et al., 2009). A modelagem é usada, porém, apenas para identificar todas as fontes de consumo de energia, e assim fornecer diretivas para que alterações a nível de código fonte e algoritmos possam ser realizadas a fim de diminuir o consumo de energia do sistema.

Num estudo realizado, Seo apresenta como modelar o consumo de energia em 5 estilos de arquitetura diferentes de sistemas distribuídos. As arquiteturas estudadas foram cliente-servidor, publish-subscribe, Peer-to-Peer (P2P), pipe-and-filter e uma baseada no C2, que é um protocolo de transferência de arquivos. O autor modela o sistema como um conjunto de componentes e um conjunto de conectores, que conectam componentes. A equação principal determina que

$$E(Comp_i) = E_{logic,i} + E_{commWithConn,i} \tag{3.2}$$

onde $E_{logic,i}$ é o consumo de energia do componente i quando está em execução, e $E_{commWithConn,i}$ representa a energia consumida ao trocar dados com conectores ligados a esse componente. A partir dessa equação, são apresentadas diversas equações que constituim um modelo que pode ser usado nos 5 estilos de arquitetura descritos (SEO, 2008).

Numa outra publicação, Seo e outros apresentam um modelo de consumo de energia semelhante, com foco em sistemas distribuídos baseados em Java. Além do consumo de energia dos componentes e dos conectores, considera-se também o consumo das infraestruturas usadas, como o coletor de lixo da Java Virtual Machine (JVM) (SEO; MALEK; MEDVIDOVIC, 2007). O trabalho dos autores também foi usado como base para alguns modelos que foram propostos nessa dissertação.

Vários dos trabalhos descritos acima modelam o consumo de energia com a mesma finalidade da proposta desta dissertação: considerar o consumo de energia como um requisito não funcional do sistema durante o projeto, podendo usá-lo durante a fase de DSE. Diversas abordagens são apresentadas para modelar o consumo de energia. De fato, pode-se visualizar um sistema embarcado através de 6 níveis diferentes de abstração.

Como visto na Fig. (5), do ponto de vista de hardware, os níveis de abstração existentes são: nível de circuito, RTL e nível de microestrutura. Do ponto de vista de software, pode-se visualizar o sistema embarcado através do nível de código fonte, do nível de algoritmo e do nível de arquitetura de software (LUO et al., 2009).

Este trabalho, como será mostrado no próximo capítulo, baseou-

Figura 5: Principais níveis de avaliação de consumo de energia (LUO et al., 2009).

se em várias dessas abordagens e criou-se um novo modelo matemático para consumo de energia em sistemas embarcados. Pode-se notar, porém, que os trabalhos descritos acima não levam em consideração muitos requisitos não funcionais, inclusive a maioria deles apenas considera o consumo de energia. Também percebe-se isso nos trabalhos que serão descritos nas próximas seções. A abordagem que será proposta nessa dissertação, por tratar vários outros requisitos, acaba sendo mais abrangente do que essas que estão sendo apresentadas.

3.4 MODELANDO DESEMPENHO

Como foi mostrado anteriormente, além de consumo de energia, um requisito muito comum no projeto de um sistema embarcado é o desempenho, principalmente quando se trata de um sistema embarcado de tempo real. De fato, todo sistema de tempo real possui requisitos de desempenho, embora nem todo sistema modele o desempenho como um requisito. Alguns grupos de pesquisa atingem melhor desempenho apenas regulando o consumo de energia, sem modelar o desempenho propriamente dito. Chede e Kulat implementam a técnica de DVS para conseguir isso em sistemas de tempo real (CHEDE; KULAT, 2008).

Como foi visto anteriormente, Azizi e outros também utilizam um framework de otimização que balanceia o consumo de energia com o desempenho em diferentes arquiteturas e circuitos (AZIZI et al., 2010). Lee e outros também apresentam uma abordagem para balancear con-

sumo de energia e desempenho, considerando também tamanho de código. Porém, esse balanceamento é realizado de forma diferente. Os autores consideram que cada tarefa possui uma lista de implementações e cada implementação é descrita como duas variáveis: o tamanho do código da tarefa e o pior caso da quantidade de ciclos necessários para executar a tarefa. O balanceamento é feito então considerando outras métricas e escolhendo a implementação mais adequada para o cumprimento de todos os requisitos (LEE et al., 2008).

Como visto anteriormente, Niar e Inglart apresentam comparações entre 3 simuladores que executam DSE e mostram as equações usadas internamente nesses simuladores. Num dos simuladores, o desempenho é medido em ciclos, com base no número de instruções a executar, e a quantidade de instruções que executam em um ciclo. Nos outros simuladores são considerados também nas equações dados como quantidade de falhas de cache, quantidade de falhas de predição de desvios condicionais, e também qual a punição por essas falhas (NIAR; INGLART, 2006).

Mu e Lysecky modelam consumo de energia e desempenho matematicamente com foco na reconfiguração dinâmica. Usando informações disponíveis em tempo de execução (por exemplo detalhes a respeito do comportamento da cache), uma ferramenta estima dados de desempenho e consumo de energia e os informa para o processo de reconfiguração de hardware, que toma decisões em tempo de execução usando as equações modeladas (MU; LYSECKY, 2011).

Allara e outros apresentam uma estratégia para modelagem de requisitos durante as etapas iniciais de um projeto visando sua utilização na etapa de DSE. O foco da abordagem é desempenho, e a estimativa de desempenho é realizada a nível de sistema, baseada em informações vindas de um modelo de desempenho que considera características de baixo nível da execução do código (ALLARA et al., 1998). Numa outra publicação, as equações que definem o desempenho são apresentadas. Para cada tipo de instrução do programa (por exemplo, *if*, *while*), uma equação é usada para calcular o tempo de execução daquela instrução (ALLARA et al., 1997).

Brandolese, por sua vez, modela o desempenho não usando instruções do programa, mas sim considera cada bloco básico do sistema como um nodo e cria um grafo com esses nodos. A modelagem matemática usa matrizes de nodos. Cálculos realizados sobre essas matrizes geram como resultado o consumo de energia e o desempenho do sistema (BRANDOLESE, 2008).

Kalavade e Moghé propõem uma ferramenta analítica chamada

AsaP para a estimativa de desempenho de sistemas embarcados. Essa ferramenta constrói modelos dos sistemas e caracteriza a carga de processamento do sistema como um processo estocástico. A saída da ferramenta é a distribuição exata do intervalo de processamento de cada tarefa. No modelo usado o sistema é visto como um conjunto de tarefas a serem executadas, estruturadas num grafo de tarefas (KALAVADE; MOGHé, 1998).

Hu e outros propõem uma modelagem para se determinar tempo de execução de tarefas específicas e também do sistema como um todo. O tempo de execução de cada tarefa é modelado matematicamente como uma variável probabilística discreta, e todas as tarefas são analisadas de forma coletiva, com o objetivo de considerar dependências entre tarefas na modelagem. Através da métrica proposta, é possível comparar facilmente diferentes alternativas no processo de DSE (HU; ZHOU; SHA, 2001).

Com foco em sistemas embarcados de tempo real, Gustafsson e outros apresentam um modelo para cálculo de Worst-Case Execution Time (WCET) de um sistema. Ao contrário das técnicas comuns para cálculo do WCET que o fazem após a implementação do sistema, os autores modelam um WCET aproximado durante as fases iniciais do projeto, com objetivo de ajudar na etapa de DSE. A abordagem trata o sistema como um conjunto de construções de código, que podem ser operadores lógicos/aritméticos, acessos a variáveis do programa, comandos que alteram o fluxo de controle do sistema, ou até mesmo construções mais complexas que ocorrem com frequência. O modelo matemático calcula o WCET analisando o tempo de execução de todas as construções do sistema e a quantidade de vezes que elas são executadas (GUSTAFSSON et al., 2009).

3.5 MODELANDO USO DE MEMÓRIA E ÁREA

Há algumas abordagens que tratam uso de memória nos seus modelos matemáticos. Uso de memória é um requisito que abrange apenas componentes de software, e não são consideradas aqui as memórias de vídeo, cache de disco ou de processador. Apesar de todos os sistemas computacionais com componentes de software utilizarem esse recurso, geralmente apenas sistemas embarcados se preocupam em usá-lo com moderação, pois sempre existe pouca memória disponível para um sistema embarcado usar. Uma razão para isso é certamente o custo alto desse recurso. Modelar um sistema embarcado de forma que ele use

bem esse recurso pode diminuir gastos com o projeto e trazer vantagens no mercado.

Uma equação de uso de memória tem geralmente como resultado a quantidade de bytes que os componentes de software possuem de código e dados. Como foi mostrado na seção anterior, o trabalho de Lee e outros modela o sistema como um conjunto de tarefas e cada tarefa é modelada como uma lista de implementações daquela tarefa. Cada implementação é descrita como duas variáveis: o tamanho de código da tarefa e o pior caso da quantidade de ciclos necessários para executar aquela implementação da tarefa, não considerando aspectos como gerenciamento de memória virtual. Já a função custo é definida como

$$f(S,E) = \alpha\frac{S}{\bar{S}} + \beta\frac{E}{\bar{E}} \tag{3.3}$$

onde S representa a utilização de memória do sistema, E representa o consumo de energia do sistema e os coeficientes α e β são constantes que permitem aos projetistas definirem uma importância maior para um dos dois requisitos. Os termos $\bar{S}$ e $\bar{E}$ representam o limite superior da utilização de memória e do consumo de energia respectivamente. Através dessas informações, um framework ajuda a escolher a melhor implementação para aquele sistema, dependendo de seus requisitos (LEE et al., 2008). Dessa forma, se um dos requisitos for pouco uso de memória, o sistema pode usar uma das implementações que possui o menor tamanho de código da tarefa.

Aplicações embarcadas de tempo real precisam de código dedicado. Como os processadores embarcados possuem pouca memória disponível, é essencial otimizar o tamanho do código para uma melhor utilização da memória. Chede e Kulat realizam uma otimização do código do programa para reduzir seu tamanho. Através do modelo proposto, é possível reduzir bastante o tamanho do código de um sistema respeitando os requisitos de tempo real e consumo de energia (CHEDE; KULAT, 2008).

Weerasekera e outros discutem métricas realistas para a análise de desempenho e custo de SoCs 3D. Através de modelos equacionais, os autores fazem um balanço das melhorias trazidas pelo uso de circuitos 3D, como melhor desempenho, em relação com os maiores custos que a tecnologia traz para o projeto. Nesse balanço, um fator que os autores precisaram considerar foi a área do sistema final, visto que ela influencia tanto o custo quanto o desempenho. Se a área não for informada pelo fabricante da IP, ela pode ser estimada. Os autores apresentam

então uma modelagem matemática da área do circuito, usando como variáveis o número de transistores e seus tamanhos, por exemplo (WEERASEKERA et al., 2009).

3.6 TRATANDO INCERTEZAS

Como foi apresentado nesse capítulo, vários grupos de pesquisa tratam incertezas nos seus modelos. O tratamento de incertezas se faz necessário quando os projetistas não possuem certeza a respeito de alguma característica de algum componente mas precisam usá-la em alguma equação. Li e outros apresentam um estudo do impacto que informações pouco precisas de tempo de execução causam ao desempenho da alocação de recursos nos estágios iniciais do projeto de sistemas embarcados multicore. A alocação de recursos em sistemas multicore é realizada com base em estimativas probabilísticas do tempo de execução das tarefas executando em diferentes cores. Os autores então propõem um método para medir a degradação de robustez do sistema e avaliar os efeitos causados por parâmetros probabilísticos imprecisos no desempenho da alocação de recursos, tentando assim evitar que as incertezas prejudiquem o projeto (LI et al., 2011).

Além do tempo de execução incerto de tarefas, incertezas também podem ser encontradas em ações humanas com base em experiências passadas e também no comportamento de componentes que não tenham sido muito bem testados, verificados e validados (NOGUEIRA et al., 2011). Kujawski e outros estudaram essas incertezas baseadas no comportamento humano. Quando projetistas precisam estimar algum valor utilizando experiência pessoal, fatores psicológicos como excesso de confiança podem afetar as estimativas, atribuindo valores muito otimistas às variáveis. Os autores usam uma distribuição Weibull de probabiliade para modelar esse tipo de variável e uma simulação de Monte Carlo para avaliar os riscos (KUJAWSKI; ALVARO; EDWARDS, 2004).

Axelsson inclui incertezas na modelagem de custo financeiro, como visto na seção anterior, e mostra como tratar os riscos envolvidos nisso. Ele também trata atributos como uma distribuição Weibull de probabilidade e essas incertezas também são avaliadas usando simulação de Monte Carlo, onde os riscos envolvidos são detectados. Por exemplo, um conjunto de tarefas precisa ser escalonado de forma a cumprir um deadline. Porém, o tempo de execução das tarefas não é uma variável constante, então mesmo que o escalonamento mostre que o deadline será cumprido, na prática existe uma incerteza a respeito

disso. Essa incerteza deve estar presente no modelo, de forma a evitar problemas futuros. A simulação de Monte Carlo, nesse caso, é útil para identificar a proporção dos riscos envolvidos nesse escalonamento (AXELSSON, 2005). Essas técnicas apresentadas para tratamento de incertezas, porém, pressupõem que os projetistas estão cientes da natureza das incertezas que surgirão ao longo do ciclo de vida do projeto, o que nem sempre ocorre (BRINDLE, 2005), onde então busca-se esse conhecimento por meio de experimentação.

3.7 TRATANDO DEPENDÊNCIAS ENTRE VARIÁVEIS

A dependência entre variáveis deve ser considerada na criação dos modelos matemáticos de um sistema. O modelo matemático pode gerar resultados errados se não tratar nas suas equações dependências existentes entre diferentes variáveis. Trabalhos que modelam o sistema como um conjunto de tarefas a serem executadas, por exemplo, podem permitir que essas tarefas dependam de outras, principalmente em sistemas multicore (KALAVADE; MOGHé, 1998; HU; ZHOU; SHA, 2001).

Clemen e outros apresentam um estudo comparativo de 6 métodos diferentes de avaliação de dependências entre duas variáveis. Após uma avaliação dos métodos, eles concluíram que o método mais recomendado para avaliação de dependências é o método de correlação. Esse método consiste numa estimativa direta de uma correlação e requer que os projetistas entendam a noção de correlação. Uma correlação perto de 0,00 implica uma fraca relação entre as variáveis e uma correlação perto de +1,00 (-1,00) indica uma forte relação positiva (negativa) (CLEMEN; FISCHER; WINKLER, 2000).

Assim como as incertezas no valor das variáveis, avaliar dependências entre variáveis também é uma atividade de estimativa por parte dos projetistas que pode se basear em experiência pessoal com projetos anteriores. Dessa forma, os mesmos fatores psicológicos também podem afetar essas estimativas de dependência. Kujawski e outros informam que em geral, dados N elementos de custo, há N*(N - 1)/2 coeficientes de correlação, e eles não podem assumir valores entre -1,00 e 1,00 arbitrariamente porque precisam satisfazer algumas propriedades matemáticas. Os autores também apresentam um modelo de correlação que permite reduzir o número de parâmetros necessários para especificar uma matriz de correlações válida. Algumas ferramentas são capazes de checar os coeficientes e verificar se eles estão consistentes (KUJAWSKI; ALVARO; EDWARDS, 2004). Uma forma então de tratar es-

sas dependências nos modelos matemáticos é incluir esses coeficientes de correlação nas equações criadas.

3.8 CONSIDERAÇÕES

A maior característica em comum entre todos os trabalhos correlatos apresentados nesse capítulo é o fato deles considerarem nos seus modelos apenas um ou dois requisitos do sistema. Isso se mostra como uma grande limitação desses trabalhos. Além disso, esses trabalhos não costumam modelar custos muito diferentes dos tradicionais. Como o objetivo da proposta desta dissertação é apresentar uma modelagem que suporte os principais custos de sistemas embarcados, considerou-se na proposta os principais requisitos que a comunidade científica trata: consumo de energia, desempenho, utilização de memória e custo financeiro.

Tabela 1: Sumário dos principais trabalhos correlatos a respeito de consumo de energia.

Nível de abstração	**Variáveis consideradas**
Nível de circuito	Tensão, frequência e capacitância (CHEDE; KULAT, 2008; LUO et al., 2009)
Nível de arquitetura de software e algoritmos	Escalonamento, comunicação, sincronização, acesso a dispositivos periféricos (DHOUIB et al., 2009)
Nível de instruções sendo executadas pelo processador	Quantas instruções (ou blocos de instruções) executam, estágios de pipeline, caminhos de execução (ZHAO et al., 2008; KIM et al., 2007; AZIZI et al., 2010; JAYASEELAN; MITRA; LI, 2006)
Nível de microestrutura	Consumo de energia de componentes e interfaces de comunicação (por exemplo, processador, memória, rádio) (LUO et al., 2009; SEO, 2008)
Híbrido, nível de instruções no processador e nível de código fonte	Quantas instruções executam, blocos básicos de execução (BRANDOLESE, 2008)

De forma a criar um sumário dos principais trabalhos correlatos que modelam consumo de energia, criou-se a Tab. (1) acima. A tabela

ilustra mostra diferentes variáveis que são consideradas quando modela-se o consumo de energia em diferentes níveis de abstração. Note que a maioria desses trabalhos modela apenas consumo de energia.

Da forma forma foi criada a Tab. (2) abaixo, onde são reunidos os trabalhos mais importantes a respeito da modelagem de desempenho. Veja novamente como a maioria dos trabalhos modela apenas desempenho.

Tabela 2: Sumário dos principais trabalhos correlatos a respeito de desempenho.

Tipo de abordagem	**Variáveis consideradas**
Melhorar desempenho regulando outros fatores, como consumo de energia e tamanho de código	Variáveis inerentes a consumo de energia e tamanho de código (CHEDE; KULAT, 2008; AZIZI et al., 2010; LEE et al., 2008)
Tarefas como variáveis probabilísticas	Deadlines a serem cumpridos, dependências entre tarefas (HU; ZHOU; SHA, 2001)
A nível de instruções das tarefas	Número de instruções, instruções por ciclo, falhas de cache (NIAR; INGLART, 2006)
Worst-case Execution Time para sistemas de tempo real	Conjuntos de construções de código (e.g. operações lógicas, desvios, acessos a variáveis) (GUSTAFSSON et al., 2009)
Modelagem de desempenho junto com energia e memória	Blocos básicos e grafos de execução (BRANDOLESE, 2008)

4 MODELAGEM PROPOSTA

A proposta desta dissertação será apresentada neste capítulo em detalhes. Trata-se de uma modelagem de custos de um sistema embarcado, onde diferentes tipos de custos são considerados, como consumo de energia, desempenho, uso de memória, etc. Além disso, a forma como os requisitos são tratados permite que os modelos sejam usados para uma grande variedade de sistemas embarcados. Uma modelagem flexível, que abrange os principais requisitos quantitativos, que pode ser usada em uma grande quantidade de projetos diferentes e que possui fácil compreensão por parte de otimizadores e usuários é a maior contribuição deste trabalho.

Através de uma especificação inicial do sistema, o objetivo é capturar os requisitos do sistema e expressá-los matematicamente. A modelagem consiste então em um grupo de equações que descreve diferentes requisitos do sistema. Modelar de forma consistente e padronizada todos os possíveis requisitos de um sistema embarcado é uma contribuição científica. Com base nessas equações de cada requisito, uma exploração de espaço de projeto pode ser fácil e eficientemente realizada.

Como foi discutido no capítulo anterior, podemos analisar um sistema embarcado como um conjunto de artefatos $S^s = \{a_1, a_2, ..., a_n\}$, onde n denota o número de artefatos e s representa a semântica dada a cada um dos artefatos do sistema. Um artefato pode ser uma tarefa executada pelo sistema, pode ser um componente do sistema, entre outras possibilidades. Nesta dissertação tratamos todo o sistema como um conjunto de componentes. Assim, cada componente c possui um conjunto T de características relevantes para cada requisito r do sistema, dado por $T_c^r = \{t_1, t_2, ..., t_m\}$, onde m é o número de características. Por exemplo, a corrente drenada por um componente qualquer c é uma característica relevante para o requisito de consumo de energia. O quanto aquele componente ocupa de memória é uma característica relevante para o requisito de uso de memória.

Dessa forma, um sistema embarcado pode ser visto como um conjunto de componentes, sendo que cada componente possui um conjunto de características importantes para cada requisito do sistema. Para cada requisito r, propõe-se uma equação onde são analisadas separadamente as características de cada componente c, tendo como resultado o custo R daquele componente para o cumprimento daquele

requisito, dada por

$$R_c^r = equation(T_c^r) \tag{4.1}$$

Soma-se então o custo de cada componente c para obter-se o custo total daquele requisito r para o sistema

$$C^r = \sum_{c \in n} R_c^r \tag{4.2}$$

que pode ser expresso em Joules, ou em bytes, ou em segundos, dependendo da equação.

A equação 4.2 consiste então numa equação genérica para um sistema embarcado qualquer. Todas as equações descritas nas próximas seções foram criadas usando-a como base. Outras equações podem futuramente ser criadas sem dificuldades utilizando esse processo, tornando extensível a abordagem proposta. Nas próximas seções serão apresentadas em detalhes cada uma das equações que foram criadas.

4.1 CONSUMO DE ENERGIA

O consumo de energia é talvez o requisito não funcional mais relevante para os sistemas embarcados operados a bateria. Muitos dos lugares onde esses sistemas estão sendo instalados não possuem uma fonte de energia externa, e o uso consciente de uma fonte de energia própria deve ser uma prioridade. O consumo de energia também se relaciona com vários outros requisitos. Quanto mais rápido o microprocessador conseguir executar as instruções do sistema, mais energia ele consumirá. Quanto maior a capacidade das baterias do sistema, mais caro o sistema ficará.

Há várias formas, porém, de se modelar o consumo de energia. Pode-se levar em consideração a corrente de cada componente do sistema, a energia gasta para executar cada instrução ou cada bloco básico, a energia gasta a nível de portas lógicas, etc. Neste trabalho a equação de consumo de energia calcula o consumo de energia esperado para o sistema usando como base o conjunto de componentes de hardware do sistema e um conjunto de conectores que ligam esses componentes a outros componentes de hardware.

Pode-se considerar o consumo de energia E de um sistema em-

barcado então como sendo

$$E = \sum_{c \in C} cons_c \tag{4.3}$$

Ou seja, soma-se o consumo de cada componente de hardware. O consumo de energia de cada componente está dividido em duas partes, como descrito na equação

$$cons_c = logic_c + comm_c \tag{4.4}$$

$logic_c$ representa o consumo de energia do componente c enquanto executa suas funcionalidades normais, enquanto $comm_c$ consiste no consumo total de energia de comunicação para o componente c.

O consumo de energia do componente executando suas funcionalidades depende do conjunto S_c de estados de execução possíveis daquele componente e pode ser descrito como

$$logic_c = \sum_{s \in S_c} \frac{curr_{c,s} \times dutycycle_{c,s}}{exp_{c,s} \times dev_{c,s}} \tag{4.5}$$

Soma-se então a energia consumida em cada estado s de execução do componente. Para cada estado, o consumo do componente c é dado então pela corrente drenada pelo componente naquele estado vezes o *duty cycle* daquele componente c naquele estado s. O *duty cycle* consiste em quanto tempo o componente fica em tal estado em relação ao período total de execução do componente.

Essa multiplificação é afetada por dois possíveis erros. $dev_{c,s}$ consiste num erro referente ao estágio de desenvolvimento do componente relativo ao estado s. Esse erro, assim como os outros vistos ao decorrer deste capítulo, causa um aumento do custo de energia calculado, indicando uma estimativa mais conservadora. Um componente que não esteja completamente desenvolvido sofrerá então um acrécimo em seu consumo de energia, proveniente de possíveis estimativas muito otimistas por parte dos projetistas (KUJAWSKI; ALVARO; EDWARDS, 2004). Já $exp_{c,s}$ representa o erro referente à experiência dos projetistas no funcionamento geral do componente c no estado s.

O consumo de energia de comunicação de um componente c depende da quantidade J_c de interfaces de comunicação que ele tem e de quantas vezes cada interface é invocada. Soma-se então o consumo de energia de cada interface de comunicação j para se obter o consumo de

energia de comunicação do componente, como na equação

$$comm_c = \sum_{j \in J_c} \frac{comm_j \times count_j}{exp_j \times dev_j \times exp_{count_j}} \tag{4.6}$$

onde $comm_j$ é o consumo de energia de uma invocação da interface j. $count_j$ representa a quantidade de invocações da interface j. Novamente exp_j representa um erro referente à experiência dos projetistas no funcionamento geral da interface em questão, dev_j representa um erro referente ao estágio de desenvolvimento da interface j e exp_{count_j} é um erro que representa a experiência dos projetistas na hora de atribuir um valor para $count_j$.

A experiência dos projetistas em relação ao valor de uma constante qualquer ou em relação ao funcionamento de algum artefato de sistema θ qualquer é dada por $EXP(\theta)$. Essa experiência pertence ao intervalo $\Re[0, 1]$ e é atribuída pelos projetistas para cada θ. Essa experiência $EXP(\theta)$ é transformada no erro exp_θ usado nas equações acima através da equação

$$exp_\theta = x_\theta + EXP(\theta) \times y_\theta \tag{4.7}$$

que multiplifica a experiência dos projetistas por uma constante y_θ e depois soma o resultado a uma outra constante x_θ. x_θ deve pertencer ao intervalo $\Re(0, 1]$ e y_θ ao intervalo $\Re[0, 1)$, sendo que $x_\theta + y_\theta = 1$ para qualquer θ.

O objetivo dessas constantes é controlar o impacto causado pela falta de experiência dos projetistas. Há situações onde $EXP(\theta) = 0,9$, ou seja, 90% de experiência, é considerado um valor razoável e pouco preocupante. Há outros casos, porém, onde os valores em questão são muito críticos e 90% de experiência pode ser considerado muito abaixo do desejado. Dessa forma, um mesmo valor de $EXP(\theta)$ pode ter diferentes semânticas dependendo do caso. Na abordagem aqui apresentada, a semântica é atribuída através das constantes x_θ e y_θ.

Essas constantes são responsáveis por balancear o erro exp_θ com base em quão grave é ter um valor de $EXP(\theta)$ abaixo de 100%. Através da equação 4.7, considera-se que o valor de exp_θ está sempre condicionado ao intervalo $\Re(EXP(\theta), 1]$. Dessa forma, no cenário onde um valor baixo para $EXP(\theta)$ é considerado muito grave, exp_θ terá um valor muito próximo a $EXP(\theta)$. Por outro lado, se uma baixa experiência dos projetistas a respeito de θ não for muito prejudicial ao projeto, o valor de exp_θ será próximo a 1, ou seja, o erro será próximo a nulo. Para se controlar esse balanceamento no valor de exp_θ, muda-se os va-

lores de x_θ e y_θ. Quanto mais grave se considerar um valor de $EXP(\theta)$ abaixo de 100%, menor deve ser o valor de x_θ.

Por exemplo, considere $EXP(\theta) = 0,4$. Dessa forma, se θ for uma constante, os projetistas possuem uma experiência de 40% a respeito do valor de θ, o que informa o quanto pode-se confiar no valor de θ. Caso θ seja um artefato do sistema, pode-se dizer que os projetistas possuem uma experiência de 40% a respeito das características e do funcionamento desse artefato. Considere um valor próximo de 0 para x_θ, como $x_\theta = 0,1$. Assim, passa-se a tratar $0,4$ como um valor muito baixo para $EXP(\theta)$ e isso é considerado grave. Com $y_\theta = 0,9$, tem-se através da equação 4.7 que $exp_\theta = 0,46$. Tem-se então um erro alto, de $0,46$.

Considere agora que um valor baixo para $EXP(\theta)$ não é algo grave, atribuindo um valor alto para x_θ. Considere então $x_\theta = 0,95$. Com esse valor de x_θ, tem-se $y_\theta = 0,05$, e portanto $exp_\theta = 0,97$, o que de acordo com a semântica que foi dada para o erro exp_θ, é considerado um erro muito pequeno. Então, partindo de um mesmo valor de $EXP(\theta)$, em um caso atribui-se um erro de $0,46$ para a equação, e no outro caso um erro de $0,97$.

Exatamente o mesmo raciocínio é aplicado em relação ao estágio de desenvolvimento de um artefato de sistema λ qualquer. O estágio de desenvolvimento do artefato λ é dado por $DEV(\lambda)$, também atribuído pelos projetistas e também pertencente ao intervalo $\Re[0,1]$, sendo que $DEV(\lambda) = 1$ indica que o artefato está completamente desenvolvido e já foi testado e validado. Esse estágio de desenvolvimento é transformado no erro dev_λ visto anteriormente através da equação

$$dev_\lambda = x_\lambda + DEV(\lambda) \times y_\lambda \tag{4.8}$$

A semântica das constantes x_λ e y_λ é a mesma vista anteriormente para x_θ e y_θ; elas controlam o quão grave é para o projeto o valor de $DEV(\lambda)$ ser menor do que 100%.

4.2 DESEMPENHO

Agora será apresentada uma modelagem do desempenho de um sistema embarcado, e dados usados nas equações de consumo de energia podem ser usados aqui também. Nessa modelagem trata-se o sistema como um conjunto de tarefas a serem executadas pelo microprocessador. Em um projeto baseado em componentes, considera-se como tarefas a aplicação e a utilização de rotinas de cada componente. Es-

sas tarefas, inclusive, podem ser de tempo de real crítico, podem possuir deadlines mais tolerantes ou podem não possuir deadlines. Além disso, essas equações de desempenho possuem a flexibilidade de permitir que os projetistas calculem o tempo de execução não do software todo, mas só de algumas tarefas específicas. Por exemplo, considere que um requisito do sistema seja que uma tarefa específica execute em determinado tempo. Os projetistas podem então usar o modelo de desempenho proposto para calcular o tempo de execução apenas daquela tarefa específica. Uma limitação, porém, do modelo proposto, é que as equações de desempenho apresentadas nessa seção só podem ser usadas para determinar o desempenho de tarefas implementadas em software.

Como será apresentado a seguir, existe uma única equação de desempenho, que pode ser usada tanto para sistemas de tempo real como também para sistemas não de tempo real. Organizar a equação dessa forma acaba adicionando indeterminismos que prejudicam a avaliação do desempenho das tarefas de tempo real. A equação, porém, possibilita que todos os termos não determinísticos sejam desconsiderados numa, permitindo uma avaliação correta do desempenho. A equação possui também outras flexibilidades, que serão descritas em detalhes ao longo desta seção.

O tempo de execução T de um sistema é dado então por

$$T = \sum_{h \in HRT} time_h + \sum_{t \in SRT} time_t + \sum_{n \in NRT} time_n \tag{4.9}$$

sendo HRT o conjunto de tarefas de tempo real crítico, SRT o conjunto de tarefas que são de tempo real mas que permitem descumprimento ocasional de deadlines, e NRT o conjunto de tarefas que não possuem deadlines a cumprir. Calcula-se então separadamente o tempo de execução das tarefas e somam-se todos ao final.

O tempo de execução de cada tarefa h de tempo real crítico é dado por

$$time_h = (\frac{cycles_h}{f} + wait_h) \times count_h \tag{4.10}$$

onde $cycles_h$ é a quantidade de ciclos necessários para executar a tarefa h e f é a frequência do microprocessador. O termo $wait_h$ representa um tempo de espera arbitrário que a tarefa h fica parada esperando durante a sua execução. Por exemplo, no caso de um nodo sensor, quando uma thread de emergência é ativada para recuperar o sistema de uma condição crítica atingida, a thread pode querer esperar algum tempo

para o ambiente se estabilizar, antes de medir novamente a condição e verificar se a situação não está mais crítica. Esse tempo de espera é somado ao tempo de execução da thread. Esse termo de espera foi inserido fora do cálculo da quantidade de ciclos necessários para executar a tarefa porque isso permite que o mecanismo de espera não seja necessariamente implementado em software dentro do código da tarefa.

Por fim, o termo $count_h$ representa a quantidade de vezes que a tarefa h será executada. Porém, muitas vezes a tarefa em questão ficará executando ininterruptamente. O termo $count_h$ deve então ser limitado a um número especificado pelos projetistas. Por exemplo, mesmo que a tarefa h seja executada por um ano ininterruptamente, os projetistas podem querer saber o tempo de execução do sistema para 10 execuções da tarefa h.

Equações semelhantes se aplicam para o cálculo do tempo de execução de cada tarefa t tolerante ao deadline e de cada tarefa n sem deadline

$$time_t = (\frac{cycles_t}{f} + wait_t) \times count_t \tag{4.11}$$

$$time_n = (\frac{cycles_n}{f} + wait_n) \times count_n \tag{4.12}$$

Os termos $count_h$, $count_t$ e $count_n$ também possuem outra utilidade. Como visto acima, os projetistas podem não querer saber o tempo de execução de todo o sistema, e sim apenas de uma ou algumas tarefas do sistema, limitando o requisito de desempenho. Isso é possível através do modelo proposto atribuindo valor 0 para os termos $count_h$, $count_t$ ou $count_n$ de uma tarefa qualquer. Isso torna nulo o tempo de execução da tarefa em questão no cálculo do tempo de execução do sistema.

A quantidade de ciclos necessários para executar cada tarefa h de tempo real crítico é dada por

$$cycles_h = \sum_{it \in IT} \frac{instr_h(it) \times \mu}{IPC(it) \times exp_h \times exp_\mu} \tag{4.13}$$

Seja IT o conjunto dos tipos de instruções existentes. Para cada tipo de instrução it, $instr_h(it)$ é a quantidade esperada de instruções daquele tipo na tarefa h e $IPC(it)$ é a quantidade de instruções daquele tipo que o processador consegue executar por ciclo. O erro exp_h repre-

senta a falta de experiência dos projetistas a respeito das instruções pertencentes a h, e como é o funcionamento da tarefa h. μ é uma constante que representa o pior caso do efeito causado por laços de repetição, falhas de cache e de predição de desvios condicionais no número de instruções executadas. Assim como anteriormente, exp_μ representa um erro proveniente da falta de experiência da equipe de projetistas a respeito da atribuição de um valor a μ. Por fim, somando o resultado para cada tipo de tarefa it, obtem-se a quantidade de ciclos necessários para executar aquela tarefa.

Para cada tipo de instrução it, a quantidade de instruções executadas daquele tipo em uma tarefa de tempo de real h é dada por

$$instr_h(it) = \frac{wcIns(it)}{exp_{wcIns(it)}} \tag{4.14}$$

onde $wcIns(it)$ é o pior caso do número de instruções do tipo it que serão executadas na tarefa h. $exp_{wcIns(it)}$ é novamente o erro atribuído à falta de experiência dos projetistas na atribuição de um valor para $wcIns(it)$.

Para cada tarefa h, o tempo de espera $wait_h$ é dado por

$$wait_h = \sum_{bs \in BS_h} \frac{waitTime_{bs} \times wcCount_{bs}}{exp_{wcCount_{bs}}} \tag{4.15}$$

Consideramos uma instrução do tipo *busy* como sendo uma instrução qualquer que causa um evento de espera na execução de uma tarefa qualquer. Usaremos o símbolo bs para nos referirmos nas nossas equações à instruções desse tipo. Uma instrução bs é então qualquer instrução do tipo *busy* dentro da tarefa h. O conjunto BS_h é então o conjunto de todas as instruções bs dentro de uma tarefa h. $waitTime_{bs}$ é o tempo determinado pelos projetistas que a tarefa h deve esperar na execução de cada instrução bs. $wcCount_{bs}$ é o pior caso da quantidade de vezes que a instrução bs será executada durante cada execução da tarefa h. Por fim, $exp_{wcCount_{bs}}$ é um erro referente à falta de experiência dos projetistas em relação à atribuição de um valor à $wcCount_{bs}$.

Já a quantidade de ciclos necessários para executar cada tarefa de tempo real não crítico t é dada por

$$cycles_t = \sum_{it \in IT} \frac{instr_t(it) \times O_t}{IPC(it) \times exp_t \times dev_t} \tag{4.16}$$

$instr_t(it)$ representa a quantidade de instruções de tipo it exe-

cutadas na tarefa t e o $IPC(it)$ é novamente o número de instruções de tipo it que o microprocessador consegue executar por ciclo. exp_t representa um erro referente à falta de experiência dos projetistas a respeito do funcionamento da tarefa t e de quais instruções ela possui. dev_t representa um erro referente ao estágio de desenvolvimento da tarefa t, que pode ainda não ter terminado sua implementação, ou ainda não ter sido testada, por exemplo.

O_t, por sua vez, é definido como

$$O_t = \frac{o_1 \times \mu + o_2 \times \phi}{exp_\mu \times exp_\phi} \tag{4.17}$$

O termo O_t representa o efeito causado por laços de repetição, falhas de cache e de predição de desvios condicionais no número de instruções executadas. Há duas estimativas para esse valor, uma de pior caso, μ, usada já anteriormente, e uma de valor esperado, ϕ, aparecendo pela primeira vez nessa equação. As constantes o_1 e o_2 são definidas pelos projetistas e balanceiam quanto de cada estimativa os projetistas desejam considerar para a tarefa t. Para isso, deve-se respeitar $o_1 + o_2 = 1$. exp_μ e exp_ϕ também representam erros nas estimativas de μ e ϕ.

Para cada tipo it, a quantidade de intruções daquele tipo executadas em uma tarefa t é

$$instr_t(it) = \frac{eIns(it)}{exp_{eIns(it)} \times dev_t} \tag{4.18}$$

onde $eIns(it)$ representa a quantidade esperada de instruções do tipo it a serem executadas na tarefa t. $exp_{eIns(it)}$ consiste no erro causado pela falta de experiência dos projetistas na estimativa de um valor para $eIns(it)$. dev_t novamente representa o erro referente ao estágio de desenvolvimento da tarefa t.

Para cada tarefa t, o tempo de espera $wait_t$ é dado por

$$wait_t = \sum_{bs \in BS_t} \frac{waitTime_{bs} \times Q_t}{dev_t} \tag{4.19}$$

BS_t é novamente o conjunto de instruções do tipo *busy* dentro da tarefa t. $waitTime_{bs}$ é novamente o tempo de espera de cada instrução bs. O termo Q_t é responsável por indicar a quantidade de vezes que a instrução bs é executada. No denominador da equação aparece novamente o erro dev_t. Ao invés de usar como denominador um erro referente à experiência dos projetistas, nessa equação do termo $wait$ re-

ferente à tarefa t usa-se como denominador um erro referente ao estágio de desenvolvimento da tarefa pois considerou-se que em tarefas que não são de tempo real crítico o estágio de desenvolvimento da tarefa afeta o seu desempenho significamente mais do que a experiência dos projetistas a respeito do funcionamento dessa tarefa. O termo Q_t é definido como

$$Q_t = \frac{o_1 \times wcCount_{bs} + o_2 \times eCount_{bs}}{exp_{wcCount_{bs}} \times exp_{eCount_{bs}}} \tag{4.20}$$

As mesmas constantes o_1 e o_2 aparecem aqui, balanceando quanto de cada estimativa os projetistas desejam considerar. O termo $wcCount_{bs}$ novamente representa o pior caso da quantidade de vezes que a instrução bs será executada, enquanto o novo termo $eCount_{bs}$ representa a quantidade esperada de vezes. Ambos os termos estão sujeitos aos erros $exp_{wcCount_{bs}}$ e $exp_{eCount_{bs}}$

Em tarefas que não são de tempo real, a quantidade de ciclos necessários para executar cada tarefa n é dada por

$$cycles_n = \sum_{it \in IT} \frac{instr_n(it) \times \phi}{IPC(it) \times exp_n \times exp_\phi} \tag{4.21}$$

onde novamente $IPC(it)$ é o número de instruções de tipo it que o microprocessador consegue executar por ciclo, $instr_n(it)$ representa a quantidade de instruções de tipo it executadas na tarefa n e exp_n representa um erro referente à falta de experiência dos projetistas a respeito da tarefa n. ϕ novamente representa a estimativa de valor esperado para o efeito causado por laços de repetição, falhas de cache e falhas de predição de desvios condicionais. Esse valor esperado também é afetado por um erro exp_ϕ referente à falta de experiência da equipe.

Para cada tipo it, a quantidade de intruções daquele tipo executadas em uma tarefa n é

$$instr_n(it) = \frac{eIns(it)}{dev_n} \tag{4.22}$$

onde $eIns(it)$ representa a quantidade esperada de instruções do tipo it a serem executadas na tarefa n. Nenhum erro é considerado para o valor de $eIns(it)$, porém dev_n novamente é usado e representa o erro referente ao estágio de desenvolvimento da tarefa n.

O termo $wait_n$ é por sua vez definido como

$$wait_n = \sum_{bs \in BS_n} \frac{waitTime_{bs} \times eCount_{bs}}{exp_{eCount_{bs}}} \tag{4.23}$$

O fator $eCount_{bs}$ novamente leva em consideração a quantidade esperada de vezes que a instrução bs será executada. Esse fator está novamente sujeito ao erro $exp_{eCount_{bs}}$, e também é multiplicado novamente pelo termo $waitTime_{bs}$, para cada instrução bs presente na tarefa n.

Através dessas equações pode-se considerar no modelo proposto tanto sistemas de tempo real como também sistemas que não são de tempo real, e qualquer variação entre eles. Isso é possível porque as tarefas de tempo de real são tratadas separadamente, em fatores distintos na equação. Também é possível considerar no cálculo apenas uma ou algumas das tarefas do sistema. Pode-se ter uma abordagem agressiva utilizando valores de pior caso sempre que possível, ou uma abordagem mais tolerante usando valores esperados.

4.3 UTILIZAÇÃO DE MEMÓRIA

A modelagem da utilização de memória num sistema embarcado considera aqui apenas a utilização de memória estática, não dinâmica, de forma que as informações mais relevantes de cada componente são a quantidade de código e de dados que ele tem. Dessa forma, não considera-se memória virtual, e considera-se que todo o código e os dados dos componentes estão sempre 100% na memória. Nessa modelagem, considera-se inicialmente a existência de um conjunto A com todas as instâncias de componentes de software diretamente usados pelo sistema. Cada instância a do conjunto A depende da presença de vários componentes para que ela execute corretamente. Consideramos então que cada instância a do conjunto A possui um conjunto a_D que contém uma cópia de cada componente de software que ela depende para poder executar. Define-se então o conjunto de componentes de software C_S que serão adicionados à memória como sendo

$$C_S = (\bigcup_{d \in a_D} d) \cup A \tag{4.24}$$

O uso de memória M de um sistema embarcado é então

$$M = Mcode + Mdata \tag{4.25}$$

onde $Mcode$ representa a memória utilizada pelo código dos componentes e $Mdata$ representa o trecho de dados utilizado pelos componentes.

Para o código de todos os componentes, soma-se o trecho de código de cada um separadamente, como descrito em

$$Mcode = \sum_{cs \in C_S} \frac{Mcode(cs)}{dev_{cs}} \tag{4.26}$$

Assim como nos modelos descritos anteriormente, dev_{cs} representa o erro referente ao estágio de desenvolvimento do componente cs. $dev_{cs} = 1$ caso o estágio de desenvolvimento do componente cs for 100%.

O uso de memória $Mcode(cs)$ referente a cada componente cs é dado por

$$Mcode(cs) = \sum_{in \in instr_{cs}} sizeb(in) \tag{4.27}$$

sendo que $instr_{cs}$ é o conjunto de instruções de um componente de software cs. $sizeb(in)$ corresponde a quantos bytes uma instrução in ocupa.

Para o trecho de dados dos componentes em C_S, também soma-se quantos bytes cada um ocupa, como descrito em

$$Mdata = \sum_{cs \in C_S} \frac{Mdata(cs) \times copies(cs)}{dev_{cs}} \tag{4.28}$$

$Mdata(cs)$ consiste em quanto de memória ocupa o setor de dados de cada componente cs. $copies(cs)$ representa quantas instâncias do componente cs estão presentes no sistema. Mais uma vez, dev_{cs} consiste no erro referente ao estágio de desenvolvimento do componente cs.

Por fim, temos que o trecho de dados de cada componente consiste em

$$Mdata(cs) = \sum_{dt \in data_{cs}} sizeb(dt) \tag{4.29}$$

onde $data_{cs}$ é o conjunto de todos os dados do componente cs e $sizeb(dt)$ representa quantos bytes cada dado dt ocupa na memória.

Note que nessas equações de uso de memória não usamos o erro *exp*, referente à experiência dos projetistas. Isso ocorre pois a experiência dos projetistas não afeta o tamanho do trecho de código ou de dados dos componentes de software, visto que esses componentes já são adquiridos prontos, e nenhum desenvolvimento se faz necessário. Por outro lado, se o estágio de desenvolvimento desses componentes for baixo, pode ocorrer de o código ou o trecho de dados do componente aumentar conforme seu desenvolvimento avança, e por isso considera-se esse estágio de desenvolvimento nessas equações.

4.4 CUSTO FINANCEIRO

O custo a que se refere nesta seção é o custo financeiro, referente à compra de componentes. Esse tipo de custo também foi modelado através da abordagem proposta. Considera-se o custo como sendo em dólares, embora qualquer moeda possa ser usada. De fato, inclusive uma interpretação genérica e relativa pode ser dada para os custos calculados. De qualquer maneira, esse custo é simples, tendo em vista que cada componente possui basicamente apenas seu preço como variável para a equação. Porém, pode-se considerar vários outros fatores que podem influenciar na compra de componentes.

O custo P de um sistema embarcado pode ser modelado como

$$P = \sum_{hw \in C} P(hw) + \sum_{sw \in C_{sw}} P(sw) \tag{4.30}$$

sendo que C é o conjunto de componentes de hardware no sistema e C_{sw} é o conjunto de componentes de software no sistema. Separa-se então o cálculo do custo dos componentes de hardware $P(hw)$ dos de software $P(sw)$.

O custo de cada componente de hardware pode ser modelado como

$$P(hw) = \frac{V(hw)}{err(hw) \times factors(hw)} \tag{4.31}$$

sendo que $V(hw)$ é o custo em dólares do componente hw. $err(hw)$ é um produto de vários erros atribuídos ao calculo do custo e $factors(hw)$ é um produto de vários fatores que influenciam no custo de um com-

ponente. Define-se $err(hw)$ como sendo

$$err(hw) = exp_{hw} \times dev_{hw} \tag{4.32}$$

exp_{hw} é um erro referente à experiência dos projetistas a respeito de características e funcionamento do componente hw e dev_{hw} é um erro referente ao estágio de desenvolvimento do componente hw. $factors(hw)$ é definido como

$$factors(hw) = qty_{hw} \times avail_{hw} \times exp_{avail_{hw}} \times train(hw) \tag{4.33}$$

Essa equação possui três fatores que influenciam no valor do componente hw da mesma forma que os erros descritos anteriormente. qty_{hw} representa o efeito causado no valor de cada componente pela quantidade de componentes hw iguais que será comprada. Se apenas uma unidade for comprada, $qty_{hw} = 1$. No caso de mais unidades serem compradas, o valor de qty_{hw} diminui tendendo a 0, geralmente de forma não linear.

Já o fator $avail_{hw}$ representa a disponibilidade no mercado do componente hw. Um componente que possui uma disponibilidade baixa terá seu valor aumentado devido ao maior esforço necessário para trazer o componente para os desenvolvedores trabalharem com ele. $avail_{hw} = 1$ quando o componente está imediatamente disponível e $avail_{hw}$ tende a 0 quando o componente não existir. O termo $exp_{avail_{hw}}$ representa a experiência dos projetistas a respeito do valor de $avail_{hw}$. O termo $train(hw)$ é definido como

$$train(hw) = trn_{hw} \times exp_{trn_{hw}} \tag{4.34}$$

trn_{hw} representa a intensidade de um eventual treinamento com a equipe de desenvolvedores para capacitá-los a utilizar o componente. Treinamentos induzem em mais custos para o projeto, então trn_{hw} possui intervalo $\Re(0, 1]$ e $trn_{hw} = 1$ quando nenhum treinamento for necessário. Novamente, o erro $exp_{trn_{hw}}$ representa a experiência dos projetistas a respeito de trn_{hw}. Esse erro se faz necessário pois o valor de trn_{hw} é estimado pelos projetistas, e portanto está sujeito à experiência deles a respeito do componente hw em questão. O mesmo vale para o erro $exp_{avail_{hw}}$ visto anteriormente.

Por fim, o custo de cada componente de software é definido como

$$P(sw) = \frac{V(sw)}{exp_{sw} \times dev_{sw} \times train(sw)} \tag{4.35}$$

sendo novamente $V(sw)$ o valor do componente em dólares. O erro exp_{sw} ajusta o custo do componente sw de acordo com a experiência da equipe de projetistas naquele componente e o erro dev_{sw} ajusta o valor do componente de acordo com o estágio de desenvolvimento dele. Um estágio mais atrasado induz em mais custos para o projeto. Por fim, novamente $train(sw)$ é usado, indicando a necessidade e a intensidade de um eventual treinamento da equipe. No cálculo do custo financeiro dos componentes de software não está se considerando que a compra de várias cópias ou licenças do componente influenciará no valor unitário dos componentes, embora isso poderia ter sido modelado também.

4.5 INTEGRANDO EQUAÇÕES

As equações criadas e descritas nas seções anteriores referem-se aos principais requisitos não funcionais de um sistema embarcado: consumo de energia, uso de memória, desempenho, e o próprio custo financeiro do sistema. Nessas equações foram considerados também vários fatores externos, como experiência da equipe de projetistas, disponibilidade dos componentes no mercado, necessidade da realização de treinamentos com a equipe, estágio de desenvolvimento dos componentes, entre outros. Fatores que estão sujeitos ao julgamento humano foram condicionados à variáveis de erro que atenuam as estimativas.

O fato de todos esses requisitos terem sido modelados pela mesma técnica permite que todos eles sejam tratados conjuntamente por alguma ferramenta que realize DSE ou que simplesmente solucione as equações, o que ajuda bastante a inter-relacionar os requisitos. Usar uma modelagem matemática para os requisitos também permite que os modelos sejam extensíveis. Novos requisitos podem ser modelados sem dificuldades, e as equações existentes podem ser aperfeiçoadas. A utilização da linguagem matemática também permite uma fácil compreensão dos modelos por parte de ferramentas computacionais e também por parte dos usuários.

A etapa de DSE, como foi discutido anteriormente, parte de uma especificação do sistema, onde todas as funcionalidades, suas peculiaridades e os requisitos não funcionais são descritos. A partir dessa especificação, inicia-se o mapeamento de todas as funcionalidades em tarefas do sistema, que podem ser implementadas tanto em hardware quanto em software, ou até mesmo utilizando componentes mecânicos. Há inúmeras possibilidades de mapeamentos para as funcionalidades desejadas. A DSE tem como objetivo identificar as melhores soluções

de mapeamento das funcionalidades ao mesmo tempo que cumpre todos os requisitos não funcionais estabelecidos. Isso é realizado através de atenuações das constantes das equações e variações nos conjuntos de componentes a serem executados. A troca de informações entre equações ajuda nesse processo.

As equações de desempenho utilizam dados como frequência do microprocessador, f, que afeta diretamente o consumo de energia. Nas equações de uso de memória, um fator importante é a quantidade de bytes necessária para armazenar uma instrução in, representada por $sizeb(in)$. Essa variável influencia o tempo de execução de cada instrução, pois faz parte do cálculo de $IPC(it)$ para cada tipo de instrução it presente nas equações de desempenho. Além disso, as equações de custo financeiro estão diretamente relacionadas a todas as outras, visto que cada componente possui um preço, descrito aqui em dólares, que geralmente aumenta conforme também aumenta a capacidade do componente de cumprir os requisitos do sistema. Por fim, adicionamos fatores como a experiência dos projetistas e o estágio de desenvolvimento dos artefatos do sistema em todas as equações, fazendo com que incertezas a respeito de valores estimados por projetistas e valores flexíveis de componentes inacabados sejam todos tratados.

Esses são apenas alguns exemplos de relações entre diferentes equações. É possível, porém, integrar todas as equações em uma única equação geral, usando como base a equação 3.3 (LEE et al., 2008). Essa equação geral poderia ser definida como

$$f(E,T,M,P) = \alpha\frac{E}{\bar{E}} + \beta\frac{T}{\bar{T}} + \gamma\frac{M}{\bar{M}} + \delta\frac{P}{\bar{P}} \tag{4.36}$$

onde E representa o consumo de energia do sistema, T representa o resultado da equação de desempenho, M representa o uso de memória do sistema todo e P representa o custo financeiro do sistema. As variáveis $\bar{E}$, $\bar{T}$, $\bar{M}$ e $\bar{P}$ representam o limite superior das equações, definido pelos projetistas. Os coeficientes α, β, γ e δ controlam a importância de cada requisito na hora de escolher a melhor solução para o sistema. Essa é uma forma simples de integrar as equações em uma só.

Ao se tentar resolver essa equação, porém, percebe-se que ela não possui apenas uma única solução ótima, visto que os fatores são conflitantes. Uma das técnicas mais conhecidas para especificar preferências entre soluções é a dominância de Pareto. Uma solução é preferível, ou seja, pertence ao conjunto de Pareto, se não há outra solução que possa melhorar pelo menos uma das equações sem piorar outra; neste caso, diz-se que essa solução é Pareto-dominante. Através da obtenção de

um conjunto de Pareto, têm-se um conjunto de boas soluções para o sistema. Desse conjunto, os projetistas devem escolher arbitrariamente uma das soluções (ZITZLER; THIELE, 1999; CANCIAN, 2011).

4.6 CONSIDERAÇÕES FINAIS E LIMITAÇÕES DO MODELO

Nesse capítulo foi apresentada a proposta dessa dissertação. O modelo proposto consiste em um conjunto de quatro grandes equações que buscam representar quatro dos principais custos de sistemas embarcados: consumo de energia, desempenho, uso de memória e custo financeiro. Para cada custo, foram consideradas então características que influenciam nos resultados do custo. Essas características podem tanto ser inerentes aos componentes do sistema como também às variáveis do projeto.

Embora considere uma quantidade bem limitada de custos, o modelo proposto pode ser estendido. Com base num repositório de componentes maior ou igual ao utilizado atualmente nas equações, novas equações também descritas em linguagem matemática podem ser criadas, de forma a considerar novos custos. Além disso, as próprias equações existentes também podem ser estendidas. Por exemplo, novas variáveis podem ser adicionadas ao cálculo do consumo de energia de cada conector ligando um componente aos outros, refinando dessa forma o cálculo do consumo de energia do sistema como um todo.

O tratamento de incertezas foi utilizado nos modelos através das variáveis de erros. Existe incerteza inerente a vários termos usados nas equações e os resultados de cada parte das equações acaba sendo influenciado por essa incerteza. A utilização de variáveis de erros tem então como objetivo considerar essas incertezas e propor uma estimativa conservadora.

Buscar a modelagem dos principais custos de um sistema embarcado visando a exploração do espaço de projeto através de uma única linguagem é um trabalho longo e detalhado, e durante esse processo algumas limitações foram impostas ao modelo, reduzindo em parte o escopo do trabalho. A exploração dessas limitações fica como um guia para possíveis trabalhos futuros.

Uma das limitações do trabalho é a deficiência do tratamento de dependências. Esse tratamento foi realizado parcialmente em relação às equações como um todo, mas não foi explorado em termos da relação entre as variáveis de uma única equação. Quando considera-se as equações como um todo, a dependência entre diferentes equações foi represen-

tada na utilização de variáveis iguais em algumas delas. Por exemplo, a frequência do processador faz parte da equação de consumo de energia e também da equação de desempenho, como visto na seção anterior. Isso permite que as equações dependam uma da outra. Em termos das variáveis de cada equação, não foi realizado um tratamento das dependências, embora tenha se buscado uma modelagem onde essa deficiência não prejudique muito os modelos. Por exemplo, as equações de desempenho tratam cada tarefa separadamente, e consideram o tempo de execução delas como se executassem em um ambiente com apenas um núcleo de processamento. Isso em si constitui uma limitação do trabalho, mas ao mesmo tempo reduz as dependências entre as tarefas.

Outra limitação dos modelos é que eles não consideram características específicas de componentes mecânicos, o que prejudica a modelagem de custos de sistemas que utilizem esse tipo de componente. Uma grande limitação imposta às equações de desempenho é o fato de elas levarem em consideração apenas a implementação em software das tarefas. Dessa forma, não se mede o desempenho de tarefas com partes ou totalmente implementadas em hardware. Já em termos de uso de memória, como foi dito nas seções anteriores, é considerada apenas a utilização de memória estática, não dinâmica, de forma que as informações mais relevantes de cada componente são a quantidade de código e de dados que ele tem. Dessa forma, não foram considerados outros tipos de memória, como memórias de vídeo, cache de disco ou de processador e memória virtual. Foi considerado que todo o código e os dados dos componentes estão sempre 100% na memória.

5 ESTUDO DE CASO

Na proposta de modelagem de um sistema embarcado apresentada no capítulo anterior, cada requisito é modelado matematicamente e utiliza dados da especificação do sistema para permitir uma eficaz exploração do espaço de projeto. Porém, utilizar essas equações de fato numa exploração está fora do escopo do trabalho. A contribuição desse trabalho são as equações, e a avaliação da proposta consiste em mostrar em um estudo de caso que é possível realizar-se toda a modelagem dos custos de um sistema embarcado com base apenas nas equações propostas. Será mostrado também que com base nessas equações será possível futuramente realizar uma extensa exploração de espaço de projeto. Dessa forma, será apresentado neste capítulo um simples estudo de caso onde foi possível utilizar todas as equações de forma didática.

5.1 ESPECIFICAÇÃO DO SISTEMA

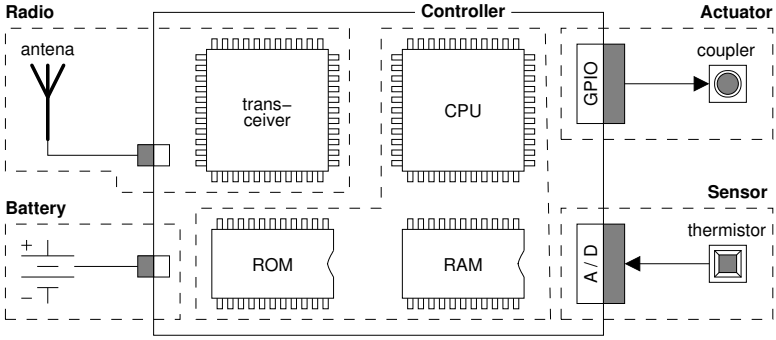

Figura 6: Diagrama em blocos do sistema a ser projetado (FRÖHLICH, 2011).

A primeira etapa, essencial para as equações serem aplicadas, é definir a especificação inicial do sistema. Utilizando como base um estudo de caso desenvolvido por Fröhlich, vamos apresentar nessa seção todas as características do sistema e como modelamos elas (FRÖHLICH, 2011).

A aplicação a ser desenvolvida implementa um sistema de mo-

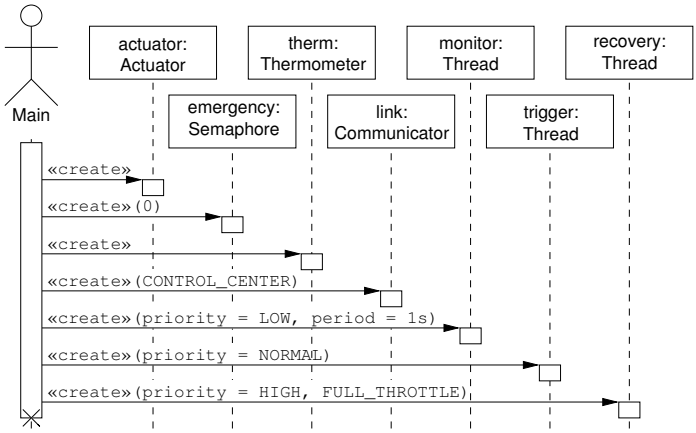

Figura 7: Diagrama de sequência da tarefa Main (FRÖHLICH, 2011).

nitoramento remoto de temperatura, capaz de sentir a temperatura do ambiente, reportá-la para uma central de controle, e reagir ativando um cooler externo quando um certo limite for ultrapassado. A interação com a central de controle é feita via rádio. O sistema opera utilizando baterias e deve funcionar ininterruptamente por um ano. Um diagrama de blocos do sistema é mostrado na Fig. (6). A aplicação é modelada através de quatro tarefas, cujo comportamento é descrito através dos diagramas de sequência das Fig. (7), (8), (9) e (10).

A tarefa Main aloca recursos comuns e cria as threads que executarão as outras tarefas. A tarefa Monitor é responsável pelo monitoramento periódico da temperatura (a cada segundo), e também reporta a temperatura atual para a central de controle (a cada 10 segundos), e, caso o limite crítico de temperatura seja ultrapassado, ativa a thread de tratamento da emergência. Trigger é responsável pelo ativamento da thread de tratamento de emergência quando a central de controle ordenar. Recovery, por fim, é a thread de tratamento da emergência, e está encarregada de ativar um atuador capaz de restaurar a temperatura ao seu nível normal.

A organização das tarefas é garantida através da atribuição de prioridades às threads e através de um semáforo. Nos diagramas de sequência mostrados acima, ações relacionadas a energia capturadas durante o projeto são expressas por mensagens e anotações. Para ser energeticamente eficiente, o circuito por trás do termômetro deve ser

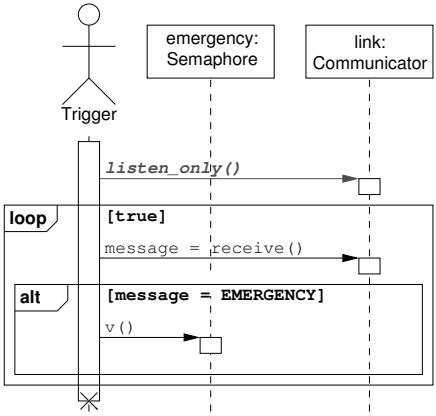

Figura 8: Diagrama de sequência da tarefa Trigger (FRÖHLICH, 2011).

mantido ativo durante todo o ciclo de uma medição, de forma a evitar repetições da fase de estabilização elétrica, que costuma representar até 98% do ciclo de uma medição. Os diagramas também mostram dicas que ajudam na gerência do consumo de energia para o componente de comunicação, que é usado em boa parte do tempo para aguardar a chegada de alguma mensagem da central de controle, e por isso pode ser configurado para um estado apenas de escuta na maior parte do tempo. Uma estratégia semelhante pode ser usada para o componente da CPU, que deve operar em frequência máxima enquanto executa a tarefa de Recovery, mas pode operar em frequências mais baixas para as outras tarefas.

Todas essas informações permitem que se tenha uma noção clara de todas as funcionalidades que o sistema deve ter e como elas devem ser implementadas. O sistema deverá usar um gerenciador automático do consumo de energia, mas ainda assim o consumo de energia é um requisito relevante pois a bateria do sistema deve durar um ano. Através dessas informações, sabemos que tipos de componentes de hardware o sistema deve ter. No caso, o sistema precisa de um processador, memórias, um rádio, um atuador, uma bateria e um sensor de temperatura. Também se faz necessária a utilização de componentes de software. No nosso estudo de caso, utilizamos o sistema operacional EPOS, o qual fornece todos os componentes de software que precisamos, entre eles um semáforo, uma implementação de threads não periódicas e periódicas, interfaces para os componentes de hardware, entre outros

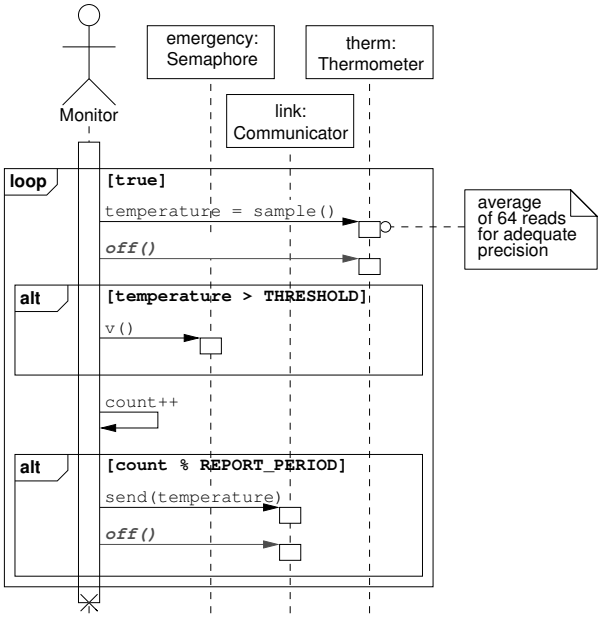

Figura 9: Diagrama de sequência da tarefa Monitor (FRÖHLICH, 2011).

componentes dos quais esses dependem (LISHA, 2012).

5.2 REPOSITÓRIO DE COMPONENTES

Com base nessa especificação funcional e num repositório de componentes diversos que cumpram essas exigências, deve-se se certificar de que todas as informações necessárias para as equações estejam disponíveis. As informações contidas no repositório podem ter sido estimadas pelos projetistas ou fornecidas pela indústria. As equações possuem diversas variáveis e muitos valores precisam ser estimados pelos projetistas (FRÖHLICH, 2011), ou obtidos experimentalmente. Para cada componente do sistema, é preciso listar todas as informações que serão relevantes nas equações. Em alguns casos esses dados não estão disponíveis e sua obtenção não seria possível dentro do tempo disponível para a pesquisa. Nesses casos, foram feitas estimativas por parte dos pesquisadores dessa dissertação.

As Tabelas (3), (4), (5) e (6) mostram todas as informações

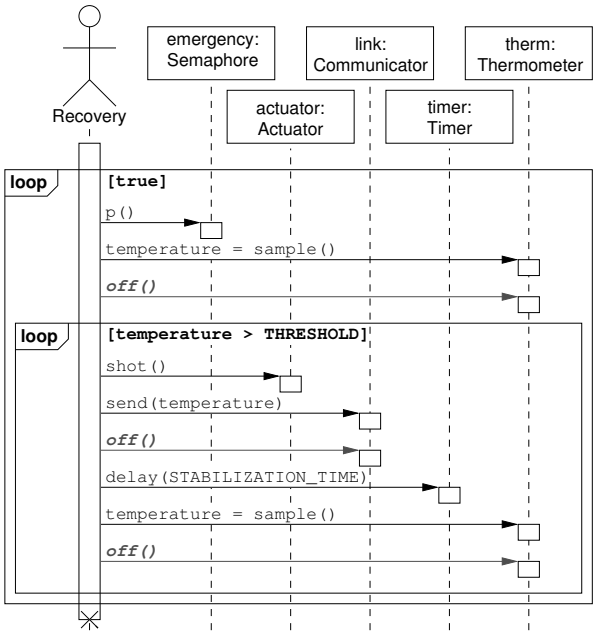

Figura 10: Diagrama de sequência da tarefa Recovery (FRÖHLICH, 2011).

contidas no repositório a respeito dos componentes de hardware e software do EposMote. A mesma tabela pode existir para todas as outras opções de motes consideradas. Nessas tabelas, cada coluna mostra as informações referentes a cada componente, com exceção dos erros que afetam todos os componentes, onde o erro em questão é repetido em cada coluna.

Nas Tabelas (3) e (5) são mostradas as seguintes informações a respeito de cada componente. Os termos iniciados em *EXP* e *DEV* seguem a definição apresentada nas equações 4.7 e 4.8. Nas primeiras linhas são mostradas as informações relativas a tempo de execução.

waitTime (Recovery) - Usado na equação 4.15, esse termo representa o tempo em milissegundos que a tarefa Recovery fica parada por ação de algum *timer*.

wcCount (waitTime) - Também usado na equação 4.15, esse termo representa o pior caso da quantidade de vezes que a tarefa Recovery fica esperando pelo tempo waitTime durante uma execução com-

Tabela 3: Informações contidas no repositório para o controlador e o rádio do EposMote em termos de desempenho e uso de memória.

Componentes	**Controlador**	**Rádio**
waitTime (Recovery)	100	0
wcCount (waitTime)	1	0
EXP (wcCount)	0,8	0,8
count (Recovery)	1	1
EXP (Recovery)	0,75	0,75
instr (Recovery)	39,6552	11,33
count (Trigger)	1	1
EXP (Trigger)	0,6	0,6
DEV (Trigger)	1	1
instr (Trigger)	14,9917	14,9917
count (Main)	1	1
EXP (Main)	0,85	0,85
DEV (Main)	0,99	0,99
instr (Main)	88,3838	29,4613
count (Monitor)	10	10
EXP (Monitor)	0,7	0,7
DEV (Monitor)	0,999	0,999
instr (Monitor)	42,042	21,021
μ	0,7	0,7
EXP (μ)	0,8	0,8
θ	0,8	0,8
EXP (θ)	0,7	0,7
o1	0,6	0,6
o2	0,4	0,4
f	26	0
Actuator	0	0
Semaphore	69	69
Thermometer	0	0
Alarm e Timer	597	171
Communicator	0	1820
Thread	394	394
Outros	757	745
Tamanho instrução	4	4

Tabela 4: Informações contidas no repositório para o controlador e o rádio do EposMote em termos de custo financeiro e consumo de energia.

Compon.	**Controlador**			**Rádio**			
Preço (sw)	45			50			
EXP (sw)	0,9254			0,9254			
DEV (sw)	0,9683			0,9683			
trn sw	0,7231			0,7231			
EXP trn sw	0,95			0,95			
Preço (hw)	21,85			21,85			
EXP (c)	0,92			0,99			
DEV (c)	0,95			1			
qty	0,1			0,1			
avail	0,9			0,9			
EXP (avail)	0,75			0,75			
trn hw	0,8			0,8			
EXP trn hw	0,9			0,9			
Interfaces	Rádio	Sensr	Atuadr	Controle		Out	
comm	0,001	0,02	0,02	0,001		0,004	
EXP (j)	0,99	0,99	0,99	0,99		0,99	
DEV (j)	1	0,98	0,95	1		0,98	
count (j)	363	3600	3	363		363	
EXP (count)	0,9	0,9	0,9	0,9		0,9	
Estados	stndby	min	max	off	listen	R	T
curr	5,1	950	7300	1	97	12k	12k
dutycycle	0,987	0,013	0,009	0,002	0,98	0,01	0,02
EXP (c,s)	0,85	0,95	0,95	0,99	0,95	0,95	0,95
DEV (c,s)	1	1	1	1	1	1	1

pleta da tarefa.

EXP (wcCount) - Último termo usado na equação 4.15, representa a experiência dos projetistas no valor atribuído a wcCount anteriormente.

count (nome da tarefa) - 4.10, 4.11 e 4.12, esse termo representa a quantidade de vezes que cada tarefa é executada no estudo de caso realizado.

EXP (nome da tarefa) e *DEV (nome-da-tarefa)* - Usados nas equações 4.13, 4.16, 4.18, 4.19, 4.21 e 4.22, esses termos representam a

Tabela 5: Informações contidas no repositório para o sensor e o atuador do EposMote em termos de desempenho e uso de memória.

Componentes	**Sensor**	**Atuador**
waitTime (Recovery)	0	0
wcCount (waitTime)	0	0
EXP (wcCount)	0,8	0,8
count (Recovery)	1	1
EXP (Recovery)	0,75	0,75
instr (Recovery)	11,33	5,665
count (Trigger)	0	0
EXP (Trigger)	0	0
DEV (Trigger)	0	0
instr (Trigger)	0	0
count (Main)	1	1
EXP (Main)	0,85	0,85
DEV (Main)	0,99	0,99
instr (Main)	29,4613	29,4613
count (Monitor)	10	0
EXP (Monitor)	0,7	0
DEV (Monitor)	0,999	0
instr (Monitor)	21,021	0
μ	0,7	0,7
EXP (μ)	0,8	0,8
θ	0,8	0,8
EXP (θ)	0,7	0,7
o1	0	0
o2	0	0
f	0	0
Actuator	0	465
Semaphore	69	68
Thermometer	356	0
Alarm e Timer	171	85
Communicator	0	0
Thread	394	394
Outros	742	742
Tamanho instrução	4	4

Tabela 6: Informações contidas no repositório para o sensor e o atuador do EposMote em termos de custo financeiro e consumo de energia.

Componentes	**Sensor**			**Aduator**	
Preço (sw)	134			130	
EXP (sw)	0,9254			0,9254	
DEV (sw)	0,9683			0,9683	
trn sw	0,7231			0,7231	
EXP trn sw	0,95			0,95	
Preço (hw)	21,85			21,85	
EXP (c)	0,8			0,9	
DEV (c)	0,98			0,95	
qty	0,1			0,1	
avail	0,9			0,9	
EXP (avail)	0,75			0,75	
trn hw	0,8			0,8	
EXP trn hw	0,9			0,9	
Interfaces	Controller			Controller	
comm	0,02			0,02	
EXP (j)	0,99			0,99	
DEV (j)	0,98			0,95	
count (j)	3600			3	
EXP (count)	0,9			0,9	
Estados	off	At 25 C	At 40 C	on	off
curr	1	850	1700	2000	0
dutycycle	0,9997	0,0002	0,0001	0,0001	0,9999
EXP (c,s)	0,95	0,8	0,8	0,8	0,95
DEV (c,s)	0,98	0,98	0,98	0,95	0,95

experiência dos projetistas e o estágio de desenvolvimento de cada uma das tarefas do repositório.

instr (nome da tarefa) - De forma a simplificar as tabelas, esse termo já traz o resultado das equações 4.14, 4.18 e 4.22, e, dependendo da tarefa, representa o pior caso ou o caso esperado da quantidade de instruções executadas na tarefa, já sob efeito dos erros atribuídos nessas equações.

μ e θ - Usadas nas equações 4.13, 4.17 e 4.21, essas constantes representam o pior caso e o caso esperado do efeito causado por laços de repetição, falhas de cache e de predição de desvios condicionais no

número de instruções executadas.

EXP (μ) e *EXP (θ)* - Usadas em conjunto com as constantes em questão, esses termos representam a experiência dos projetistas na atribuição de um valor para as constantes μ e θ.

o1 e *o2* - Usadas na equação 4.17, essas constantes são responsáveis por balancear o efeito das constantes μ e θ nas estimativas de tempo de execução.

f - Usado nas equações 4.10, 4.11 e 4.12, esse termo representa a frequência em MHz na qual funciona o controlador do nodo sensor.

A seguir são mostrados os componentes de software do sistema, do *Actuator* a *Thread*, e sua utilização de memória relativa a cada componente de hardware. As tabelas já simplificam a utilização de memória e contabilizam juntos os trechos de código e dados dos componentes, considerando também a quantidade de instâncias de cada um. Dados dos componentes *Alarm* e *Timer* foram colocados juntos, e os dados referentes aos outros componentes menores, incluindo alguns do sistema operacional, foram colocados em *Outros*. Esses valores apenas mostram a quantidade de instruções e dados no componente. Para calcular o verdadeiro tamanho do componente, deve-se multiplicar esses valores pelo valor apresentado em *Tamanho instrução*, que representa o tamanho em *bytes* de cada instrução ou unidade de dado.

Nas Tabelas (4) e (6) os termos *EXP* e *DEV* se repetem, e também aparecem variáveis de custo financeiro, primeiro de software, representadas pela sigla *sw*, e depois de hardware, representadas pela sigla *hw*.

Preço (sw) e *Preço (hw)* - Usados nas equações 4.31 e 4.35, esses termos representam respectivamente o preço dos componentes de software e de hardware do sistema, referentes a cada componente de hardware. Esses valores estão representados aqui em dólar americano (US$), embora qualquer moeda pudesse ser usada.

EXP (sw) e *EXP (c)* - Usados nas equações 4.32 e 4.35, esses termos representam a experiência dos projetistas nos componentes de software e em cada componente *c* de hardware. A experiência nos componentes de software é mostrada aqui com um valor único, para simplificar a tabela, embora cada componente de software possua sua própria experiência dos projetistas.

DEV (sw) e *DEV (c)* - Também usados nas equações 4.32 e 4.35, esses termos representam o estágio de desenvolvimento dos componentes de software e de cada componente *c* de hardware, da mesma forma que os termos de experiência dos projetistas.

trn sw e *trn hw* - Usados na equação 4.34 e de forma implícita

na equação 4.35, esses termos representam a intensidade de um eventual treinamento com a equipe de desenvolvedores para capacitá-los a utilizar os componentes de software e cada componente de hardware.

EXP (trn sw) e *EXP (trn hw)* - Usados em conjunto com os termos acima, esses termos representam a experiência dos projetistas na escolha de um valor para representar o treinamento com a equipe de desenvolvedores.

qty - Usado na equação 4.33, representa o efeito causado pela quantidade de componentes que serão comprados no preço de cada componente.

avail e *EXP (avail)* - Também usados na equação 4.33, representam o efeito causado pela disponibilidade no mercado no valor a ser pago por cada componente, e a experiência dos projetistas na atribuição de um valor para esse efeito.

Nas linhas finais das tabelas são mostradas as informações referentes ao consumo de energia, que estão separados em consumo das interfaces de comunicação e consumo dos componentes em si.

comm - Usado na equação 4.6, representa o consumo em μA de cada acesso a cada uma das interfaces de comunicação do componente.

EXP (j) e *DEV (j)* - Usados também na equação 4.6, esses termos representam a experiência dos projetistas e também o estágio de desenvolvimento das interfaces de comunicação existentes, representadas por *j*.

count (interface) e *EXP (count)* - Usados também na equação 4.6, representam a quantidade de acessos realizados a cada uma das interfaces j de comunicação e a experiência dos projetistas na especificação desses valores.

curr - Usado na equação 4.5, representa a corrente em μA consumida por cada componente em cada um de seus estados de funcionamento.

dutycycle - Também usado na equação 4.5, representa o *duty cycle* de cada componente em cada estado.

EXP (c,s) e *DEV (c,s)* - Usados na equação 4.5, esses termos representam a experiência dos projetistas e o estágio de desenvolvimento de cada componente c em termos dos seus estados s de funcionamento.

Esses dados descritos nas tabelas acima foram usados nas equações, de forma a obter um valor que representasse uma configuração do sistema para cada um dos requisitos especificados. No caso do consumo de energia, o repositório de componentes utilizado possui um grupo de motes capazes de executar a aplicação desejada. Como visto na Tab. (3), para o cálculo do consumo de energia, os componentes de cada

mote foram considerados separadamente. Para cada componente, foram utilizadas informações como experiência dos projetistas, estágio de desenvolvimento, corrente, *duty cycle* e interfaces de comunicação. Todas essas informações foram reunidas e estruturadas em tabelas. Além do Epos Mote, também foram considerados no estudo os motes Mica-2, Mica-Z, Telos e ZigBit (KRÄMER; GERALDY, 2006; POLASTRE; SZEWCZYK; CULLER, 2005; LISHA, 2012).

Tabela 7: Tarefas e suas características relevantes para o desempenho.

Nome	**exp**	**WC instr**	**waitTime**	**exp(wcCount)**
Recovery	0.750000025	67.98015	100	0.8
	exp	**dev**	**E instr**	
Trigger	0.600004	1	29.98338	
	exp	**dev**	**E instr**	
Main	0.850000015	0.99	176.76768	
Monitor	0.70000003	0.999	84.0841	

Para o requisito de desempenho, considerou-se todas as tarefas do sistema e dividiu-se essas tarefas em tarefas de tempo real crítico, tempo real não crítico e não tempo real. Nesse estudo de caso estão sendo consideradas as 4 tarefas descritas acima: Main, Monitor, Trigger e Recovery. Para cada uma delas, foram listadas na Tab. (7) as informações referentes à quantidade de instruções executadas, *delays* ocasionados por software, além de erros referentes à experiência dos projetistas e ao estágio de desenvolvimento das tarefas.

Na Tabela (7), *exp* e *dev* representam erros baseados em EXP e DEV, que por sua vez representam a experiência dos projetistas e o estágio de desenvolvimento das tarefas do sistema. *WC instr* e *E instr* representam o pior caso e o caso esperado da quantidade de instruções executadas em uma execução completa da tarefa. $waitTime$ é o tempo que a tarefa deve ficar parada esperando por causa de algum *timer* e *exp(wcCount)* é um erro baseado no termo *EXP (wcCount)* mostrado nas Tab. (3) e (5), e representa o erro causado pela falta de experiência dos projetistas na atribuição de um valor a *wcCount*, que por si só representa o pior caso da quantidade de vezes que a tarefa ficará esperando o tempo $waitTime$.

Considerou-se nesse estudo de caso a execução 10 vezes da tarefa Monitor e apenas uma vez das demais tarefas. A tarefa Recovery é considerada de tempo real crítico e possui um *delay* de 100ms, em-

bora no trabalho de Fröhlich tenha-se utilizado 1000ms, ou 1000000μs (FRÖHLICH, 2011). Foi utilizado um valor menor nesse estudo de forma a não deixar o tempo de execução tão dependente desse *delay*.

Considerou-se um valor de *wcCount* igual a 1. *exp(wcCount)* representa o valor $exp_{wcCount_{bs}}$ da equação 4.15. Ao contrário do trabalho apresentado por Fröhlich, a tarefa Trigger é considerada neste trabalho como sendo de tempo real não crítico, de forma a possibilitar uma utilização completa das equações de desempenho. As tarefas Main e Monitor foram consideradas como não sendo de tempo real. Ao invés de mostrar a quantidade de instruções executadas de cada tipo e os erros referentes a esses valores, a Tabela (7) já resume esses valores nos valores *WC instr* e *E instr*, assim com fez as Tab. (3) e (5).

Embora não mostrados aqui nem nas tabelas anteriores por questões de espaço, o repositório completo dessas informações também possui a quantidade de instruções por ciclo para cada tipo de instrução.

Tabela 8: Componentes de software e suas características relevantes para o uso de memória.

Componentes	**dev**	**Instruções**	**Dados**	**copies**
Actuator	0,8500000015	463	2	1
Semaphore	0,9999	275	0	1
Thermometer	0,8500000015	355	1	1
Alarm	0,9900000001	550	1	1
Scheduler	1	1088	2	1
System	0,99	1000	10	1
Communicator	0,9500000005	1807	13	1
Thread	0,9999	1572	1	4
MMU	1	220	1	1
Timer	0,9900000001	473	0	1
Heap	1	102	0	1
String	1	150	0	1
Malloc	1	415	0	1

Em termos de utilização de memória, o sistema operacional EPOS foi desmembrado em termos dos componentes de software usados na aplicação, como Semaphore, Alarm, Scheduler, Thread, MMU, Timer, Heap, entre outros. Para cada componente, foram descritas na Tab. (8) quantas instruções ele possui e quantos dados possui cada instância dele. Embora ferramentas modernas consigam facilmente determinar o uso de memória de componentes, nas equações propostas esse uso é cal-

culado, de forma a manter as equações mais abrangentes e profundas. Será considerado nesse estudo de caso que cada instrução ou dado possui 4 bytes de comprimento, então um componente com 10 instruções tem 40 bytes de tamanho (MARCONDES, 2009; MACHADO; FRÖHLICH, 2010).

A Tabela (8) mostra os dados com um rigor maior de detalhes do que o apresentado nas Tab. (3) e (5). Para cada componente, *dev* representa um erro relacionado ao estágio de desenvolvimento (DEV) de cada componente e é usado nas equações 4.26 e 4.28. *copies* representa a quantidade de instâncias de cada componente no sistema.

Tabela 9: Componentes de hardware e suas características relevantes para o custo financeiro.

Componentes	**Preço**	**exp**	**dev**	**qty**	**avail**	**trn**
Controle Mica-Z	30	0,935	1	0,1	0,8	0,8
Rádio Mica-Z	30	0,92	1	0,1	0,8	0,8
Sensor Mica-Z	30	0,7525	1	0,1	0,8	0,8
Atuador Mica-Z	30	0,86	0,9001	0,1	0,8	0,8
Controle Telos	28,18	0,635	0,9501	0,1	0,5	0,4
Rádio Telos	28,18	0,6	0,9501	0,1	0,5	0,4
Sensor Telos	28,18	0,2008	0,9001	0,1	0,5	0,4
Atuador Telos	28,18	0,37	0,8002	0,1	0,5	0,4
Controle Epos	21,85	0,96	0,9501	0,1	0,9	0,8
Rádio Epos	21,85	0,992	1	0,1	0,9	0,8
Sensor Epos	21,85	0,802	0,9802	0,1	0,9	0,8
Atuador Epos	21,85	0,94	0,9502	0,1	0,9	0,8

Por fim, considerou-se tanto os componentes de hardware como também os de software para o cálculo do custo financeiro do sistema. Para cada componente de hardware, foram listados o preço dele e também dados referentes à: quantidade de unidades do componente que serão compradas; a disponibilidade do componente no mercado; a necessidade de se realizar treinamento com a equipe de projetistas; a experiência desses projetistas na utilização do componente; e também o estágio de desenvolvimento do componente, assim como nas Tab. (4) e (6). Todos esses dados estão reunidos na Tab. (9), onde além do EposMote também apresentamos dados de 2 outros *motes*. Embora os componentes dos *motes* formem uma unidade de *mote*, o preço dos *motes* aparece dividido entre os componentes na Tab. (9).

O preço aparece na Tab. (9) em US$. *exp* e *dev* aparecem na

equação 4.32 e representam erros atribuídos ao preço dos componentes por fatores como experiência dos projetistas e estágio de desenvolvimento dos componentes. *qty*, *avail* e *trn* representam respectivamente fatores de quantidade de unidades compradas, disponibilidade do componente no mercado e necessidade de treinamento com a equipe de projetistas, e todos esses fatores também influenciam no valor final de cada componente. Os valores *exp(avail)* e *exp(trn)* representam erros atribuídos ao valor de cada componente relacionados à deficiência das estimativas de valores a *avail* e *trn*.

Tabela 10: Componentes de software e suas características relevantes para o custo financeiro.

Componentes	**Preço**	**exp**	**dev**	**trn**	**exp(trn)**
Actuator	100	0,70003	0,900000001	0,2	0,955
Semaphore	10	0,99000001	0,999	0,8	0,955
Thermometer	100	0,70003	0,8500000015	0,2	0,955
Alarm	5	0,8500015	0,900000001	0,8	0,955
Scheduler	10	1	1	0,95	0,955
System	1	1	0,9500000001	0,8	0,955
Communicator	10	0,900001	0,9900000001	0,5	0,955
Thread	100	0,99000001	0,9999	0,7	0,955
MMU	5	1	1	0,95	0,955
Timer	5	0,900001	0,999	0,8	0,955
Heap	5	1	1	0,95	0,955
String	5	1	1	0,95	0,955
Malloc	5	1	1	0,8	0,955

Para os componentes de software, foram listadas as mesmas informações, com exceção da quantidade de unidades que serão compradas e da disponibilidade do componente no mercado. Todas essas informações estão descritas na Tab. (10), com um rigor de detalhes bem maior do que o apresentado nas Tab. (4) e (6). Os componentes de sofware considerados foram os mesmos da Tab. (8). Assim como na Tab. (9), os valores *exp*, *dev*, *trn* e *exp(trn)* influenciam no custo financeiro total dos componentes de software, e representam respectivamente erros referentes à experiência dos projetistas, o estágio de desenvolvimento, a necessidade de treinamento com a equipe e o erro na atribuição de valores a *trn*.

Tabela 11: Cálculo de $logic_c$ da equação 4.5.

Componentes	**Estados**	**curr * dutycycle**	**exp * dev**	**logic**
Controller Mica-Z				
	standby	7.8952	0.95	8.3107
	min	50.44	0.95	53.0947
	max	0.865	0.95	0.9105
Radio Mica-Z				
	off	0.002	0.96	0.0021
	listen	108.0469	0.92	117.4423
	receive	1.105	0.92	1.2011
	transmit	175.875	0.92	191.1685
Sensor Mica-Z				
	off	0.9997	0.9505	1.05176
	At 25 C	0.16	0.802	0.1995
	At 40 C	0.18	0.802	0.2244
Actuator Mica-Z				
	on	0.25	0.8170	0.306
	off	0	0.9167	0
Controller EposMote				
	standby	5.03319	0.925	5.4413
	min	12.35	0.975	12.6667
	max	0.73	0.975	0.7487
Radio EposMote				
	off	0.002	0.992	0.002
	listen	94.8466	0.96	98.7985
	receive	1.17	0.96	1.2187
	transmit	235.17	0.96	244.9687
Sensor EposMote				
	off	0.9997	0.9315	1.07322
	At 25 C	0.17	0.786	0.2163
	At 40 C	0.17	0.786	0.2163
Actuator EposMote				
	on	0.2	0.8360	0.2392
	off	0	0.9215	0

Para aplicar as equações, foram utilizadas tabelas simples, de

forma a ilustrar a facilidade de aplicar o modelo proposto. Primeiramente, a Tabela (11) mostra o cálculo de $logic_c$ da equação 4.5, que representa o consumo de energia de cada componente c quando executa suas funcionalidades. A tabela mostra o resultado das multiplicações de $curr \times dutycycle$ e $exp \times dev$, e o resultado final da equação, na coluna *logic*. A última coluna mostra o resultado da equação para cada componente.

Tabela 12: Cálculo de $cons_c$ da equação 4.4.

Componentes	**logic**	**comm**	**cons**
Controller Mica-Z	62.316	211.5277	273.8437
Radio Mica-Z	309.8139	50.8028	360.6167
Sensor Mica-Z	1.4757	201.2070	202.6827
Actuator Mica-Z	0.306	0.1813	0.4873
Controller Telos	35.0862	326.7203	361.8064
Radio Telos	856.546	67.1057	923.6517
Sensor Telos	2.0577	313.6194	315.6771
Actuator Telos	0.4646	0.2613	0.726
Controller EposMote	18.8567	82.5185	101.3752
Radio EposMote	344.9881	2.0615	347.0496
Sensor EposMote	1.5058	82.0426	83.5484
Actuator EposMote	0.2392	0.07053	0.3098

A Tabela (12) mostra o cálculo final do consumo de energia para cada componente, considerando o valor de $logic_c$ e $comm_c$ calculados em cada caso. O resultado da equação aplicada para cada componente, apresentado na coluna *cons* é obtido somando-se os valores das colunas *logic* e *comm*. *logic* consiste no consumo de energia de cada componente, como calculado na equação 4.5, e *comm* é o consumo de energia das interfaces de comunicação de cada componente, como calculado na equação 4.6.

Para a equação de desempenho, foi calculada a quantidade de ciclos necessária para executar cada tarefa, através das equações 4.13, 4.16 e 4.21. Os dados independentes de arquitetura necessários para calcular o tempo de execução de cada tarefa estão na Tab. (13). Os valores de *cycles* são calculados nas equações 4.13, 4.16 e 4.21, e representam a quantidade de ciclos que são necessários para executar cada tarefa. *wait* é calculado nas equações 4.15, 4.19 e 4.23 e representa o atraso no tempo de execução das tarefas ocasionado por *timers* no código. *count* representa a quantidade de vezes que cada tarefa é

Tabela 13: Variáveis usadas no cálculo do tempo de execução de cada tarefa, como apresentado nas equações 4.10, 4.11 e 4.12, com foco na implementação no EposMote.

Tarefa	**cycles**	**wait**	**count**
Recovery	74.1028833775	124.9999998438	1
Trigger	59.3737203356	0	1
Main	215.0503970117	0	1
Monitor	11.6156251529	0	10

executada no estudo de caso em questão.

Tabela 14: Tempo de execução de cada tarefa em diferentes *motes*.

time	**Mica-Z**	**Telos**	**EposMote**
Recovery	129.6314300548	134.2628602659	127.8501107429
Trigger	3.710857521	7.421715042	2.2836046283
Main	13.4406498132	26.8812996265	8.2711691158
Monitor	7.2597657206	14.5195314411	4.4675481357

Para calcular o tempo de execução de cada tarefa, é preciso especificar um *mote* do nosso repositório de componentes de hardware, pois a equação de desempenho utiliza dados específicos de cada *mote*. Usando como base os mesmos *motes* vistos até agora neste capítulo, chegou-se aos tempos mostrados na Tab. (14). Esses tempos de execução levam em consideração a frequência de cada processador e os dados mostrados na Tab. (13).

Para o requisito de uso de memória, foram calculadas as equações 4.27 e 4.29. Os resultados estão na Tab. (15). *Mcode* representa o resultado da equação 4.27 para cada componente de software do sistema, que leva em consideração o tamanho do trecho de instruções de cada componente e o erro referente ao estágio de desenvolvimento do componente. *Mdata* representa o resultado da equação 4.29 para cada componente, que leva em consideração os dados do componente e quantos bytes cada dado ocupa.

Já para o requisito de custo financeiro, os cálculos foram separados em uma tabela para componentes de hardware e outra para componentes de software. A Tabela (16) mostra os dados usados na equação 4.31, além dos já mostrados anteriormente. *err* é o resultado a equação 4.32 para cada componente e compila os erros *exp* e *dev* referentes à

Tabela 15: Cálculo do uso de memória através de código e dados dos componentes.

Componentes	**Mcode**	**Mdata**
Actuator	2178.8235255668	9.4117646893
Semaphore	1100.110011	0
Thermometer	1670.588232346	4.7058823446
Alarm	2222.2222219978	4.04040404
Scheduler	4352	8
System	4040.4040403632	40.4040404036
Communicator	7608.4210486272	54.7368420765
Thread	6288.6288628857	16.00160016
MMU	880	4
Timer	1911.1111109181	0
Heap	408	0
String	600	0
Malloc	1660	0

experiência dos projetistas e ao estágio de desenvolvimento dos componentes. *train* é o resultado da equação 4.34 e calcula a necessidade real de um treinamento com a equipe de desenvolvedores (*trn*) sob efeito do erro $exp_{trn_{hw}}$, referente à experiência dos projetistas em atribuir um valor a *trn*. $factors$ é o resultado da equação 4.33 e compila alguns fatores que também influenciam no custo financeiro de cada componente, como a quantidade de unidades compradas, a disponibilidade no mercado e também a necessidade de um treinamento com a equipe calculada anteriormente em *train*. A coluna Custo Financeiro exibe então o custo final de cada componente, como calculado na equação 4.31, já com os erros atribuídos.

Além de todos os dados já mostrados também na Tab. (10), a Tabela (17) mostra os dados usados na equação 4.35, para cálculo do custo financeiro de cada componente de software. *train* é o resultado da equação 4.34 aplicado aos componentes de software, e calcula a necessidade real de um treinamento com a equipe de desenvolvedores (*trn*) sob efeito do erro $exp_{trn_{sw}}$, referente à experiência dos projetistas em atribuir um valor a *trn*. Com base nesses valores e nos valores apresentados na Tab. (10), obtém-se os valores da coluna Custo Financeiro, resultado da aplicação da equação 4.35.

Tabela 16: Cálculo do custo financeiro dos componentes de hardware.

Componentes	**err**	**train**	**factors**	**Custo Financeiro**
Controle Mica-Z	0.935	0.692	0.0429	747.84
RÃ¡dio Mica-Z	0.92	0.692	0.0429	760.04
Sensor Mica-Z	0.7525	0.692	0.0429	929.22
Atuador Mica-Z	0.774	0.692	0.0429	903.40
Controle Telos	0.6033	0.382	0.0115	4049.02
RÃ¡dio Telos	0.57	0.382	0.0115	4285.42
Sensor Telos	0.1807	0.382	0.0115	13516.49
Atuador Telos	0.296	0.382	0.0115	8252.35
Controle Epos	0.912	0.728	0.0508	471.80
RÃ¡dio Epos	0.992	0.728	0.0508	433.77
Sensor Epos	0.786	0.728	0.0508	547.49
Atuador Epos	0.893	0.728	0.0508	481.86

5.4 RESULTADOS OBTIDOS

Nesta seção serão mostrados os resultados das equações. Como visto anteriormente, cada equação foi aplicada ao conjunto de componentes que estão no repositório de componentes. Ao analisar os resultados para cada componente, pode-se perceber quais são os melhores componentes para compor o sistema final. Uma exploração desse espaço do projeto pode então ser realizada para a escolha final dos componentes.

A Tabela (18) mostra os resultados da equação de consumo de energia, já tratando cada *mote* como um único componente. Essa também mostra todas as opções que foram consideradas no estudo. Ao se analisar os resultados, pode-se perceber que o *mote* EposMote é o que o menos consome energia, enquanto o mote Telos é o que mais consome energia. A Tabela (19) traz os resultados da equação de desempenho, aplicados aos EposMote. O tempo de execução completo do sistema, levando em consideração as limitações impostas e os dados fornecidos pelo reporitório é de 142,87 milissegundos.

No caso do requisito de uso de memória, foi considerada apenas uma implementação dos componentes de software do sistema, feita utilizando-se o sistema EPOS. Por esta razão, o requisito de uso de memória não possui diferentes alternativas de implementação. Caso a implementação utilize mais memória do que os projetistas estão dis-

Tabela 17: Cálculo do custo financeiro dos componentes de software.

Componentes	**train**	**Custo Finaceiro**
Actuator	0.191	831.01
Semaphore	0.764	13.23
Thermometer	0.191	879.89
Alarm	0.764	8.55
Scheduler	0.90725	11.02
System	0.764	1.38
Communicator	0.4775	23.50
Thread	0.6685	151.11
MMU	0.90725	5.51
Timer	0.764	7.28
Heap	0.90725	5.51
String	0.90725	5.51
Malloc	0.764	6.54

Tabela 18: Resultado da equação de consumo de energia para cada opção considerada.

Opções	**E**
EposMote	532,28297
Mica-Z	837,63043
Mica-2	924,21786
ZigBit	1065,08745
Telos	1601,86124

Tabela 19: Resultado da equação de desempenho para cada tarefa considerada.

Tarefas	**time (ms)**	**count**	**T (ms)**
Recovery	127,85	1	127,85
Trigger	2,28	1	2,28
Main	8,27	1	8,27
Monitor	0,447	10	4,47

postos a colocar no sistema, a especificação do sistema deve ser revista, seja mudando a implementação dos componentes de software ou alterando as especificações de capacidade de memória. A Tabela (20) exibe

Tabela 20: Resultado da equação de uso de memória.

Código	**Dados**	**M**
34920,30905	141,30053	35061,60959

a utilização de memória para código, dados e o total. O resultado final não é um valor inteiro, devido a todos os erros atribuídos nas equações de uso de memória. Porém, os projetistas podem considerar o resultado como sendo o menor valor inteiro que seja maior do que o obtido nas equações.

Tabela 21: Resultado da equação de custo financeiro para cada opção considerada.

Custo Software	1950,07
Plataforma	**Custo Total**
EposMote + Software	3885,00
Mica-2 + Software	5245,89
Mica-Z + Software	5290,58
ZigBit + Software	5722,96
Telos + Software	32053,37

Por fim, na Tabela (21) estão os resultados das equações de custo financeiro. Esses resultados estão divididos no custo dos componentes de software e no custo dos componentes de hardware. Como existe apenas uma implementação de software, há apenas um custo para software. O custo de hardware varia dependendo do *mote* em questão. Na Tabela (21) já estão então somados os custos de software e hardware para cada *mote*. Como pode-se perceber, o EposMote é a opção mais viável em termos de custo financeiro, enquanto o *mote* Telos é a opção menos viável. É importante ressaltar que esse custo não é apenas o valor monetário a ser pago pelos componentes. Vários outros fatores influenciam no resultado, como disponibilidade no mercado, experiência da equipe de projetistas e também o estágio de desenvolvimento dos sistemas, por exemplo.

Embora não tenham sido considerados todos os aspectos dos *motes* analisados, o mote EposMote cumpre os requisitos definidos na especificação do projeto. Caso ele não cumprisse algum requisito, necessariamente os outros *motes* precisariam ser avaliados com mais detalhes ou a especificação precisaria ser revista. Essa decisão depende

da vontade dos projetistas.

Independente da solução escolhida, todas as tabelas descritas nesse capítulo foram obtidas sem muitas dificuldades, apenas utilizando as equações propostas, um repositório de componentes e uma descrição de todas as funcionalidades do sistema. O trabalho realizado aqui pode ser facilmente incrementado com a utilização de ferramentas mais completas de exploração de espaço de projeto, capazes de avaliar diversos outros aspectos das equações, aspectos estes que já estão modelados. Além disso, repositórios com mais opções de componentes também podem ser usados sem causar alterações nos modelos. De fato, as equações não precisam sofrer alterações para serem usadas em diferentes ferramentas de exploração, nem para modelar outros sistemas embarcados diferentes dos que foram usados como exemplo.

5.5 EQUAÇÕES DE COEFICIENTES

As equações vistas nessa dissertação permitem que sejam atribuídos à maioria das variáveis alguns erros que tentam atenuar possíveis dependências e incertezas inerentes à essas variáveis. Esses erros, porém, devem ser definidos pelos próprios projetistas, talvez com base em projetos anteriores, ou com base em ferramentas de otimização. É possível otimizar os erros e coeficientes usados nas equações, de forma a atingir melhores resultados no projeto integrado de software e hardware e também de forma a reutilizar esses valores para projetos futuros.

Visando facilitar a otimização desses coeficientes, nesta seção serão apresentadas algumas equações reduzidas, onde as variáveis que representam dados dos componentes serão substituídas por valores reais mas os coeficientes serão mantidos. Isso resulta em equações com diversas constantes e uma quantidade bem reduzida de variáveis. De fato, apenas erros e coeficientes permanecem, e isso possibilita que essas equações de coeficientes sejam otimizadas com mais facilidade.

Como as equações usam valores do repositório de componentes, uma única equação se transforma em várias, uma para cada componente do repositório. Para simplificar, as equações que serão mostradas nesta seção consideram apenas a utilização do *mote* EposMote. Com base no requisito de desempenho, serão apresentadas então equações de coeficientes de desempenho para cada uma das tarefas do sistema, quando executadas no EposMote. A equação da tarefa Recovery executando no EposMote pode então ser definida como:

$$time_{Recovery}(EposMote) = \frac{\mu}{26 \times exp_h \times exp_\mu} \times (\frac{20}{1,5 \times exp_{wcIns}} \tag{5.1}$$

$$+\frac{3,75}{1,5 \times exp_{wcIns}} + \frac{19}{exp_{wcIns}} + \frac{2,5}{exp_{wcIns}} \tag{5.2}$$

$$+\frac{1}{exp_{wcIns}} + \frac{1,25}{0,25 \times exp_{wcIns}} + \frac{1,25}{0,75 \times exp_{wcIns}} \tag{5.3}$$

$$+\frac{2,5}{0,8 \times exp_{wcIns}} + \frac{1,25}{0,8 \times exp_{wcIns}} + \frac{2,5}{0,75 \times exp_{wcIns}}) \tag{5.4}$$

$$+\frac{100}{exp_{wcCount_{bs}}} \tag{5.5}$$

O número 26 que aparece na equação acima veio da Tab. (3) e representa a variável f, que é a frequência do processador. A equação apresenta um somatório de frações. Nesse somatório, o numerador consiste no pior caso do número de instruções executadas naquela tarefa para cada tipo de instrução. No caso, como existem 10 termos sendo somados entre os parênteses, considera-se 10 tipos de instrução diferentes. O número real que se encontra multiplicando os valores de exp_{wcIns} em cada fração vem do número de instruções daquele tipo que são executadas em um único ciclo do processador. O valor 100 no fim da equação vem do tempo que a tarefa fica parada esperando por causa do *timer* inserido.

Dessa forma, todos os dados substituídos na equação acima foram obtidos do repositório de componentes, e só restam as contantes e os erros atribuídos aos valores. Também são apresentadas abaixo as equações de coeficientes de desempenho das tarefas Trigger, Main e Monitor para o EposMote. Novamente, o número 26 (ou 2.6 devido à execução 10 vezes da tarefa Monitor) representa a frequência do processador. Os numeradores na soma entre parênteses representam o número de instruções de cada tipo, e o número real no denominador de cada fração da soma representa o número de instruções daquele tipo que são executadas em um único ciclo. Esses valores foram resumidos quando foram apresentados nas Tab. (3), (5) e (7).

$$time_{Trigger}(EposMote) = \frac{1}{26} \times (\frac{10 \times O_t}{1,5 \times exp_t \times exp_{eIns} \times {dev_t}^2} \tag{5.6}$$

$$+\frac{8 \times O_t}{exp_t \times exp_{eIns} \times {dev_t}^2} + \frac{6 \times O_t}{exp_t \times exp_{eIns} \times {dev_t}^2} \tag{5.7}$$

$$+\frac{O_t}{0,25 \times exp_t \times exp_{eIns} \times {dev_t}^2} + \frac{O_t}{0,75 \times exp_t \times exp_{eIns} \times {dev_t}^2}) \tag{5.8}$$

$$time_{Main}(EposMote) = \frac{1}{26} \times (\frac{62 \times \phi}{1,5 \times exp_n \times exp_\phi \times dev_n} \tag{5.9}$$

$$+\frac{21 \times \phi}{1,5 \times exp_n \times exp_\phi \times dev_n} + \frac{63 \times \phi}{exp_n \times exp_\phi \times dev_n} \tag{5.10}$$

$$+\frac{19 \times \phi}{exp_n \times exp_\phi \times dev_n} + \frac{3 \times \phi}{0,25 \times exp_n \times exp_\phi \times dev_n} \tag{5.11}$$

$$+\frac{5 \times \phi}{0,25 \times exp_n \times exp_\phi \times dev_n} + \frac{\phi}{0,75 \times exp_n \times exp_\phi \times dev_n} \tag{5.12}$$

$$+\frac{\phi}{0,8 \times exp_n \times exp_\phi \times dev_n}) \tag{5.13}$$

$$time_{Monitor}(EposMote) = \frac{1}{2,6} \times (\frac{2 \times \phi}{1,5 \times exp_n \times exp_\phi \times dev_n} \tag{5.14}$$

$$+\frac{\phi}{1,5 \times exp_n \times exp_\phi \times dev_n} + \frac{2,2 \times \phi}{exp_n \times exp_\phi \times dev_n} \tag{5.15}$$

$$+\frac{2 \times \phi}{exp_n \times exp_\phi \times dev_n} + \frac{0,2 \times \phi}{0,75 \times exp_n \times exp_\phi \times dev_n} \tag{5.16}$$

$$+\frac{\phi}{0,8 \times exp_n \times exp_\phi \times dev_n}) \tag{5.17}$$

No caso do requisito de uso de memória, como considerou-se apenas uma implementação dos componentes de software, pode-se gerar uma equação de coeficientes única para uso de memória, como visto a seguir.

$$M = \frac{1860}{dev_{cs}} + \frac{1100}{dev_{cs}} + \frac{1424}{dev_{cs}} + \frac{2204}{dev_{cs}} + \frac{4360}{dev_{cs}} + \frac{4040}{dev_{cs}} \quad (5.18)$$

$$+\frac{7280}{dev_{cs}} + \frac{6304}{dev_{cs}} + \frac{884}{dev_{cs}} + \frac{1892}{dev_{cs}} + \frac{408}{dev_{cs}} + \frac{600}{dev_{cs}} + \frac{1660}{dev_{cs}} \quad (5.19)$$

Nesta equação, cada fração representa um componente de software. O numerador das frações representa o tamanho em bytes daquele componente, e o numerador representa um erro referente ao estágio de desenvolvimento do componente.

Com base nessas equações, é possível otimizar os coeficientes desconhecidos para um melhor desempenho no EposMote ou um menor uso de memória. O mesmo poderia ser feito para os outros *motes* e para outros tipos de tarefas. As equações de consumo de energia e custo financeiro também poderiam ser representadas dessa forma, porém não serão mostradas nessa seção por questão de espaço, visto que existem diversos componentes no repositório. As equações demonstradas nessa seção ficam então como exemplo, permitindo que projetistas façam o mesmo com as outras equações.

5.6 CONSIDERAÇÕES E LIMITAÇÕES DO ESTUDO DE CASO

Nesse capítulo foi apresentado um estudo de caso que busca demonstrar a adequabilidade da proposta apresentada no capítulo anterior. O estudo consiste no projeto de um sistema embarcado com requisitos não funcionais e funcionais que requerem uma modelagem de custos que leve em consideração aspectos como consumo de energia, desempenho, uso de memória e custo financeiro. Dessa forma, as equações propostas puderam ser utilizadas no projeto desse sistema.

Como já descrito anteriormente, a utilização das equações está subordinada à existência de um repositório de componentes, onde são descritas diversas características de cada componentes, as quais são usadas nas equações. No estudo de caso realizado, essas características se dividiam entre estimativas por parte dos projetistas e dados concretos obtidos através de outras publicações ou tabelas fornecidas pela indústria.

Uma limitação desse estudo de caso foi que alguns dos dados que deveriam ser fornecidos pela indústria não puderam ser obtidos, e também foram estimados pelos projetistas. Nesse caso, embora os dados sejam fictícios, a avaliação das equações em si não foi prejudi-

cada. É importante frisar que o trabalho desenvolvido é empírico, e por isso os resultados não podem ser usados como parâmetro para uma implementação real do sistema projetado no estudo. Porém, os resultados são válidos para o objetivo do estudo, que era demonstrar que é possível, através unicamente da modelagem proposta, modelar os todos os principais custos de um sistema embarcado dentre os suportados pelo modelo.

A realização desse estudo de caso possibilita que se identifiquem os pontos positivos e negativos da abordagem proposta. O principal ponto negativo da proposta consiste nas equações dependerem muito do repositório de componentes. Esse repositório precisa ter diversas informações a respeito dos componentes, e várias dessas informações nem sempre estão disponíveis. Algumas características dessas exigiriam inclusive a utilização de outras ferramentas para que elas pudessem ser obtidas. Por outro lado, cumpriu-se o objetivo de modelar com uma única linguagem os principais requisitos de sistemas embarcados. A utilização da linguagem matemática mostrou-se vantajosa, pois permite uma representação mais formal dos custos. Um dos maiores desafios do trabalho era conseguir representar todos os custos desejados usando uma única linguagem, e esse desafio foi cumprido, embora várias limitações tenham sido impostas ao modelo para permitir isso.

6 CONCLUSÕES E TRABALHOS FUTUROS

A utilização de sistemas embarcados está cada vez mais presente na vida das pessoas. Eles estão presentes em diversos lugares, executando funcionalidades das mais variadas. E ao mesmo tempo em que a utilização desses sistemas cresce, o mercado também passa a investir mais nessa área e a competição entre fabricantes se acentua. É importante não apenas fornecer um produto, mas principalmente desenvolvê-lo no menor tempo possível e fabricá-lo a baixos custos. O projeto de um sistema embarcado deve então tratar diversas questões referentes a requisitos funcionais e não funcionais do sistema, de forma mais crítica do que ocorre em sistemas de propósito geral. Ao mesmo tempo, devido à variabilidade de finalidades para os sistemas embarcados, algumas técnicas e metodologias podem ser menos eficazes ou até mesmo não serem aplicáveis a alguns tipos de sistemas.

A primeira etapa do projeto de um sistema embarcado é então a definição de requisitos funcionais, os quais variam amplamente entre diferentes projetos. Devido a essa pluralidade, conseguir reutilizar artefatos de software, hardware ou metodologias de desenvolvimento pode reduzir muito os custos de diferentes sistemas embarcados. Um fator agravante desse cenário é que muitos dos requisitos de sistemas embarcados são conflitantes. Reduzir o consumo de energia de um sistema embarcado pode ocasionar uma perda de desempenho ou um maior uso de memória, ou até mesmo um maior custo total para o sistema. Encontrar um meio termo que satisfaça todos os requisitos do sistema pode ser uma tarefa inviável sem a utilização de ferramentas e modelos bem definidos.

É essencial que seja realizado um planejamento amplo e que ao mesmo tempo considere diversas especificidades desses sistemas embarcados. É importante também que o planejamento possa ser reutilizado para outros sistemas ou expandido. Porém, de forma geral, há poucos casos no mundo em que essa modelagem é realizada levando-se em consideração tantos fatores diferentes, como consumo de energia, área, peso, utilização de memória, desempenho, preço do sistema, etc, e que ao mesmo tempo seja padronizada, extensível, reutilizável e de fácil compreensão.

Para resolver esse problema foi proposto nessa dissertação a modelagem dos custos de um sistema embarcado através de equações matemáticas amplas e ao mesmo tempo que tratem especificidades. Através de uma especificação inicial do sistema, o objetivo é capturar

os requisitos do sistema e expressá-los matematicamente. A modelagem consiste então em um grupo de equações que descreve diferentes requisitos do sistema. Modelar de forma consistente e padronizada os principais requisitos de um sistema embarcado é uma contribuição científica. Com base nessas equações de cada requisito, uma exploração de espaço de projeto pode ser facilmente e eficientemente realizada.

Essas equações abstraem os requisitos do sistema, tratam as incertezas inerentes a eles, e podem ser aplicadas para uma grande variedade de sistemas embarcados, permitindo que os projetistas modelem todos os principais requisitos do sistema apenas com base nessas equações. Após serem apresentadas as especificidades de cada equação e sua relação com sistemas reais, foi apresentado um estudo de caso onde foi projetado do início um novo sistema embarcado, com sua especificação de requisitos funcionais e não funcionais. Todas as equações criadas foram então aplicadas a essa especificação inicial do sistema. A contribuição desse trabalho são essas equações, e através do uso delas obteve-se soluções satisfatórias de componentes e configurações para o sistema embarcado, onde todos os requisitos foram cumpridos.

Como trabalho futuro, almeja-se integrar as equações a uma ferramenta de exploração do espaço de projeto capaz de otimizar as equações, aperfeiçoando-as de modo a aumentar a eficácia delas. Além disso, todas as limitações da abordagem descritas nessa dissertação servem de guia para possíveis extensões do modelo proposto.

REFERÊNCIAS

ALLARA, A. et al. System-level performance estimation strategy for sw and hw. In: *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*. [S.l.: s.n.], 1998. p. 48 –53. ISSN 1063-6404.

ALLARA, A. et al. A flexible model for evaluating the behavior of hardware/software systems. In: *Proceedings of the Fifth International Workshop on Hardware/Software Codesign*. [S.l.: s.n.], 1997. p. 109 –114.

AXELSSON, J. A method for evaluating uncertainties in the early development phases of embedded real-time systems. In: *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. [S.l.: s.n.], 2005. p. 72 – 75. ISSN 1533-2306.

AXELSSON, J. Cost models with explicit uncertainties for elextronic architecture trade-off and risk analysis. In: *INCOSE 2006 Annual International Symposium on Systems Engineering*. [S.l.: s.n.], 2006.

AZIZI, O. et al. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis. In: *Proceedings of the 37th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2010. (ISCA '10), p. 26–36. ISBN 978-1-4503-0053-7. <http://doi.acm.org/10.1145/1815961.1815967>.

BHATTI, S. et al. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, Springer Netherlands, v. 10, p. 563–579, 2005. ISSN 1383-469X. 10.1007/s11036-005-1567-8. <http://dx.doi.org/10.1007/s11036-005-1567-8>.

BOEHM, B. et al. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, Springer, v. 1, n. 1, p. 57–94, 1995. <http://www.springerlink.com/index/10.1007/BF02249046>.

BRANDOLESE, C. Source-level estimation of energy consumption and execution time of embedded software. In: *11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*. [S.l.: s.n.], 2008. p. 115 –123.

BRINDLE, K. E. *The Relationship Between Life-Cycle Costing and Performance: An Exploratory Analysis*. Dissertação (Mestrado), 2005.

BUTT, S.; SAYYAH, P.; LAVAGNO, L. Model-based hardware/software synthesis for wireless sensor network applications. In: *2011 Saudi International Electronics, Communications and Photonics Conference*. [S.l.: s.n.], 2011. p. 1 –6.

CANCIAN, R. L. *Um Modelo Evolucionário de Otimização Multiobjetivo para Exploração do Espaço de Projeto em Sistemas Embarcados*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2011.

CHEDE, S.; KULAT, K. Algorithm to optimize code size and energy consumption in real time embedded system. *Journal of Computers*, v. 3, p. 15–21, June 2008. ISSN 1796-203X.

CHEN, H.-C. et al. Cost efficient codesign for jpeg system. In: *15th IEEE International Symposium on Consumer Electronics*. [S.l.: s.n.], 2011. p. 497 –502. ISSN 0747-668X.

CHULANI, S.; BOEHM, B.; STEECE, B. Bayesian analysis of empirical software engineering cost models. *IEEE Transactions on Software Engineering*, v. 25, n. 4, p. 573 –583, jul/aug 1999. ISSN 0098-5589.

CLEMEN, R. T.; FISCHER, G. W.; WINKLER, R. L. Assessing dependence: Some experimental results. *Management Science*, JSTOR, v. 46, n. 8, p. 1100–1115, 2000. <http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.46.8.1100.12023>.

DEBARDELABEN, J. A.; MADISETTI, V. K.; GADIENT, A. J. Incorporating cost modeling in embedded system design. *IEEE Design Test of Computers*, v. 14, n. 3, p. 24–35, 1997. <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=605989>.

DHOUIB, S. et al. Modelling and estimating the energy consumption of embedded applications and operating systems. In: *Proceedings of the 12th International Symposium on Integrated Circuits*. [S.l.: s.n.], 2009. p. 457 –461.

DONG, X.; XIE, Y. System-level cost analysis and design exploration for three-dimensional integrated circuits (3d ics). In: *Asia and South Pacific Design Automation Conference*. [S.l.: s.n.], 2009. p. 234 –241.

DUNKELS, A.; GRONVALL, B.; VOIGT, T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In: *29th Annual IEEE International Conference on Local Computer Networks*. [S.l.: s.n.], 2004. p. 455 – 462. ISSN 0742-1303.

DUNKELS, A. et al. Software-based on-line energy estimation for sensor nodes. In: *Proceedings of the 4th workshop on Embedded networked sensors*. New York, NY, USA: ACM, 2007. (EmNets '07), p. 28–32. ISBN 978-1-59593-694-3. <http://doi.acm.org/10.1145/1278972.1278979>.

FRÖHLICH, A. A. *Application-Oriented Operating Systems*. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 2001. (GMD Research Series, 17).

FRÖHLICH, A. A. A comprehensive approach to power management in embedded systems. *International Journal of Distributed Sensor Networks*, v. 2011, p. 1:19, 2011.

FRÖHLICH, A. A.; STEINER, R. V.; RUFINO, L. M. A trustful infrastructure for the internet of things based on eposmote. In: *Proceedings of the 9th IEEE International Conference on Dependable, Autonomic and Secure Computing*. [S.l.: s.n.], 2011.

FUMMI, F. et al. A cosimulation methodology for hw/sw validation and performance estimation. *ACM Transactions on Design Automation of Electronic Systems*, ACM, New York, NY, USA, v. 14, p. 23:1–23:32, April 2009. ISSN 1084-4309. <http://doi.acm.org/10.1145/1497561.1497566>.

GAJSKI, D. Ip-based design methodology. In: *Proceedings of the 36th Design Automation Conference*. [S.l.: s.n.], 1999. p. 43.

GUSTAFSSON, J. et al. Approximate worst-case execution time analysis for early stage embedded systems development. In: LEE, P. N. S. (Ed.). *Proceedings of the 7th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems*. [S.l.]: Lecture Notes in Computer Science (LNCS), Springer, 2009. p. 308–319.

HART, J. K.; MARTINEZ, K. Environmental sensor networks:a revolution in the earth system science? *Earth-Science Reviews*, Elsevier, v. 78, p. 177–191, 2006. <http://eprints.ecs.soton.ac.uk/13093/>.

HU, X.; ZHOU, T.; SHA, E.-M. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 9, n. 6, p. 833 –844, dec. 2001. ISSN 1063-8210.

JACOBSON, I. *Object Oriented Software Engineering: A Use Case Driven Approach*. [S.l.]: Addison-Wesley Professional, 1992.

JAYASEELAN, R.; MITRA, T.; LI, X. Estimating the worst-case energy consumption of embedded software. In: *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.: s.n.], 2006. p. 81 – 90.

JUNIOR, A. H.; FRÖHLICH, A. A. On the monitoring of system-level energy consumption of battery-powered embedded systems. In: *2011 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.: s.n.], 2011. p. 2608–2613.

KALAVADE, A.; MOGHé, P. A tool for performance estimation of networked embedded end-systems. In: *Proceedings of the 35th annual Design Automation Conference*. New York, NY, USA: ACM, 1998. (DAC '98), p. 257–262. ISBN 0-89791-964-5. <http://doi.acm.org/10.1145/277044.277116>.

KIM, J. et al. Fast estimation of software energy consumption using ipi(inter-prefetch interval) energy model. In: *IFIP International Conference on Very Large Scale Integration*. [S.l.: s.n.], 2007. p. 224 –229.

KRÄMER, M.; GERALDY, A. Energy measurements for micaz node. *University of Kaiserslautern, Kaiserslautern, Germany, Technical Report KrGe06*, 2006.

KUJAWSKI, E.; ALVARO, M. L.; EDWARDS, W. R. Incorporating psychological influences in probabilistic cost analysis. *Syst. Eng.*, John Wiley and Sons Ltd., Chichester, UK, v. 7, p. 195–216, September 2004. ISSN 1098-1241. <http://dl.acm.org/citation.cfm?id=1077494.1077495>.

LAJOLO, M. et al. Cosimulation-based power estimation for system-on-chip design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v. 10, n. 3, p. 253 –266, june 2002. ISSN 1063-8210.

LEE, S. et al. A design framework for real-time embedded systems with code size and energy constraints. *ACM Transactions in Embedded Computing Systems*, ACM, New York, NY, USA, v. 7, p. 18:1–18:27, January 2008. ISSN 1539-9087. <http://doi.acm.org/10.1145/1331331.1331342>.

LI, J. et al. Resource allocation robustness in multi-core embedded systems with inaccurate information. *Journal of Systems Architecture*, v. 57, n. 9, p. 840 – 849, 2011. ISSN 1383-7621. <http://www.sciencedirect.com/science/article/pii/S1383762111000361>.

LISHA. *EPOS Project*. 2012. DisponÃvel em: http://epos.lisha.ufsc.br Acesso em 18 de julho 2012.

LUO, G. et al. Analysis and optimization of embedded software energy consumption on the source code and algorithm level. In: *4th International Conference on Embedded and Multimedia Computing*. [S.l.: s.n.], 2009. p. 1 –5.

MACHADO, A. M.; FRÖHLICH, A. A. A Lua Virtual Machine for Resource-Constrained Embedded Systems. In: *Proceedings of the IADIS International Conference Applied Computing*. [S.l.: s.n.], 2010. p. 175–182. ISBN 978-972-8939-30-4.

MAHARBIZ, M. M.; SATO, H. Cyborg beetles. *Scientific American*, v. 303, p. 94:99, 2010.

MARCONDES, H. *Uma Arquitetura de Componentes HÃbridos de Hardware e Software para Sistemas Embarcados*. Dissertação (Mestrado), 2009.

MARWEDEL, P. *Embedded System Design*. [S.l.]: Kluwer Academic Publishers, 2003. ISBN 1-4020-7690-9.

MU, J.; LYSECKY, R. Profile assisted online system-level performance and power estimation for dynamic reconfigurable embedded systems. In: *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. Piscataway, NJ, USA: IEEE Press, 2011. (ASPDAC '11), p. 737–742. ISBN 978-1-4244-7516-2. <http://dl.acm.org/citation.cfm?id=1950815.1950956>.

NIAR, S.; INGLART, N. Rapid performance and power consumption estimation methods for embedded system design. In: *7th IEEE International Workshop on Rapid System Prototyping*. [S.l.: s.n.], 2006. p. 47 –53. ISSN 1074-6005.

NOGUEIRA, B. et al. A formal model for performance and energy evaluation of embedded systems. *EURASIP Journal on Embedded Systems*, Hindawi Publishing Corp., New York, NY, United States, v. 2011, p. 2:1–2:12, January 2011. ISSN 1687-3955. <http://dx.doi.org/10.1155/2011/316510>.

PERSSON, E. *Resource utilization in embedded systems - an economical perspective*. Dissertação (Mestrado), 2008.

PHAM, H.; ZHANG, X. A software cost model with warranty and risk costs. *IEEE Transactions on Computers*, v. 48, n. 1, p. 71 –75, jan 1999. ISSN 0018-9340.

PIMENTEL, A. D. The artemis workbench for system-level performance evaluation of embedded systems. *International Journal of Embedded Systems*, v. 3, p. 181–196, 2008.

POLASTRE, J.; SZEWCZYK, R.; CULLER, D. Telos: enabling ultra-low power wireless research. In: *Fourth International Symposium on Information Processing in Sensor Networks*. [S.l.: s.n.], 2005. p. 364 – 369.

POP, P. Embedded system design: Optimization challenges. In: *CPAIOR*. [S.l.: s.n.], 2005. p. 16 –16.

PRABH, K. S. *Real-Time Wireless Sensor Networks*. Tese (Doutorado) — University of Virginia, May 2007.

QIU, M.; SHA, E. H. M. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, ACM, New York, NY, USA, v. 14, p. 25:1–25:30, April 2009. ISSN 1084-4309. <http://doi.acm.org/10.1145/1497561.1497568>.

RAGAN, D.; SANDBORN, P.; STOAKS, P. A detailed cost model for concurrent use with hardware/software co-design. In: *Proceedings of the 39th annual Design Automation Conference*. New York, NY, USA: ACM, 2002. (DAC '02), p. 269–274. ISBN 1-58113-461-4. <http://doi.acm.org/10.1145/513918.513989>.

SAMETINGER, J. *Software Engineering with Reusable Components*. [S.l.]: Springer, 1997.

SEO, C. *Prediction of energy consumption behavior in component-based distributed systems*. Tese (Doutorado) — University of Southern California, 2008.

SEO, C.; MALEK, S.; MEDVIDOVIC, N. An energy consumption framework for distributed java-based systems. In: *Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2007. (ASE '07), p. 421–424. ISBN 978-1-59593-882-4. <http://doi.acm.org/10.1145/1321631.1321699>.

SIMON, D. E. *An Embedded Software Primer*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1999.

UEDA, K. et al. Architecture-level performance estimation for ip-based embedded systems. In: *Proceedings of the conference on Design, automation and test in Europe - Volume 2*. Washington, DC, USA: IEEE Computer Society, 2004. (DATE '04), p. 21002–. ISBN 0-7695-2085-5. <http://dl.acm.org/citation.cfm?id=968879.969144>.

WEERASEKERA, R. et al. Two-dimensional and three-dimensional integration of heterogeneous electronic systems under cost, performance, and technological constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 28, n. 8, p. 1237 –1250, aug. 2009. ISSN 0278-0070.

WENG, L.-C.; WANG, X.; LIU, B. A survey of dynamic power optimization techniques. In: . Los Alamitos, CA, USA: IEEE Computer Society, 2003. v. 0, p. 48. ISBN 0-7695-1944-X.

WOLF, W. A decade of hardware/software codesign. *Computer*, v. 36, n. 4, p. 38 – 43, april 2003. ISSN 0018-9162.

YANG, T.; TOH, Y. K.; XIE, L. Run-time monitoring of energy consumption in wireless sensor networks. In: *IEEE International Conference on Control and Automation*. [S.l.: s.n.], 2007. p. 1360 –1365.

ZAMORA, N. H.; HU, X.; MARCULESCU, R. System-level performance/power analysis for platform-based design of multimedia applications. *ACM Transactions on Design Automation of Electronic Systems*, ACM, New York, NY, USA, v. 12, p. 2:1–2:29, February 2007. ISSN 1084-4309. <http://doi.acm.org/10.1145/1188275.1188277>.

ZHAO, X. et al. Fine-grained energy estimation and optimization of embedded operating systems. In: *International Conference on Embedded Software and Systems Symposia*. [S.l.: s.n.], 2008. p. 90 –95.

ZITZLER, E.; THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE*

Transactions on Evolutionary Computation, v. 3, n. 5, p. 257–271, 1999.