UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MODELAGEM DE SOFTWARE DE TEMPO REAL UTILIZANDO O *PROFILE* MARTE DA UML

EDUARDO AUGUSTO SILVESTRE

Uberlândia - Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

EDUARDO AUGUSTO SILVESTRE

MODELAGEM DE SOFTWARE DE TEMPO REAL UTILIZANDO O *PROFILE* MARTE DA UML

Dissertação de Mestrado apresentada à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como parte dos requisitos exigidos para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador:
Prof. Dr. Michel dos Santos Soares

Uberlândia, Minas Gerais
2012

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Modelagem de software de tempo real utilizando o *profile* MARTE da UML**" por **Eduardo Augusto Silvestre** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação**.

Uberlândia, 20 de Abril de 2012

Orientador: _____
Prof. Dr. Michel dos Santos Soares
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Stéphane Julia
Universidade Federal de Uberlândia

_____
Prof. Dr. Heitor Augustus Xavier Costa
Universidade Federal de Lavras

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Data: Abril de 2012

Autor:	**Eduardo Augusto Silvestre**
Título:	**Modelagem de software de tempo real utilizando o *profile* MARTE da UML**
Faculdade:	**Faculdade de Computação**
Grau:	**Mestrado**

Fica garantido à Universidade Federal de Uberlândia o direito de circulação e impressão de cópias deste documento para propósitos exclusivamente acadêmicos, desde que o autor seja devidamente informado.

_____

Autor

Dedicatória

Dedico este trabalho a todos os meus familiares.

Agradecimentos

Agradeço à Deus pela força, proteção e amparo.

Agradeço à toda a minha família. Especialmente a meu pai Anselmo, minha mãe Vera e a minha irmã Leidiane.

Agradeço ao meu orientador, Michel, por ter acreditado e apoiado na realização desta pesquisa sempre de forma paciente e solícita. Certamente, sem dedicação e atenção isto não teria sido possível.

Agradeço também ao professor Ilmério, que foi meu primeiro orientador no mestrado.

Agradeço ao Instituto Federal do Triângulo Mineiro. Especialmente ao diretor do *Campus* Patrocínio, professor Ernani, que adequou os meus horários para a realização deste trabalho.

Agradeço a todos os amigos do mestrado.

Agradeço também a todos os funcionários da FACOM.

Por fim, agradeço a todos que contribuíram direta ou indiretamente para a realização e conclusão deste trabalho.

O sofrimento é passageiro, desistir é para sempre.

Lance Armstrong

Resumo

Métodos e linguagens orientadas a objetos tem sido amplamente aplicados em atividades de modelagem e projeto de software de tempo real. Entre todas as notações orientadas a objetos a mais usada é a UML. Apesar dos seus muitos problemas conhecidos na teoria e na prática, como a fraca representação de restrições temporais e alocação de recursos, a UML tem sido efetivamente aplicada neste domínio. O *profile* MARTE foi proposto para resolver essas questões. Neste trabalho o *profile* MARTE é aplicado juntamente com a UML para modelagem de um software de controle de sinais de trânsito com o objetivo de analisar os prós e contras da utilização deste recente *profile* OMG. O *profile* MARTE é comparado com UML e SPT (um *profile* anterior ao MARTE). O resultado é que com MARTE os modelos UML são mais específicos e expressivos, mas também são mais complexos. Adicionalmente, tornar-se proficiente na utilização do *profile* é um grande desafio. Pesquisas e aplicações industriais ainda são necessárias para melhores conclusões sobre a utilização do MARTE.

Palavras chave: MARTE, UML, Arquitetura de Software, Sistemas Distribuídos, Sistemas de Tempo Real, Controle de Sinais de Tráfego.

Abstract

Object-oriented methods and languages have been widely applied in activities of modeling and design of real-time systems. Among all object-oriented notations, UML is the most used one. Despite the many well-known issues both in theory and practice, such as lack of concepts to represent time constraints and resource allocation, UML has been frequently applied to this domain. The MARTE profile was proposed in order to solve these issues. In this work the MARTE profile is applied together with UML to modeling a software for traffic signals control in order to analyze the pros and cons of using this recent OMG profile. The MARTE profile is compared with UML and SPT (a former MARTE profile). The result is that with MARTE the models are more specific and expressive, but are also more complex. In addition, mastering the MARTE language is a hard challenge. Researches and industrial applications are still required to achieve better conclusions on the use of MARTE.

Keywords: MARTE, UML, Software Architecture, Real-Time Systems, Distributed Systems, Road Traffic Signals Conrol.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Neste primeiro capítulo serão abordados assuntos introdutórios e relevantes para a contextualização de todo o trabalho. A Seção 1.1 faz a contextualização sobre software de tempo real, apresentando conceitos relevantes para melhor compreensão do texto. A Seção 1.2 descreve o problema motivador para esta pesquisa. A Seção 1.3 lista os principais objetivos do trabalho, assim como algumas das suas contribuições. A Seção 1.4 detalha aspectos da metodologia de pesquisa, incluindo as questões, estratégias e instrumentos de pesquisa. A Seção 1.5 descreve trabalhos relacionados à abordagem proposta nesta pesquisa. Por fim, a Seção 1.6 mostra uma visão geral dos capítulos da dissertação.

1.1 Contextualização sobre a modelagem de software de tempo real

Software atualmente é usado para controlar uma ampla variedade de sistemas, desde máquinas domésticas simples até fábricas inteiras. O software nesses sistemas é um software embutido de tempo real que deve reagir a eventos gerados pelo hardware e emitir sinais de controle em resposta a esses eventos [Sommerville 2010].

Existem alguns sistemas de tempo real bastante conhecidos e empregados nas mais diversas situações. Alguns dos sistemas mais famosos são os sistemas integrados a aviões (navegação e *display*), sistemas multimídia (*games* e simuladores), sistemas médicos (cirurgia realizada por robô, cirurgia remota e imagens médicas), sistemas industriais (robótica e inspeção automatizada) e sistemas civis (controle de elevador e sistemas automotivos).

A preocupação com o desenvolvimento de aplicações de tempo real existe há décadas, mas especialmente nos últimos anos, as aplicações de tempo real vem recebendo maior atenção [Yongfeng et al. 2009]. Isto se deve ao uso disseminado de componentes de hardware, que devido aos avanços tecnológicos, tiveram um aumento significativo em sua capacidade de processamento, ao mesmo tempo em que diminuíram o custo, o consumo

de energia e o tamanho.

Os recentes avanços nas áreas de eletrônica, informática e microeletrônica tem possibilitado a utilização destes sistemas numa série de processos que em outros tempos não eram economicamente viáveis ou tecnicamente possíveis. As tendências indicam que os sistemas de tempo real do futuro devem ser ainda maiores, mais complexos e com longos ciclos de vida, exibindo um comportamento dinâmico e altamente adaptativo em relação às diferentes condições de operação [Bihari e Gopinath 1992] [Hölzl et al. 2008]. Estes requisitos adicionais tornam o projeto de sistemas de tempo real extemamente complexos, pois, além de terem que satisfazer requisitos temporais, trabalham com uma alta quantidade de componentes de software e hardware [Williams 2006].

O projeto de software de tempo real é complexo e envolve conceitos normalmente pouco explorados e analisados pela computação de propósito geral [Williams 2006]. O projeto destes sistemas deve levar em consideração o hardware e o software necessários à realização da tarefa para a qual o sistema foi concebido. Geralmente, existem fatores limitantes no projeto como, por exemplo, a disponibilidade limitada de recursos computacionais (quantidade de memória, poder de processamento), o limite do consumo de energia ou ainda o curto espaço de tempo de projeto [Carro e Wagner 2003].

Historicamente, muitos sistemas de tempo real foram desenvolvidos de maneira *ad-hoc* [Stankovic 1988] [Fidge e Lister 1992] [Oshana 2005]. Em geral, as fases iniciais do desenvolvimento de sistemas computacionais, como a análise de requisitos e o projeto da arquitetura eram negligenciadas em parte das abordagens existentes. O desenvolvimento concentrava-se nas fases de codificação dos programas - geralmente escritos em *Assembly* para microcontroladores. À medida que sistemas de tempo real foram se tornando mais complexos, seus desenvolvedores foram obrigados a utilizar conceitos de metodologias de projeto, como modularização, hierarquia, abstração e encapsulamento de informação, por exemplo [Kopetz 2011]. Além disso, existe a multi-disciplinaridade da equipe de projeto, em que estão envolvidas equipes de engenheiros de hardware, de engenheiros de software, de programadores e equipes responsáveis pelos testes e validações do sistema de tempo real. A metodologia de projeto utilizada deve permitir comunicação clara e uniforme entre os projetistas, de modo a minimizar os erros causados pela interpretação incorreta dos requisitos e funcionalidades do sistema.

Métodos convencionais de engenharia de software mostraram-se ineficientes para o desenvolvimento de sistemas de tempo real, uma vez que estes, além de não considerarem requisitos temporais, usualmente não abordam o projeto de hardware e software de forma integrada [Stankovic 1988] [Stewart 1999] [OMG 2011]. Segundo reportado em [Kemmerer e Ghezzi 1992], uma abordagem de desenvolvimento de software de tempo real deve contemplar tanto requisitos temporais quanto complexidade, além de auxiliar os desenvolvedores durante o ciclo de vida do sistema, incluindo a engenharia de requisitos, fase de projeto (*design*), implementação, testes e implantação.

Uma das etapas do processo de desenvolvimento de software de tempo real que precisa ser considerada é a modelagem de software. A modelagem de software é uma atividade importante na construção de um software de qualidade. Os modelos são construídos para representar e compreender a complexidade inerente a um software. Além disso, a modelagem ajuda a economizar custos na medida em que eventuais problemas podem ser descobertos antes da fase de construção. A modelagem de software de tempo real é importante porque erros em software de tempo real tem consequências graves, como acidentes e perdas financeiras. A modelagem surge como uma maneira de tentar minimizar os problemas no desenvolvimento de software de tempo real. No decorrer deste trabalho, é estudada e utilizada uma linguagem de modelagem específica para modelagem de software de tempo real.

1.2 Problema de pesquisa

Nesta seção, são descritos detalhadamente conceitos para entender o problema que esta pesquisa tenta amenizar. A Subseção 1.2.1 descreve a complexidade do desenvolvimento de software de tempo real e a Subseção 1.2.2 foca na complexidade da modelagem de software de tempo real.

1.2.1 Complexidade do software de tempo real

O desenvolvimento de software de tempo real é uma atividade complexa, desafiadora e passível de vários problemas não comuns ao desenvolvimento de software tradicional [Williams 2006]. Sabe-se que uma das principais atividades no processo de desenvolvimento de software é a definição e a especificação dos requisitos de software. Requisitos mal definidos e/ou mal especificados podem prejudicar o desenvolvimento de um projeto, implicando em maiores custos e prazos para a conclusão do projeto [Pressman 2010] [Sommerville 2010]. No desenvolvimento de aplicações de tempo real, a definição e a especificação de requisitos também é importante. Entretanto, é difícil realizar uma especificação bem definida destes requisitos. Primeiramente, o domínio de conhecimento das aplicações de tempo real é menos conhecido e explorado do que o domínio das aplicações de propósito geral [Jaffe et al. 1991] [Fidge e Lister 1992] [Bruda e Akl 2001]. Desta maneira, especificar precisamente os requisitos é mais difícil devido a carência de relatos, experiências anteriores na área e conhecimento pouco difundido [Bucci et al. 1995] [Kirner e Davis 1996] [Bruda e Akl 2001]. Em software comercial, a atenção aos requisitos não funcionais, como o desempenho, é um fator importante; em software de tempo real este fator é imprescindível.

Outro grande problema no desenvolvimento de software de tempo real é encontrar profissionais qualificados para trabalhar em aplicações críticas. A maioria dos programa-

dores está acostumado a trabalhar em aplicações comerciais tradicionais: *web*, *desktop* e dispositivos móveis. Não há abundância em mão de obra qualificada para trabalhar com aplicações críticas onde devem ser consideradas questões como desempenho, segurança, alocação de recursos, programação concorrente e tempo restrito [Brown e McDermid 2007].

O desenvolvimento de software de tempo real geralmente envolve o estudo e a aplicação de conhecimentos em diversas áreas. Por exemplo, a construção de um software controlador de voo envolve conhecimentos de computação, matemática, física, mecânica e engenharias. Além disso, o software de tempo real geralmente é constituído por uma grande quantidade de subsistemas interligados com o objetivo de formar o todo. A comunicação entre subsistemas diferentes pode ser uma barreira se os subsistemas forem desenvolvidos em tecnologias ou plataformas diferentes. Muitos destes subsistemas podem ainda ser desenvolvidos por empresas diferentes, tornando a comunicação mais complexa.

1.2.2 Conceitos de modelagem de software de tempo real

Assim como no desenvolvimento de aplicações de propósito geral, a modelagem de software é utilizada no desenvolvimento de software de tempo real. Entretanto, devido aos aspectos citados anteriormente, a modelagem de software de tempo real geralmente é mais complexa do que a modelagem de software tradicional [Douglass 2004].

O ambiente de tempo real incorpora outras características relevantes primordialmente neste tipo de sistema, como as restrições temporais, necessidade de gerenciar recursos limitados e críticos e a necessidade de modelar a infra-estrutura física do sistema, como processador e redes de comunicação [OMG 2011].

A utilização correta de linguagens de modelagem tem um papel importante no desenvolvimento de software. Se a linguagem de modelagem não for bem definida e não for capaz de produzir modelos expressivos e completos o suficiente, erros não serão descobertos e poderão ser propagados para a etapa de construção do software. Além disso, erros podem até mesmo ser criados na etapa de modelagem. Desta forma, a modelagem de software não atingirá o seu objetivo principal que é facilitar a construção do software e a atividade de modelagem - que é negligenciada por grande parte dos engenheiros de software [Forward e Lethbridge 2002] - não será utilizada de maneira sistemática no cenário do desenvolvimento de software de tempo real.

Existe na literatura várias abordagens utilizadas para modelagem de software de tempo real, como a análise estruturada [Ward e Mellor 1986] [Hatley e Pirbhai 1987] (Subseção 2.2.1), métodos formais [Woodcock et al. 2009] (Subseção 2.2.2), UML [OMG 2010b] (Subseção 2.2.3) e SPT [OMG 2005b] (Subseção 2.2.4). A análise estruturada é uma abordagem de modelagem pouco utilizada para sistemas modernos. As abordagens formais apresentam dificuldades na utilização, porque, dentre outros fatores, não foram bem recebidas pelos profissionais devido ao uso de notações lógico-matemáticas [Pressman

2010] [Sommerville 2010]. A UML *(Unified Modeling Language)*, linguagem de modelagem padrão para desenvolvimento de software, é utilizada frequentemente para modelagem de software de tempo real, apesar de não ser adequada para este fim. A UML tem vários problemas para modelar software de tempo real, como a dificuldade em modelar tempo, representação e alocação de recursos e representação do ambiente físico (Subseção 2.2.3). O *profile SPT (UML Profile for Schedulability, Performance, and Time)* foi criado como tentativa de superar os problemas da UML para modelagem de software de tempo real. Porém, não foi bem recebido pela comunidade de desenvolvimento de software, devido a seus inúmeros problemas citados na literatura (Subseção 2.2.4).

De acordo com o exposto no parágrafo anterior, percebe-se que não existe uma linguagem padrão para modelagem de software de tempo real. Um problema ainda maior é que as linguagens para modelagem de software de tempo real não conseguem modelar adequadamente os requisitos específicos desses ambientes.

Como tentativa de minimizar os problemas descritos anteriormente a OMG *(Object Management Group)* criou o MARTE *(UML Profile for Modeling and Analysis of Real-Time and Embedded Systems)* [OMG 2011] como tentativa de criar uma linguagem padrão para modelagem de software de tempo real.

1.2.3 Domínio da aplicação

As dificuldades no desenvolvimento e na manutenção de software de tempo real apresentadas anteriormente são válidas para sistemas controladores de sinais de trânsito. Os sistemas controladores de sinais de trânsito tem como características serem sistemas heterogêneos, legados, com incompatibilidade de hardware e software, de manutenção cara e lenta e com constantes alterações nas políticas de trânsito.

Um sistema heterogêneo é um sistema distribuído que contém diferentes tipos de hardware e software trabalhando juntos de maneira colaborativa para resolver problemas [Burback 1998].

Sistemas legados podem ser definidos como sistemas em uso por muito tempo, que atendem aos requisitos dos usuários e são de difícil substituição, porque a reimplementação de seu código é inviável financeiramente ou porque eles são imprescindíveis. Os sistemas legados resistem a modificações e evoluções. Um sistema legado tem vários problemas, dentre eles os mais sérios, segundo [Bisbal et al. 1999], são:

- Os sistemas legados geralmente são executados sobre hardware obsoleto, lento e caro de manter;
- A manutenção do software é geralmente cara. Falhas são dispendiosas e consomem tempo devido a falta de documentação e falta de entendimento generalizado do funcionamento interno do sistema;

- Esforços de integração são prejudicados pela ausência de interfaces claras;
- Sistemas legados são muito difíceis, se não impossíveis, de expandir.

Uma outra característica em sistemas controladores de sinais de trânsito são as constantes mudanças no tráfego, dependendo do momento do dia, do dia da semana ou em um período do ano [Wiering et al. 2004]. A utilização de múltiplas junções (interseções) torna o problema ainda mais complexo, pois o estado do semáforo influencia o fluxo do tráfego para outros semáforos. Devido a essa constante mudança no tráfego, existe a necessidade de mudanças nas políticas de trânsito, forçando os projetistas a encontrar soluções para os problemas do dia a dia do tráfego [Sirikijpanichkul et al. 2005].

A implantação de semáforos é uma atividade cara e lenta. As constantes mudanças das políticas de tráfego estão relacionadas a medidas de desempenho e financeiras dos sinais de trânsito [Koonce et al. 2008]. A manutenção de semáforos é lenta e também pode ocasionar problemas no fluxo normal do tráfego.

1.3 Metas e objetivos da pesquisa

O ponto chave do trabalho é avaliar experimentalmente a utilização do *profile* MARTE na modelagem de um software de tempo real. O MARTE foi aplicado em um estudo de caso na modelagem de um software de controle de transportes inspirado por [Laplante 2004], [Kutz 2004] e [Koonce et al. 2008]. A maioria dos trabalhos existentes na área segue uma abordagem apenas de engenharia de tráfego, com pouca preocupação em tranformar conceitos e teorias de tráfego em modelos de software a ser implementado. No trabalho mais próximo a este, de [Laplante 2004], o software é modelado utilizando a UML. Nesta pesquisa, o software foi modelado utilizando UML e as extensões presentes no MARTE para modelar os requisitos do problema descritos no Capítulo 6. De maneira geral, os principais objetivos deste trabalho são:

- Estudar os conceitos do *profile* MARTE com o objetivo de conhecer as suas principais características para modelar tempo, recursos, processos e outros atributos relacionados a software de tempo real;
- Aplicar o *profile* MARTE, junto com a UML 2.X, para modelagem de software de tempo real. O sistema de tempo real escolhido é um software de controle na área de transportes.
- Avaliar os benefícios trazidos pelo *profile*, as dificuldades encontradas na utilização do *profile* e sugestões de melhorias para novas versões;
- Propor uma arquitetura de referência em múltiplos níveis de abstração para modelagem de software de tempo real. A arquitetura proposta deve ser independente da linguagem de modelagem.

1.4 Metodologia

No decorrer deste capítulo, é apresentada a metodologia usada nesta pesquisa. Para isso, serão definidas questões de pesquisa, estratégias de pesquisa e instrumentos de pesquisa.

1.4.1 Questões de pesquisa

Para realizar os objetivos da pesquisa, o trabalho está focalizado em explorar a efetividade do MARTE para modelagem de software de tempo real. [Shaw 2003] propõe uma abordagem para elaborar questões de pesquisa (*RQ - Research Questions*) para trabalhos na área de engenharia de software. Segundo a classificação proposta pela autora, o tipo de questão deste trabalho está inserido no cenário de "projeto, avaliação ou análise de uma instância particular". Baseando-se nesta classificação, algumas questões de pesquisa foram formuladas:

RQ1 - Comparando com outros profiles *OMG, como a UML e o SPT, o profile MARTE representa melhor as características de software de tempo real ?*

Essa questão de pesquisa é respondida no Capítulo 4. A resposta para esta pergunta está relacionada a comparação realizada no capítulo. A comparação é realizada utilizando artigos como referência e a utilização do *profile* MARTE pelo autor.

RQ2 - A utilização de uma arquitetura de referência de software de tempo real representa modelos de maneira mais dequada do que a utilização de arquiteturas convencionais para representação de software de tempo real?

Essa questão de pesquisa é respondida no Capítulo 5. A resposta para esta pergunta é apresentada pela proposta de uma nova arquitetura de referência em múltiplas visões utilizando UML e MARTE para a representação de modelos de software de tempo real.

RQ3 - Os modelos gerados usando UML e MARTE são mais expressivos que os gerados usando apenas UML ?

Essa questão de pesquisa é respondida nos Capítulos 4 e 6, mas principalmente no Capítulo 6. A resposta para esta pergunta está relacionada aos diagramas gerados na utilização da UML e do MARTE e na descrição das extensões existentes no MARTE.

1.4.2 Estratégia de pesquisa

Dois paradigmas caracterizam a pesquisa em engenharia de software: ciência comportamental e ciência do *design* (*design science*) [Von Alan et al. 2004]. Em termos de objetivos, o paradimga ciência comportamental busca "*Qual é a verdade*", enquanto o paradigma ciência do *design* busca criar "*O que é efetivo*".

O paradigma ciência comportamental busca desenvolver e verificar teorias que explicam ou preveem comportamento humano ou de organizações. Esse paradigma tem suas

raízes em métodos de pesquisa de ciência natural. Os principais objetivos são desenvolver e justificar teorias que explicam ou preveem fenômenos nas organizações ou nos humanos sobre análise, projeto, implementação, gerenciamento e uso para sistemas de informação.

O paradigma ciência do *design*, paradigma seguido neste trabalho, tem suas raízes na engenharia [Denning 1997] [Tsichritzis 1997]. Esse paradigma é fundamentalmente utilizado para a resolução de problemas com ênfase no produto e nas soluções [Rossi e Sein 2003]. O paradigma ciência do *design* objetiva desenvolver conhecimento que possa ser usado por profissionais em seus campos de atuação para resolução de problemas [Van Aken 2005]. O termo ciência do *design* é escolhido para destacar a orientação do novo conhecimento ao *design* (de soluções de problemas do mundo real) e as ferramentas necessárias para ações adequadas de domínio dos profissionais.

É importante destacar a diferença entre a abordagem ciência do *design*, objeto da presente pesquisa, com as práticas de *design* de produtos, envolvendo protótipos com a função de pré-teste para averiguação da aceitação de inovações tecnológicas. Nestas últimas, utilizando-se de teorias psicossociais, a preocupação é predizer e explicar fenômenos organizacionais e humanos perante a introdução de inovações tecnológicas, tendo como foco a melhor aceitação e adoção da inovação pela futura comunidade de usuários. Na abordagem ciência do *design*, fundamentada em teorias epistemológicas, o objetivo é desenvolver *corpus* de conhecimentos orientados pelas práticas de implementação, gerenciamento e uso de artefatos [Wastell et al. 2009].

Em [Hevner et al. 2004], é definido um guia com orientações para aplicação da ciência do *design*. Este guia é apresentado na Tabela 1.1. Essas orientações derivam do princípio fundamental desse paradigma, na qual o conhecimento e o entendimento de um problema e sua solução são adquiridos pela construção e aplicação de um artefato.

Orientações	**Descrições**
A concepção como um artefato	A ciência do *design* deve produzir um artefato viável na forma de uma construção, modelo, método ou instanciação.
Relevância do problema	O objetivo da ciência do *design* é desenvolver soluções baseadas em tecnologia para problemas de negócio relevantes e importantes.
Avaliação	A utilidade, a qualidade e a eficácia de um artefato precisam ser rigorosamente demostrada através de métodos adequadamente executados.
Contribuições da pesquisa	Uma pesquisa eficaz, dentro do paradigma da ciência do *design*, precisa prover contribuições claras e verificáveis.
Rigor metodológico	A ciência de *design* reside na aplicação de métodos rigorosos para a construção e avaliação do artefato.
Processo de pesquisa	A busca por um artefato eficaz requer o uso de meios disponíveis para atingir os fins desejáveis, sem deixar de satisfazer as leis que regem o ambiente do problema.
Comunicação dos resultados	Os resultados devem ser apresentados adequadamente tanto para audiências orientadas à tecnologia como para audiências orientadas à gestão.

Tabela 1.1: Orientações para aplicação do ciência do *design* [Hevner et al. 2004]

Existe uma falta de consenso sobre o resultado desejado em um projeto de pesquisa.

Em muitos casos, a solução como modelo de engenharia de software é avaliada como boa o suficiente em termos de sua aceitação e não em termos da melhor solução. Os produtos resultantes, de acordo com [Hevner et al. 2004], incluem:

1. Construções ou conceitos: abstrações e representações que definem os termos usados na descrição de um artefato;
2. Modelos: abstrações e representações usadas para descrever e representar o relacionamento entre conceitos;
3. Métodos: conjunto de passos e práticas usadas para representar algoritmos, processos ou abordagens de como realizar uma certa tarefa;
4. Instanciações: sistemas implementados ou prototipados usados para realizar um artefato;
5. Teoria aperfeiçoada: a prática pode ajudar a um melhor entendimento dos relacionamentos, conceitos e processos [Rossi e Sein 2003].

Os produtos de trabalho utilizados nesta pesquisa são as construções, modelos e instanciações. As construções ou conceitos são referenciados na arquitetura de referência proposta (Capítulo 5). Os modelos são representados pelos diagramas UML adicionados pelas extensões do MARTE (Capítulo 6). Os métodos são representados pela sequência de passos dada pela modelagem de software de tempo real feita neste trabalho, dada pela arquitetura, modelagem estrutural e comportamental (Capítulos 4, 5 e 6).

1.4.3 Instrumentos de pesquisa

Segundo [Easterbrook et al. 2008], instrumentos de pesquisa são usados para coletar e posteriormente analisar dados para criar modelos, métodos ou teorias de uma estratégia de pesquisa. Os instrumentos de pesquisa utilizados dependem de muitos fatores como o tipo de questões de pesquisa utilizadas, o objetivo da pesquisa e a quantidade de teoria existente. Um conjunto de instrumentos de pesquisa foi aplicado durante este trabalho. Cada método é brevemente descrito a seguir.

A realização de uma fase de revisão da literatura em pesquisa é fortemente recomendada para obter conhecimento inicial sobre um assunto [Sjoberg et al. 2007]. Como primeira etapa deste trabalho, uma revisão extensiva da literatura foi realizada, com alvo principal no problema foco e nas diversas soluções para o problema foco. O próximo passo foi obter entendimento dos domínios (Engenharia de Software e Sinais de Trânsito).

Outro instrumento de pesquisa aplicado neste trabalho é o estudo de caso. Em [Yin 2008], o estudo de caso é definido como uma investigação experimental que analisa um fenômeno contemporâneo dentro do contexto da sua vida real, especialmente quando as fronteiras entre o fenômeno e o contexto não são claramente evidentes. Pesquisas

utilizando estudos de caso podem ser caracterizadas como qualitativas e observatórias [Yin 2008]. No lugar de exemplos, métodos de estudo de caso envolvem um exame profundo de uma instância ou evento. Quando selecionado com cuidado, até mesmo um único estudo de caso pode obter sucesso em termos da formulação teórica e testes [Yin 2008]. Isso pode acontecer porque se um caso crítico e abrangente for selecionado para testar uma teoria e se a teoria for suportada para esse caso, é provável que a teoria seja verdadeira para outros estudos de caso. Estudo de caso é o método de pesquisa preferido quando o "como" e o "por que" estão sendo investigados.

Segundo [Runeson e Höst 2009], o estudo de caso é um instrumento de pesquisa adequado para pesquisas em engenharia de software porque os objetos do estudo de caso são fenômenos contemporâneos difíceis de estudar isoladamente. Estudos de caso não geram os mesmos resultados que experimentos controlados geram, mas fornecem um entendimento profundo do fenômeno sobre estudo. Como estudos de caso são estudos experimentais controlados e analíticos, eles vem sendo criticados por supostamente ter menos valor e sendo impossível de generalizar. Esta crítica pode ser satisfeita através da aplicação de práticas de metodologia adequada de pesquisa bem como a reconsideração de que o conhecimento é mais do que a significância estatística [Lee 1989] [Flyvbjerg 2006].

1.5 Trabalhos relacionados

Os trabalhos mais próximos a este são os que tentaram realizar a modelagem de software de controle de interseção de tráfego através de sinais de trânsito. Máquinas de estados finitos foram usadas para modelar software de tempo real [Cassandras e Lafortune 2010], mas tem a deficiência da explosão de estados [Brave 1993]. *Statecharts* [Harel 1987] estendem máquinas de estados finitos dotando-as com ortogonalidade, profundidade e sincronização. [Huang 2006] usou *statecharts* para sinais de trânsito. Técnicas de inteligência artificial como agentes [Deng et al. 2005] [Srinivasan e Choy 2006] e lógica Fuzzy [Zeng et al. 2007] também foram usadas para modelar controle de sinais de trânsito. Redes de Petri foram aplicadas na pesquisa de controle de sinais de trânsito. [Lin et al. 2003] usaram Redes de Petri com foco em uma interseção, [Tolba et al. 2003] realizaram a avaliação e o desempenho na utilização de Redes de Petri no controle de sinais de trânsito e [Soares e Vrancken 2008] propuseram uma abordagem baseada na aplicação de Redes de Petri e prova de teorema.

Existem alguns trabalhos que utilizam a UML para modelar o controle de sinais de trânsito. Neste contexto, o trabalho de [Laplante 2004] é um dos mais importantes, sendo por isso inspirador desta pesquisa. Há também o trabalho de [Zhou et al. 2004] que combina a UML com arquiteturas de referências de sistemas no campo de manufaturas integradas a computador. É usado como estudo de caso um projeto piloto na China para desenvolvimento e projeto de uma metodologia para *ITS (Intelligent Transportation*

Systems). [Bonnefoi et al. 2007] apresenta o projeto e a especificação de soluções para *ITS* usando UML. [Arrayangkool e Unakul 2009] descreve uma abordagem para projetar e desenvolver arquiteturas ITS pelo uso de processo orientado a objetos e UML. No trabalho de [Ranjini et al. 2011], são analisados os problemas do controle de tráfego rodoviário e a UML é usada para modelar um controle de tráfego adaptativo que reage dinamicamente a eventos.

De acordo com a revisão bibliográfica feita, até o presente momento, não existem trabalhos na literatura que utilizam a UML juntamente com as extensões do MARTE para modelagem de software de controle de sinais de trânsito.

1.6 Visão geral da dissertação

Este trabalho contribui com a pesquisa e prática em Engenharia de Software propondo a utilizando da UML juntamente com o *profile* MARTE para modelagem de software de tempo real. O texto desta dissertação encontra-se organizado como indicado nos parágrafos a seguir.

No Capítulo 2, é feita uma breve revisão de conceitos, onde são apresentados conceitos de software de tempo real, sistemas embutidos, sistemas distribuídos e sistemas complexos. Além disso, é feita uma análise histórica das abordagens para modelagem de software de tempo real.

O Capítulo 3 faz uma análise dos principais conceitos relativos a sistemas de transportes relevantes para o estudo de caso aqui apresentado, focando principalmente em sinais de trânsito e sistemas de transportes inteligentes.

No Capítulo 4, o *profile* MARTE é apresentado detalhadamente. Além disso, o *profile* MARTE é comparado com outras linguagens propostas pela OMG para modelagem de software de tempo real, entre elas a UML e o SPT.

O Capítulo 5 propõe um modelo de arquitetura de software em múltiplas visões para ser aplicado no projeto de sistemas distribuídos de tempo real, no qual MARTE e UML são utilizados como linguagens de modelagem. Este capítulo visa propor uma arquitetura de referência para ser utilizada no desenvolvimento de software de tempo real.

O Capítulo 6 tem como principal objetivo detalhar a aplicação do *profile* MARTE juntamente com a UML na modelagem do sistema de controle de semáforos.

Finalmente, no Capítulo 7, são apresentadas as conclusões referentes ao trabalho, algumas contribuições propostas e também possíveis trabalhos futuros que podem ser realizados para dar continuidade a esta pesquisa.

Capítulo 2

Referencial teórico

Neste capítulo serão abordados conceitos importantes para o melhor entendimento das contribuições propostas nos próximos capítulos. A Seção 2.1 apresenta uma visão geral sobre os conceitos de sistemas de tempo real. A Seção 2.2 descreve as principais abordagens de modelagem de software utilizadas para modelagem de software de tempo real. Na Subseção 2.2.1, a análise estruturada com suas extensões para tempo real é descrita. Na Subseção 2.2.2, técnicas formais, como máquinas de estados finitos e redes de Petri, são apresentadas. A Subseção 2.2.3 apresenta a UML, suas principais características para modelagem de software de tempo real e seus problemas. A Subseção 2.2.4 explicita o *profile* SPT, com suas características e problemas. Por fim, a Subseção 2.2.5 apresenta uma visão geral do MARTE.

2.1 Conceitos de software de tempo real

De acordo com [Burns e Wellings 2001], um sistema de tempo real é qualquer sistema de processamento de informação que precisa responder a estímulos gerados externamente dentro de um período de tempo finito e especificado. A correção de um sistema de tempo real depende não somente do resultado lógico, mas também do tempo em que ele é entregue. Os sistemas de tempo real devem satisfazer explicitamente restrições de tempo de resposta sob o risco de ocorrência de danos severos, incluindo falhas e defeitos. Sistemas de tempo real geralmente não tem um visor e teclado tradicional, mas baseiam-se em um dispositivo aparentemente não computadorizado [Douglass 2004].

Um conceito importante no contexto de sistemas de tempo real é o tempo de resposta. O tempo de resposta [Laplante 2004] é o tempo entre a apresentação de um conjunto de entradas para um sistema (estímulo) e para a realização dos comportamentos exigidos (resposta), incluindo a disponibilidade das saídas associadas.

O software de tempo real sofre com os mesmos problemas que aparecem no desenvolvimento de software tradicional, mas adicionam uma dimensão extra: considerações sobre o tempo restrito [Williams 2006]. Desta forma, software para sistemas de tempo

real é considerado difícil de construir do que o software para sistemas *desktop* [Douglass 2004] [Williams 2006].

Os sistemas de tempo real são frequentemente reativos, embutidos, distribuídos e complexos [Laplante 2004]. Sistemas reativos são aqueles em que o agendamento de tarefas é dirigido pela sua interação com o ambiente. Por exemplo, um sistema de controle de incêndio reage quando o botão é pressionado.

O termo "sistema embutido" não possui uma definição clara e única [Henzinger e Sifakis 2007]. Segundo [Henzinger e Sifakis 2007], um sistema embutido é um artefato de engenharia que envolve computação, sujeito a um conjunto de restrições físicas, advindas do meio externo, que afetam a forma como essa computação deve ser executada e como essa deve reagir mediante solicitações externas.

Um sistema distribuído é uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente [Tanenbaum e Steen 2006]. Os computadores interconectados permitem o compartilhamento dos recursos do sistema, como por exemplo hardware, software e dados [Coulouris et al. 2011].

Para [Rouse 2003], um sistema complexo é um sistema cuja complexidade pode ser atribuída às seguintes caraterísticas: grande número de elementos, grande número de relacionamento entre os elementos, relacionamentos descontínuos, relacionamentos não-lineares e características incertas dos elementos e dos seus relacionamentos. O autor propõe algumas abordagens para tratar a complexidade. A abordagem adequada a este trabalho é analisar e modelar um sistema complexo pela decomposição hierárquica em componentes menores (dividir para conquistar). A divisão hierárquica em componentes de um problema complexo focaliza em dividir um problema complexo em partes menores e, possivelmente, menos complexas. A solução do problema complexo é feita pela resolução (e integração) das partes menores e menos complexas.

No contexto de sistemas de tempo real, existem três classificações principais. A classificação é feita considerando o quão rigorosos ou tolerantes os sistemas são em relação ao cumprimento dos requisitos temporais [Shaw 2000]. A classificação divide os sistemas em sistemas de tempo real *soft*, sistemas de tempo real *hard* e sistemas de tempo real *firm*.

Segundo [Burns e Wellings 2001], um sistema de tempo real *soft* é aquele em que o prazo para a entrega é importante, mas o sistema continuará funcionando corretamente se o prazo não for atendido. De acordo com [Laplante 2004], um sistema de tempo real *soft* é aquele em que o desempenho degradado não acarretará grandes perdas devido ao não cumprimento às restrições do tempo de resposta. Exemplos típicos são sistemas de comutação telefônico e sistema de aquisição de dados.

Segundo [Burns e Wellings 2001], um sistema de tempo real *hard* é aquele em que é absolutamente obrigatório que o tempo de resposta ocorra dentro do prazo exigido. De acordo com [Laplante 2004], um sistema de tempo real *hard* é aquele em que o não cumprimento do prazo pode levar a uma falha que pode ser financeira ou ocasionar mortes.

Exemplos típicos são sistemas de controle de vôo, sistemas de sinalização de ferrovias e sistemas de controle de uma planta nuclear.

Segundo [Burns e Wellings 2001], um sistema de tempo real *firm* é aquele sistema que é *soft*, mas a perda de uma quantidade muito grande pode provocar falhas completas do sistema ou até mesmo catástrofes. De acordo com [Laplante 2004], um sistema de tempo real *firm* é aquele em que pequenos atrasos não levam a falha total, mas atrasos maiores podem levar a uma falha catastrófica e completa do sistema. Por exemplo, um caixa eletrônico e um sistema de vendas de passagens aéreas.

O projeto e a implementação de sistemas de tempo real requer atenção a vários aspectos [Laplante 2004], como: a seleção do hardware e do software e avaliação da solução; especificação e projeto; representação correta do seu comportamento temporal; entendimento das nuances da linguagem de programação e as implicações resultantes do tempo real; tolerância a falhas e confiabilidade do sistema através de um projeto cuidadoso e; medir e prever o tempo de resposta para reduzi-lo, entre outros.

Segundo [Williams 2006], a maior parte dos sistemas de tempo real são projetados e implementados com ferramentas de propósito geral de verificação e de implementação. Entretanto, as necessidades de segurança num número cada vez maior de aplicações e a ligação dessa com a correção temporal desses sistemas conflitam com as metodologias e ferramentas convencionais, sob pena de perdas em termos financeiros, ambiental e humano.

Apesar da evolução nos últimos anos, em termos de conceitos e métodos para tratar a problemática de sistemas de tempo real, a adoção no setor produtivo de novos algoritmos, suportes e metodologias não ocorrem no mesmo ritmo. Na prática, o uso de meios de propósito geral continuam, mesmo no caso de aplicações críticas, o que pode ser a causa de muitas situações desastrosas [Williams 2006].

O problema de tempo real consiste em especificar, verificar e implementar sistemas ou programas que, mesmo com recursos limitados, apresentam comportamentos previsíveis, atendendo as restrições temporais impostas pelo ambiente ou pelo usuário [Joseph 1992].

2.2 Visão geral da modelagem de software de tempo real

Uma abordagem comum para o desenvolvimento de software é fazer a modelagem do problema e solução antes de sua construção. Segundo [Booch et al. 2005], a modelagem é uma parte central das atividades que levam à implantação de um bom software. Os modelos são construídos para comunicar estrutura e comportamentos desejados do sistema, para visualizar e controlar a arquitetura do sistema e para compreender melhor o sistema sendo elaborado. Um modelo é uma simplificação da realidade. Engenheiros de

software utilizam modelos para simplificar a realidade porque não é possível compreender os aspectos de sistemas complexos em sua totalidade [Booch et al. 2005].

Para a criação de modelos de tempo real é necessário modelar não só a estrutura e o comportamento do software, mas também os recursos físicos e lógicos, sua infra-estrutura e os aspectos temporais [Selic 1999]. Essas características distinguem claramente a modelagem de sistemas de tempo real da modelagem de outros sistemas.

Nos últimos anos, muitas notações e métodos na engenharia de software de tempo real foram propostos. Entretanto, ainda não existem muitas avaliações sobre esses métodos. Linguagens de especificação representativas e expressivas tornaram-se essenciais, devido à complexidade do software. Uma linguagem de especificação pode ajudar a melhorar o projeto e gerenciar o desenvolvimento de sistemas de software complexos [Lee 2002].

Algumas abordagens de modelagem surgiram ao longo das últimas décadas para modelar software de tempo real. Entre as representações mais conhecidas na literatura estão a análise estruturada com extensões para tempo real, as abordagens formais e as abordagens orientadas a objetos. A abordagem estruturada é representada, por exemplo, pelas notações Hatley e Pirbhay [Hatley e Pirbhai 1987] e Ward e Mellor [Ward e Mellor 1986]. Dentre as técnicas formais, destacam-se autômatos [Hopcroft et al. 1979], as Redes de Petri [Murata 1989], os Statecharts [Harel 1987] e a notação Z [Spivey 1989] [Woodcock e Davies 1996]. As abordagens orientadas a objetos são representadas principalmente pela UML [OMG 2010b] e suas extensões, SPT [OMG 2005b] e MARTE [OMG 2011]. Os tópicos descritos serão abordadados com mais detalhes nas subseções seguintes.

2.2.1 Análise estruturada para sistemas de tempo real

A análise estruturada é um meio de tentar analisar sistemas utilizando notações gráficas, abordagem *top-down*, métodos de decomposição funcional e algoritmos. A análise estruturada trata apenas os aspectos de análise que podem ser estruturados, como as especificações funcionais [Yourdon 1989]. Maiores detalhes sobre a análise estruturada são encontrados em [Gane e Sarson 1977], [Yourdon e Constantine 1979], [DeMarco 1979], [Peters 1987] e [Yourdon 1989].

Além das abordagens tradicionais da análise estruturada, existem as extensões da análise estruturada para sistemas de tempo real que consideram as características específicas desse ambiente, como as restrições temporais. Em uma significativa porcentagem das aplicações de tempo real, o sistema deve monitorar informações em tempo contínuo geradas por processos do mundo real. A notação convencional para o fluxo de dados não faz distinção entre dados discretos e dados em tempo contínuo [Ward e Mellor 1986].

Existem algumas extensões conhecidas para sistemas de tempo real. As extensões da análise estruturada propostas por Hatley/Pirbhay ampliam a notação básica da análise estruturada. Suas extensões concentram-se menos na criação de símbolos gráficos adici-

onais e mais na representação e na especificação dos aspectos orientados ao controle do software. Para isso, é definido um diagrama de fluxo de controle *(CFD - Control Flow Diagram)*, que apresenta os mesmos processos que o diagrama de fluxo de dados *(DFD - Data Flow Diagram)*, mas com foco no fluxo de controle e não o fluxo de dados.

A especificação de controle *(CSPEC - Control Specification)* representa o comportamento do sistema de duas maneiras diferentes. A CSPEC contém um diagrama de transição de estado *(STD - State Transition Diagram)* que consiste em uma especificação sequencial do padrão de comportamento. Ela também pode conter uma tabela de ativação de programa *(PAT - Program Activation Table)* - uma especificação combinatória de comportamento. A CSPEC descreve o comportamento do sistema, mas não oferece quaisquer informações sobre o funcionamento interno dos processos ativados como resultado desse comportamento. A especificação de processos *(PSPEC - Process Specification)* é usada para descrever os processos do DFD que aparecem no nível de refinamento final. Ao definir uma PSPEC para cada processo do modelo de fluxo, o engenheiro de software cria uma especificação que pode servir como um primeiro passo na criação da especificação de requisitos de software e como um guia para projeto do componente de programa que implementará o processo.

Existem alguns trabalhos relevantes para a modelagem de tempo real utilizando as notações de Hatley/Pirbhay. No seu trabalho sobre análise de técnicas para representação de concorrência, [Sanden 1989] conclui que as abordagens de Hatley/Pirbhay não são adequadas para representar aspectos de concorrência. [Peters 1989] propõe um meio para representar características de tempo na análise estruturada de Hatley/Pirbhay. [Tatsuta 1995] propõe uma maneira de verificar consistência e completeza em modelos gerados pela abordagem de Hatley/Pirbhay. [Mathalone 1997] compara a abordagem de Hatley/Pirbhay com a metodologia BBM *(Behaviorally-Based Methodology)*. BBM é uma metodologia para especificar requisitos de sistemas complexos. A BBM mostrou algumas melhorias na especificação para sistemas complexos não presentes em Hatley/Pirbhay.

As extensões da análise estruturada propostas por Ward/Mellor ampliam a notação básica da análise estruturada nas especificações de fluxo de informação obtida ou produzida em tempo real [Ward e Mellor 1986].

Existem alguns trabalhos relevantes para a modelagem de tempo real utilizando as notações de Ward/Mellor. [Kelly 1987] analisa quatro técnicas para a modelagem de software de tempo real. O autor conclui que a abordagem de Ward/Mellor tem problemas como a fraca reusabilidade e a dificuldade de se adaptar a mudanças. As vantagens estão relacionadas à técnica oferecer uma boa visão geral do sistema. [Nielsen e Shumate 1987] fazem um estudo da linguagem ADA e concluem que a técnica de Ward/Mellor é adequada para projetos utilizando esta linguagem para grandes software de tempo real. Em uma análise feita em [Adler 1988], o autor conclui que a abordagem de Ward/Mellor não suporta decomposição de diagramas em diferentes níveis de abstração. Finalmente, os tra-

balhos de [Khalsa 1989] e [Alabiso 1988] mostram a transformação da análise estruturada proposta por Ward/Mellor para representações orientada a objetos.

Além dos anteriormente citados, existem outros problemas encontrados nas abordagens da análise estruturada para tempo real. Segundo [Bray 2002], os modelos são essencialmente baseados em processos (incentivam o modelo estrutural do problema existente). Existe também a dificuldade na integração de DFDs e diagramas entidade relacionamento *(ERD - Entity Relationship Diagrams)*. Além disso, as extensões da análise estruturada para tempo real são falhas em não fazer uma menção específica a requisitos. Complementando, as dificuldades comuns para modelagem de tempo real, como a especificação comportamental, alocação de recursos, tempo, propriedades não-funcionais e concorrência permanecem [Neill e Laplante 2003].

2.2.2 Métodos formais

Métodos formais usam modelos lógico-matemáticos para modelagem e verificação do ciclo de vida de um software [Woodcock et al. 2009]. Um método é formal se ele tem alguma base lógico-matemática, tipicamente dado por uma linguagem de especificação formal. Essa base matemática fornece um meio de definir precisamente noções como consistência e completeza e, mais relevantemente, especificação, implementação e correção [Wing 1990] [Plat et al. 1992]. O uso dos métodos formais não garante um sistema correto [Clarke e Wing 1996], entretanto, eles podem ajudar no entendimento do sistema revelando inconsistências, ambiguidades e incompletudes que poderiam não ser entendidas sem o uso dos métodos formais.

O rigor matemático habilita o usuário a analisar e verificar esses modelos em qualquer parte do ciclo de vida do software. Métodos formais de desenvolvimento de software recebem muita atenção nos centros de pesquisa [Plat et al. 1992] [Fantechi e Ferrari 2009]. Na indústria, a sua aplicabilidade é frequentemente debatida, apesar de vários casos de sucesso anteriores [Larsen et al. 1996] [Bowen e Hinchey 2006] [Abrial 2007] [Woodcock et al. 2009]. Alguns trabalhos mostrando a utilização dos métodos formais para modelagem de software de tempo real serão brevemente descritos a seguir.

Uma máquina de estados finitos (*FSM - Finite State Machine*) ou autômato de estados finitos (*FSA - Finite State Automaton*) [Hopcroft et al. 1979] é um modelo matemático formal usado na especificação e no projeto de uma ampla gama de sistemas. Existem algumas aplicações para modelagem de software de tempo real utilizando FSM. No trabalho [Clarke et al. 1986], é mostrada uma maneira de verificar se um autômato atende às especificações de sistemas concorrentes expressas em lógica temporal. [Shaw 1992] faz um uso abrangente das FSM para modelagem. Foi criada uma notação para especificar sistemas de tempo real incluindo o ambiente físico controlado e monitorado. O sistema foi denominado *CRSM (Communicating real-time state machines)*. O CRSM fornece algu-

mas propriedades de tempo e acesso em tempo real. Em [Cheng e Krishnakumar 1993], é mostrada uma extensão dos autômatos para modelar circuitos integrados. O trabalho [Lee e Yannakakis 1996] executa testes sobre as FSM para garantir o funcionamento correto dos sistemas e para descobrir aspectos do seu comportamento.

Os *statecharts* [Harel 1987] são uma ampla extensão do formalismo convencional das máquinas de estado. O *statechart* estende o diagrama de transição de estados tradicional com três elementos: tratamento de hierarquia, concorrência e comunicação. Os *statecharts* são usados extensivamente. Em [Bianco et al. 2002], é feita uma extensão da modelagem de *statecharts* para sistemas de tempo real. O trabalho definiu um conjunto de extensões para modelagem de sistemas de tempo real e ofereceu uma semântica para essas extensões em termos de *statecharts* temporais. O trabalho [Huszerl et al. 2000] apresenta um método que permite a análise quantitativa de confiança do sistema usando os *statecharts*. Outro trabalho relevante na aplicação de *statecharts* para sistemas de tempo real é encontrado em [Giese e Burmester 2003], em que são modeladas duas linhas representacionais em *statecharts*, a linha de expressividade e a linha de comportamento temporal. O produto do trabalho é a criação de um *statechart* chamado *Extended Hierarchical Timed Automaton*. [Whittle e Schumann 2002] mostram um estudo de caso em um departamento da NASA, onde os *statecharts* são utilizados no domínio de controle de tráfego aéreo. Em [Burmester e Giese 2003], é desenvolvida uma extensão para a ferramenta *Fujaba* [Nickel et al. 2000] que supera as limitações dos *statecharts* por suportar uma semântica bem definida baseada em autômatos temporais e uma síntese de código que garante as características temporais especificadas. O autores concluem que *statecharts* de tempo real fornecem uma técnica de modelagem que permite especificar o comportamento de sistemas complexos. Além disso, os *statecharts* contém informações necessárias para realizar a geração de código a partir do modelo. Finalmente, em [Tichy e Kudak 2003] foi desenvolvida uma extensão para a ferramenta *Fujaba* [Nickel et al. 2000] para suportar monitoramento da execução de *statecharts* em que é feito um *framework* para a visualização do comportamento de sistemas embutidos. Esse monitoramento de execução é necessário porque muitos sistemas de tempo real não têm uma interface amigável que mostra o status atual do sistema.

As Redes de Petri (RdP) [Murata 1989] são uma ferramenta de modelagem gráfica e matemática aplicável a muitos sistemas, principalmente na descrição e estudo de sistemas concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos ou estocásticos. Embora exista uma ampla divulgação acadêmica ao longo das últimas décadas sobre o uso e vantagens das RdP, o seu potencial foi reconhecido no início da década de oitenta do século passado, em que esta teoria foi usada para implementações práticas nas áreas de sistemas de manufatura devido à disponibilidade de novos recursos de hardware e software [PAIS 2004]. Em [Du et al. 2007], os autores propõem uma rede lógica para modelar e analisar os sistemas de tempo real cooperativos. O modelo é baseado em RdP, técnicas de *workflow* e lógica temporal. Comparando com as abordagens tradicionais, a aborda-

gem proposta fornece mais funções, melhor semântica e pode modelar sistemas de tempo real de maneira mais elegante. No trabalho de [Dotoli et al. 2008], é discutida a identificação do problema de sistemas a eventos discretos usando um procedimento de RdP. Para resolver o problema, são considerados dois casos na utilização de RdP: no primeiro caso, o lugar e a transição são assumidos conhecidos; no segundo caso, os lugares e as transições são assumidos desconhecidos. [Liu et al. 2009] formulam uma RdP orientada a objetos de tempo real estendida para modelar desempenho e simular o ambiente de *Semiconductor wafer fabrication system (SWFS)*. A criação de um protótipo neste estudo mostrou que as redes de Petri conseguiram modelar com sucesso os cenários exigidos.

A notação Z é um método formal proposto por Jean-Raymond Abrial em 1977 [Spivey 1989] e teve seu desenvolvimento realizado no laboratório de computação da Universidade de Oxford. Esta notação é baseada na lógica de primeira ordem e na teoria de conjuntos de Zermelo-Fraenkel [Spivey 1989] [Woodcock e Davies 1996]. Existem algumas aplicações da notação Z na literatura. Em [Smith e Hayes 2000], é usado o *Object-Z*, uma extensão orientada a objetos da linguagem Z. Os autores mostram como a construção de classes em *Object-Z* pode ser usada para estruturar especificações em tempo real. No trabalho de [Kumar e Goel 2010], é modelado um exemplo de *Automated Teller Machine (ATM)* com as suas funcionalidades básicas: sacar, consultar, depositar e transferir. A linguagem Z junto com a ferramenta utilizada são suficientes para construir o modelo conceitual e o modelo comportamental existente em um sistema ATM.

As principais vantagens dos métodos formais são a capacidade de eliminar problemas comuns encontrados em outros modelos não-formais, como a ambigüidade, inconsistência e incompletude [Wing 1990] [Plat et al. 1992] [Clarke e Wing 1996] [Gogolla 2004]. Os métodos formais também servem de base para verificação de software. Além disso, são apropriados para sistemas críticos (por exemplo, aeronaves e dispositivos médicos). Quando aplicados nas primeiras fases do desenvolvimento de software, eles podem ajudar a diminuir os problemas propensos neste momento, como a imprecisão de requisitos.

Apesar das muitas vantagens, os métodos formais também apresentam algumas desvantagens. Desenvolvedores consideram métodos formais inadequados, restritos a sistemas críticos, muito caros, insuficientes, difíceis e não práticos [Broy 2006]. Embora métodos formais possam reduzir erros na tradução dos requisitos para o código fonte, eles não podem garantir que erros e inconsistências nos requisitos não existam [Vrancken et al. 2008]. A dificuldade no aprendizado dos métodos formais também é altamente debatida. [Hall 1990] considera que a teoria matemática dos métodos formais é adequada para engenheiros de software e seu treinamento não é difícil. Para [Abrial 2006], a experiência mostra que engenheiros de software podem aprender facilmente os conceitos matemáticos e as notações usadas nos métodos formais. Por outro lado, de acordo com [Goguen 1997], métodos formais são difíceis de ser utilizados pela maioria dos programadores, que tem pouco treinamento ou habilidade em técnicas avançadas de matemática.

2.2.3 UML

Atualmente, o paradigma mais utilizado para modelagem de software é o paradigma orientado a objetos (OO) [Booch et al. 2007]. Diversos autores tem proposto a utilização de técnicas orientadas a objetos no projeto de hardware e de software de sistemas de tempo tempo real [Stoel e Karrfalt 1995] [McUmber e Cheng 1999] [Sinha et al. 2000] [Axelsson 2000] [Lavazza et al. 2001] [Björklund e Lilius 2002]. Existem diversas notações de modelagem OO [Rumbaugh et al. 1991] [Jacobson 1992] [Booch 1994], entre elas, a mais usada é a UML [OMG 2010a], sendo considerada um padrão *de facto* para modelagem de software [Fowler 2003] [Booch et al. 2005] [Selic 2006].

A UML é amplamente usada para expressar modelos de propósito geral em engenharia de software e apresenta algumas características que suportam aspectos de tempo real [Berkenkotter 2003] [Gérard e Terrier 2003]. Por exemplo, a UML suporta modelagem de concorrência utilizando conceitos como objetos ativos, composição de estados concorrentes e operações concorrentes. Para expressar restrições temporais, a UML fornece dois tipos de dados: *Time* e *TimeExpression*. Essas asserções podem ser usadas tanto no diagrama de estados como no diagrama de sequências. Além disso, a UML 2.0 apresenta um novo diagrama chamado Diagrama de Tempo que permite modelar tempo ou mudanças de estados no tempo.

A UML apresenta muitas vantagens e é bastante usada na modelagem de software de tempo real, tanto na academia quanto na indústria [Soares e Vrancken 2009]. [Arnold et al. 2003] propõem uma nova abordagem de desenvolvimento para robôs industriais, com foco na modelagem do problema, no lugar do foco no robô em si. Em [Zoughbi et al. 2007], a UML é usada para a construção de um *profile* para modelagem de aplicações aeroespaciais. No trabalho de [Toma et al. 2007], a UML é usada no cenário médico para a modelagem de *workflows* em um ambiente de cirurgia maxilo-facial. [Vepsalainen et al. 2008] mostram a criação e a aplicação de um *profile* UML para a utilização na automação de manufaturas.

Entretanto, diversos autores identificaram que a UML apresenta dificuldades para modelar software de tempo real. A UML não consegue representar adequadamente questões relacionadas a sistemas de tempo real como escalonamento [Douglass 2004] e gerenciamento de tarefas [Berkenkotter 2003]. A representação de tempo na UML é pobre [Graf et al. 2006] e os seus diagramas não são suficientes para descrever comportamentos complexos de um sistema [Lee 2002]. Diagramas comportamentais, como o diagrama de sequência, não podem representar restrições de tempo efetivamente porque são não-temporais, expressando apenas a ordem cronológica na troca de mensagens entre os objetos [André et al. 2007].

2.2.4 SPT

Para resolver as dificuldades existentes na UML para modelagem de software de tempo real foi criado o *profile SPT (UML Profile for Schedulability, Performance, and Time)* [OMG 2005b]. O SPT foi criado como uma forma de modelar conceitos de tempo real em UML. O *profile* estende a UML fornecendo estereótipos e valores rotulados para representar requisitos de desempenho, recursos e comportamentos usados do sistema.

SPT é um *framework* para modelar qualidade de serviço, recursos, tempo e conceitos de concorrência para suportar os modelos UML, fornecendo ao usuário um conjunto de estereótipos e valores rotulados para anotar modelos UML. A estrutura do SPT, como mostrada na Figura 2.1, é composta de *sub-profiles* [OMG 2005b]:

O pacote *General Resource Model* (GRM) é um pacote núcleo do SPT. É dividido em três pacotes:

- *RTresourceModeling*: para os conceitos básicos de qualidade de serviço e recursos;
- *RTconcurrencyModeling*: para modelagem de concorrência;
- *RTtimeModeling*: para modelagem de mecanismos de tempo.

O pacote *Analysis Modeling* define *sub-profiles* de análise e inclui:

- *PAprofile*: para análise de desempenho;
- *SAprofile*: para análise de escalonamento.

Para modelagem de tempo, o *sub-profile RTtimeModeling* define um modelo para representação do tempo. Este *sub-profile* é formado por um conjunto de estereótipos e propriedades associadas ao modelo que aplicam elementos na UML para especificar valores de tempo como o *<<RTtime>>*, mecanismos de relacionamento de tempo como *<<RTclock>>*, *<<RTtimer>>*, *<<RTtimeout>>* ou restrições temporais como por exemplo *<<RTdelay>>* e *<<RTinterval>>* [OMG 2005b]. A utilização das extensões do SPT na modelagem de software real é realizada de maneira semelhante a modelagem do sistema controlador de sinais de trânsito realizada no Capíutlo 6.

Existem algumas utilizações do SPT. Em [Xu et al. 2003], é mostrado o uso do *profile* SPT para descrever aspectos de desempenho em um projeto. Para isto, é feita uma análise sobre os requisitos não-funcionais e de desempenho (tempo de resposta, capacidade e escalabilidade). No trabalho de [Addouche et al. 2004], é apresentado um *profile* UML chamado DAMRTS *(Dependability Analysis Models for Real-Time Systems)* representando uma extensão ao SPT. Esse trabalho fornece conceitos para especificar sistemas de tempo real com informações estocásticas e probabilísticas, permitindo análise de confiança. Em [Thramboulidis 2004], o SPT é usado no cenário de controle e automação, em que o SPT é aplicado em um ambiente onde é adotada uma abordagem dirigida por modelos. [Bennett e Field 2004] mostram a aplicação de uma metodologia de engenharia

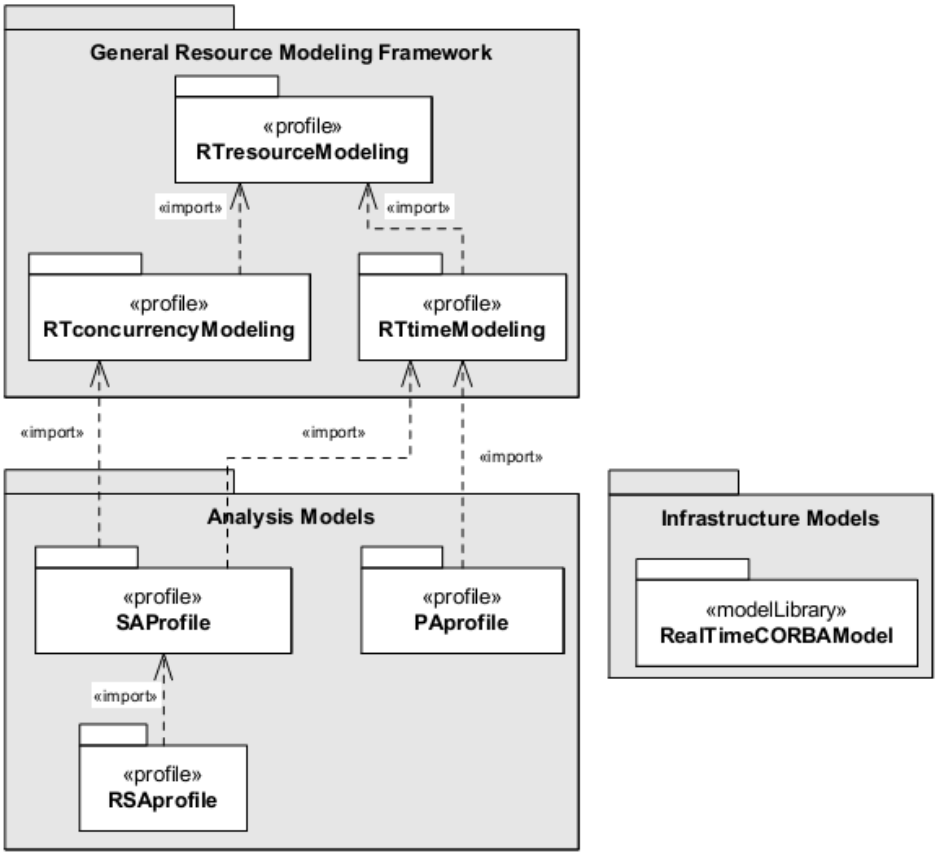

Figura 2.1: O *profile* SPT [OMG 2005b]

de desempenho baseada no SPT em um estudo de caso no cenário de telecomunicação móvel.

Apesar das melhorias trazidas pelo SPT, como a faciliade de modelar recursos compartilhados e propriedades não-funcionais, existem alguns problemas citados na literatura. De acordo com [OMG 2007], várias melhorias foram requeridas para o SPT, entre elas: a modelagem de plataformas de hardware e software, modelagem de tempo lógico, Engenharia de Software Baseada em Componentes *(CBSE - Component Based Software Engineering)*, alinhamento com a *UML 2.0*, Qualidade de Serviço e Mecanismos e Características de Tolerância a Falhas*(QoS&FT - Quality of Service and Fault Tolerance Characteristics and Mechanisms)* e Desenvolvimento Dirigido por Modelos *(MDA - Model Driven Architecture)*. Além disso, as construções do SPT foram consideradas abstratas e difíceis de usar [OMG 2007].

Embora o *profile* UML para modelar qualidade do serviço, características e mecanismos de tolerância a falhas, *(QoS&FT - UML Profile for Quality of Service and Fault Tolerance specification)*, define um *framework* para expressar um conceito similar a propriedades não-funcionais, existem algumas razões para definir um novo no contexto da especificação de tempo real do SPT [OMG 2011].

SPT forneceu um mecanismo direto, usando anotações, para propriedades não-funcionais *(NFPs - Non-functional properties)*. Entretanto, sua abordagem não está definida for-

malmente para permitir NFPs definidos pelos usuários ou para diferentes domínios especializados. Na verdade, SPT define uma gramática para conceitos, como por exemplo expressões de *"RTimeValue"*, mas não define um mecanismo para estender ou refinar essas construções para necessidades mais específicas [OMG 2011].

O *sub-profile* do SPT para escalonabilidade e análise é definido de maneira muito independente, reduzindo a habilidade de reusar modelos anotados para qualquer tipo de análise [OMG 2011].

Uma aplicação pode usar o tempo de duas maneiras: como uma referência a um instante de tempo ou como um período de tempo [OMG 2011]. O SPT não oferece um meio para tratar tempo lógico.

Outro problema relevante no SPT é relativo a data do seu lançamento. O *profile* foi lançado em sua primeira versão em setembro de 2003 [OMG 2003], ainda relativo à UML 1.X. A UML 2.0 foi lançada em 2005 [OMG 2005a]. Desenvolvedores podem ter começado a utilizar o SPT antes do lançamento da nova versão da UML e ainda não terem migrado de versão [OMG 2005b]. Assim, esses desenvolvedores podem estar utilizando um *framweork* desatualizado com o metamodelo da UML 2.0.

2.2.5 MARTE

A partir dos problemas encontrados no SPT (subseção 2.2.4) um novo *profile* UML foi proposto: *MARTE (Modeling and Analysis of Real-Time and Embedded systems)* [OMG 2011], com foco na UML 2.0. MARTE é uma evolução do SPT, especializando a UML ao adicionar conceitos para modelar e analisar sistemas embutidos de tempo real. MARTE fornece conceitos necessários para descrever caracterísitcas que especificam a semântica de sistemas de tempo real em diferentes níveis de abstração.

MARTE consiste na definição de fundamentos para uma descrição baseada em modelos de sistemas embutidos de tempo real. Os principais conceitos são refinados para interesses de modelagem e análise. As partes de modelagem fornecem suporte necessário para uma especificação de um projeto de sistemas embutidos de tempo real desde a especificação até o projeto. MARTE também é útil na análise baseada em modelos. Neste sentido, o objetivo não é definir novas técnicas para analisar sistemas embutidos de tempo real, mas suportá-las. Desta forma, MARTE fornece facilidades para anotar modelos com informações necessárias para realizar uma análise específica.

Entre outros, os benefícios de usar o *profile* são [OMG 2011]:

- Fornecer um meio comum de modelar tanto aspectos de hardware como aspectos de software em sistemas embutidos de tempo real com objetivo de melhorar a comunicação entre desenvolvedores;
- Habilitar a interoperabilidade entre ferramentas de desenvolvimento usadas para especificação, projeto, verificação e geração de código;

- Favorecer a construção de modelos que podem ser usados para fazer predições quantitativas considerando características específicas de sistemas embutidos de tempo real, tanto aspectos de hardware quanto aspectos de software.

O *profile* MARTE é descrito mais detalhadamente na Seção 4.1.

Capítulo 3

Descrição do domínio de aplicação

Este capítulo discorre sobre os principais conceitos utilizados ao longo deste trabalho, como a importância e os problemas dos sistemas de transportes para a sociedade, a utilização de semáforos e as políticas de sinais de trânsito. Na Seção 3.1, os sistemas de transportes inteligentes são discutidos. Na Seção 3.2, são apresentados os problemas e as necessidades para utilização de sinais de trânsito e seus conceitos básicos, bem como o seu histórico. Na Seção 3.3, algumas políticas de sinais de trânsito são discutidas brevemente.

3.1 Sistemas de transportes inteligentes

O objetivo deste trabalho não é detalhar os conceitos de transportes e sinais de trânsito. O exposto neste capítulo foi utilizado como base para a modelagem do software controlador de interseções apresentado no Capítulo 6. Para maiores detalhes sobre o assunto, recomenda-se pesquisa aprofundada em bibliografia especializada, incluindo "*Traffic Control Systems Handbook, Federal Highway Administration*" [FHWA 1996], "*Handbook of transportation science*" [Hall 1999], "*Traffic Engineering*" [Roess et al. 2004], "*Handbook of Transportation Engineering*" [Kutz 2004], "*Perspectives on intelligent transportation systems (ITS)*" [Sussman 2005], "*ITS America*" [ITSAmerica 2007] e "*Traffic Signal Manual*" [Koonce et al. 2008].

Os Sistemas de Transportes Inteligentes (*ITS - Intelligent Transportation Systems*) tem por objetivo a otimização dos transportes em geral, i.e., a otimização da estrutura envolvida no transporte de pessoas e bens fazendo uso de uma vasta gama de tecnologias para a criação de vias de comunicação, veículos e vias mais inteligentes, aumentando a fluidez no transporte de pessoas e mercadorias [Figueiredo 2005].

Existem diversas definições para o conceito de ITS. Para [ITSAmerica 2007], os ITS englobam uma vasta área de tecnologias de informação, comunicações e controle. Estas, quando integradas nas infra-estruturas dos sistemas de transportes e nos próprios veículos, ajudam a monitorar o fluxo de tráfego, reduzir congestionamentos, sugerir alternativas aos viajantes, aumentar a produtividade, salvar vidas, poupar tempo e dinheiro, reduzindo

os impactos na saúde e no ambiente. [Kutz 2004] define, de maneira correlata, os ITS como um fenômeno global emergente, que utiliza uma gama de tecnologias aplicadas aos transportes para tentar salvar vidas, dinheiro e tempo. As tecnologias envolvidas incluem microeletrônica, comunicação, tecnologia da informação e disciplinas como engenharia de transportes, telecomunicações, ciência da computação, finanças, eletrônica, comércio e manufatura automotiva. Outras definições correlatas podem ser encontrados em [Hall 1999] [Papageorgiou et al. 2003] [Roess et al. 2004], em que os autores concordam que ITS envolve a aplicação de modernas tecnologias ligadas a tecnologia da informação. Essas tecnologias envolvem automação de estradas, sistemas automáticos de coleta de pedágio, GPS, sistemas embutidos, sistemas de informação ao usuário e dispositivos inteligentes de controle.

O êxito no desenvolvimento e na aplicação destas tecnologias pode ser a chave para resolver muitos dos problemas de transporte dos nossos dias. A potencialidade dos ITS para encontrar soluções para os transportes europeus do século XXI é apontada no documento "A política européia de transportes no horizonte 2010: a hora das opções" publicado pela Comissão Europeia em 2003 [EUT 2001]. Numa abordagem visando o aperfeiçoamento do sistema europeu de transportes, devem ser encaradas três questões principais: segurança, poluição e congestionamentos.

Os ITS podem alertar antecipadamente para a ocorrência de acidentes rodoviários e outros incidentes, através de painéis de mensagens variáveis, permitindo aos condutores reduzir a velocidade antes de se depararem com o tráfego parado, diminuindo a quantidade de acidentes secundários [Ghosh e Lee 2000].

Os principais objetivos dos ITS, que também podem ser vistos como benefícios, são resumidos a três grandes questões: aumento da segurança, preservação do meio ambiente e aumento da eficácia e eficiência dos transportes (economia de tempo e dinheiro) [Figueiredo 2005].

No decorrer da evolução dos programas ITS, em que estiveram envolvidas organizações de vários países, principalmente da Europa, os EUA e o Japão, houve a necessidade de estabelecer uma gama de aplicações no âmbito dos ITS, também denominadas de serviços ao usuário. Assim, é conveniente pensar em ITS de forma subdividida, como nas seis categorias propostas por [Sussman 2005]:

- Sistemas Avançados de Gerenciamento de Tráfego *(ATMS - Advanced Traffic Management Systems)*: compreendem as funções administrativas do tráfego de uma forma global. Por exemplo, eles podem ser responsáveis pela detecção de congestionamentos e sugestão de novas rotas aos motoristas em virtude disso. Seu maior objetivo é o tráfego de veículos fluir da forma mais eficiente possível;
- Sistemas Avançados de Informação ao Viajante *(ATIS - Advanced Traveller Information Systems)*: visam fornecer informações dos mais variados tipos aos viajantes,

seja dentro de seu veículo, em casa ou no trabalho, dando-lhe assim poder de decisão sobre as opções disponíveis para sua viagem. Incluem, por exemplo, informações sobre condições meteorológicas, condições de estradas, locais de incidentes e rotas ótimas;

- Sistemas Avançados de Controle de Veículos *(AVCS - Advanced Vehicle Control Systems)*: oferecem ao motorista maior conhecimento sobre o seu veículo em determinadas situações, principalmente aquelas que podem resultar em maior eficiência e segurança no seu uso. Sistemas de alerta a colisões são um bom exemplo de AVCS, além de sistemas que auxiliem manobras como estacionar um veículo;
- Operações de Veículos Comerciais *(CVO - Commercial Vehicle Operations)*: adotados por companhias privadas de táxis, vans ou ônibus visando melhoria na produtividade e eficiência de suas operações;
- Sistemas Avançados de Transporte Público *(APTS - Advanced Public Transportation Systems)*: aplicáveis a processos que visem, por exemplo, disponibilizar informações ao usuário do transporte público e auxiliar na melhoria do escalonamento e itinerário dos veículos que fazem parte da rede pública.
- Sistemas Avançados de Transporte Rural (*ARTS - Advanced Rural Transportation Systems)*: desafios sobre como ITS podem ser exatamente aplicados a este setor ainda requerem alguns esforços de pesquisa.

3.2 Sistemas de controle de tráfego

O trânsito nas cidades é um problema inserido na vida das pessoas. Principalmente nas grandes cidades, a populacão enfrenta problemas com filas e congestionamentos, o que traz desconforto e prejudica o dia-a-dia dessas pessoas. Para tentar amenizar esse problema, pesquisadores estão estudando maneiras de controlar o tráfego urbano, pois, controlando o tráfego, consegue-se redução nos congestionamentos, propiciando mais segurança e qualidade de vida aos motoristas e pedestres [Andrade 2008].

As interseções (junções) constituem a parte crítica do sistema urbano, pois, em razão dos movimentos conflitantes de veículos e pedestres que ali ocorrem, são os pontos onde é menor a capacidade de tráfego e maior a frequência de acidentes. Nas interseções com menor volume de veículos e pedestres, a operação com uma das vias como preferencial, isto é, com prioridade de passagem, constitui uma solução adequada no tocante ao desempenho operacional. No entanto, com o crescimento do tráfego começa a haver dificuldade dos veículos e dos pedestres para entrar ou passar pela via principal, com consequente formação de filas, maior demora e, quase sempre, aumento da freqüência de acidentes. Quando a situação fica crítica, é indicada a implantação de um sinal de trânsito (semáforo) no local [Bezerra 2007]. Sinais de trânsito são um dos meios mais efetivos de controle

de tráfego [Kutz 2004].

Semáforos instalados sem indicação técnica e/ou com projetos inadequados (físico e/ou operacional) podem ocasionar problemas como aumento da demora e da quantidade de paradas, redução da capacidade, aumento da frequência de acidentes, aumento nas violações das regras de trânsito e utilização de rotas alternativas para evitar o semáforo, gerando problemas para as ruas de características locais [Kutz 2004]. Por outro lado, semáforos instalados com indicação técnica e com projeto adequado trazem em geral as seguintes vantagens: redução da demora e da quantidade de paradas, aumento da capacidade, redução do tamanho das filas, redução da freqüência de acidentes, redução da emissão de poluentes e a redução no consumo de combustível [Kutz 2004].

Diversos estudos [Matzoros e Van Vliet 1992] [Martins 1996] [Robertson et al. 1996] mostram que semáforos com programação deficiente aumentam consideravelmente o consumo de combustível, a emissão de poluentes e o ruído. Outros estudos, como os de [King e Goldblatt 1975] [Datta e Dutta 1990], suportam a idéia de que a instalação de semáforos reduz o número de acidentes.

A partir das constatações anteriores, entende-se que os semáforos são importantes para o controle de tráfego. Além disso, um bom projeto de construção e implantação dos sinais de trânsito ajuda na utilização de semáforos com qualidade. No processo de construção dos semáforos, o software é uma parte importante. A criação de modelos adequados que consigam refletir a realidade deste sistema ajuda na melhor compreensão do software proposto. Este trabalho atuará nesse ponto, propondo modelos que reflitam adequadamente a realidade de um sistema de controle de tráfego para tentar diminuir problemas relativos à implementação de software em sistemas de controle de trânsito.

Sinais de trânsito são aplicados para reduzir o congestionamento e atrasos, regular o fluxo de veículos, avisar e melhorar a segurança em uma interseção para veículos e pedestres [Roess et al. 2004]. Entre as principais vantagens dos sinais de trânsito são a sua flexibilidade a esquemas de sinalização, a habilidade de fornecer tratamento de prioridade e coordenação das ruas.

O primeiro sinal foi instalado em Londres em 1868 na interseção das ruas George e Bridge, próximo a Casa do Parlamento. O principal propósito do sinal era fornecer proteção aos pedestres. O primeiro semáforo mecânico apareceu nos Estados Unidos por volta de 1913 em Detroit. Ele mostrava as palavras “STOP” e “GO” em faces alternadas. O primeiro sistema de sinais de trânsito interconectados foi instalado em Salt Lake City em 1917, com seis interseções controladas simultaneamente por um dispositivo manual. A fase de amarelo, adicional às fases verde e vermelho, foi adicionada em Detroit em 1920. O propósito da luz amarela era sinalizar aos motoristas para se retirar da área da interseção [Mueller 1970].

Controladores de tráfego modernos implementam temporização nos sinais e garantem que as indicações dos sinais operem consistentemente e continuamente de acordo com

as fases e tempos programados. O primeiro sistema de controle de tráfego baseado em computador foi instalado em Toronto [Casciato e Cass 1962].

Sistemas de controle de sinais de tráfego avançados tem demonstrado grandes benefícios para o tráfego, como diminuição do tempo de viagem e da quantidade de veículos parados, melhorando a velocidade média e minimizando atrasos. O resultado é um consumo de combustível moderado e baixo impacto ambiental [FHWA 1996].

3.3 Políticas de sinais de trânsito

O *Federal Highway Administration, no Traffic Control System Handbook* [FHWA 1995] [FHWA 1996], classifica as formas de controle de tráfego em duas maneiras diferentes.

- A primeira forma considera o tipo do controlador. O tipo de controlador representa como o semáforo responde às variações do tráfego local. O tipo de controlador pode ser classificado em tempo fixo, atuado ou adaptativo;
- A segunda forma considera o tipo de operação. O tipo de operação representa a forma de operação do semáforo em relação à área onde se encontra instalado. O tipo de operação pode ser classificado em três categorias: interseção isolada, corredores ou rede (área).

Na política de tempo fixo de operação, os valores do tempo de ciclo, a duração e a sequência das fases têm valores fixos, sendo calculados com base nos dados históricos do fluxo de veículos da interseção. Qualquer alteração na programação deve ser modificada mecanicamente no controlador. Esse tipo de aparelhagem deve ser instalada em interseções com volume de tráfego previsível, estável e constante, ou onde o volume de pedestres é grande ou constante. Os semáforos que usam tempo fixo são mais acessíveis economicamente para adquirir, instalar e manter do que os de tempo atuado. A deficiência deste tipo de controle está no fato de não se ajustar à flutuação do tráfego ao longo do dia, ou seja, não responde às variações de tráfego. Entretanto, esta limitação pode ser minimizada, no caso de mudanças pré-estabelecidas na programação semafórica, para determinados períodos do dia, durante os quais o volume do tráfego não altera significativamente [FHWA 1995] [FHWA 1996] [Davol 2001] [Junior 2007]. Porém, com o tempo, a tendência é que os planos de tempo fixo fiquem obsoletos. A troca de planos em cidades com constante crescimento e centenas de semáforos é dispendiosa, lenta e propensa a erros.

A política de tempo atuado responde às variações de tráfego que, através de sensores na interseção, determina qual o plano semafórico será adotado. Estes são instalados onde a flutuação do volume de trafego é irregular ou onde se deseja minimizar as interrupções. Ele é constituído por quatro componentes: os sensores, a unidade de controle, as luzes de tráfego e os cabos de conexão. Os sensores são colocados no pavimento, mas em alguns

casos são posicionados no poste do semáforo. O veículo passa pelo detector que informa ao controlador a presença do veículo [Junior 2007].

Existem três tipos de controladores de tempo atuado [FHWA 1995]. Nos semi-atuados, aplica-se a fase verde para a via principal, exceto quando o detector indica a presença de um veículo na via secundária para entrar na interseção. O detector somente será instalado na via secundária. Nos totalmente atuados, os detectores são instalados em todas as aproximações da interseção, sendo muito utilizado quando o volume de veículos varia ao longo do dia, tornando necessária a modificação do plano semafórico. No tipo volume-densidade, as informações do tráfego atual são registradas e guardadas. Usando as informações registradas, pode-se calcular ou recalcular a duração do tempo mínimo da fase de verde baseado na demanda atual.

Em uma fase atuada, existem três parâmetros de tempo: o tempo mínimo da fase verde, a extensão da fase verde e o tempo máximo da fase verde. O Tempo Mínimo da Fase Verde (*Minimum Green Time*) fornece tempo suficiente para veículos armazenados entre detectores e a linha de parada entrarem na interseção [Kutz 2004]. A Extensão da Fase Verde (*Gap/Passage Time*) é definida como o tempo necessário para um veículo mover do detector da via ao ponto de parada da interseção, na velocidade da via [Kutz 2004]. O Tempo Máximo da Fase Verde (*Maximum Green Time*) é o tempo máximo de verde permitido para uma fase [Kutz 2004].

Os parâmetros anteriores (Tempo Mínimo da Fase Verde, Extensão da Fase Verde e Tempo Máximo da Fase Verde) são uma função do tipo e configuração dos detectores instalados na interseção. O controlador atuado tem como principal característica a possibilidade de estender a fase de verde. A Figura 3.1 mostra como a fase de verde pode ser estendida pela atuação de um veículo. Para isso, os parâmetros definidos anteriormente são utilizados. Independentemente da demanda, a fase de verde é definida para o tempo mínimo. Se um veículo é detectado e a duração máxima do verde não foi alcançada, então o tempo de verde é estendido até a duração da extensão. Isso pode ocorrer repetidamente como mostrado na Figura 3.1. O tempo de verde será terminado quando não ocorrer mais atuações no último tempo estendido ou quando o máximo tempo de verde for atingido [Davol 2001] [Tian 2002].

Outra característica do tempo atuado é a possibilidade de pular uma fase se não existe demanda por esta fase, ou seja, se não existem veículos aguardando por aquele movimento (determinado pelos detectores), o controlador pula a fase e vai para a seguinte.

A política de tempo adaptativo surgiu devido à dificuldade de determinar uma programação semafórica que acompanhasse a variação do tráfego em todos os níveis. Ela é semelhante ao de tempo atuado, mas pode alterar mais parâmetros do que somente a duração do tempo de verde, respondendo a variações de tráfego em tempo real. Os principais ajustes feitos são o tempo de ciclo e a divisão das fases, que determina quanto do tempo de ciclo é composta cada fase. A alteração baseia-se nos dados coletados pelas

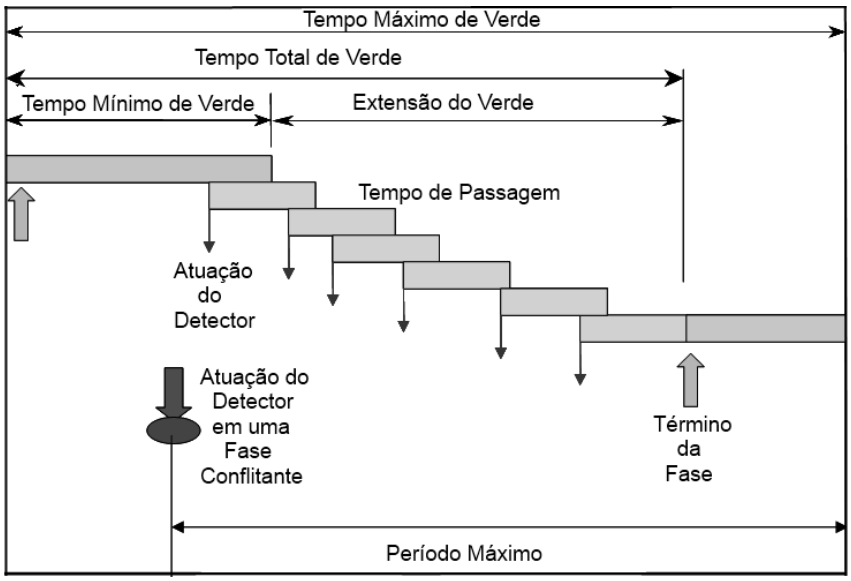

Figura 3.1: Operação de uma fase em modo atuado

aproximações da interseção e estes dados são usados para estimar as condições da interseção e responder em tempo real. A otimização deste controlador está baseada na alocação em tempo real do tempo de verde em função do máximo desempenho ou dos mínimos atrasos e paradas dos veículos [FHWA 1995] [FHWA 1996] [Davol 2001] [Junior 2007].

Como descrito anteriormente, a operação dos semáforos pode ser classificada em isolada, corredores ou rede (área). Os semáforos de uma interseção isolada operam sem considerar as eventuais interferências exercidas pela operação de semáforos das interseções adjacentes. O controlador pode ser de tempo fixo, atuado ou adaptativo, no entanto, não é muito comum o emprego de controladores adaptativos. A escolha do tipo está relacionada com os recursos disponíveis e dentre aquele que melhor atende aos objetivos propostos, como por exemplo, redução de acidentes, minimização de atrasos e segurança [Junior 2007].

Em uma operação do tipo corredor (artéria), o controle do tráfego ao longo de trechos viários tem como objetivo fazer a coordenação ou a sincronização de semáforos, através do acionamento progressivo dos tempos de verde nas interseções sinalizadas, permitindo que os veículos desloquem-se ao longo de uma rota, reduzindo o número de interrupções da corrente de fluxo devido à ocorrência das fases de sinalização vermelha. Este efeito de sincronismo é chamado de *onda verde* [Davol 2001].

Para possibilitar a coordenação de semáforos de tempo fixo, é necessário adotar o mesmo tempo de ciclo para todas as interseções e determinar a defasagem para cada interseção. Os semáforos atuados operam com o mesmo princípio, porém são mais flexíveis, permitindo que fases não coordenadas sejam suprimidas ou estendidas, baseadas na demanda. No entanto, sempre vai existir um tempo de verde, durante cada ciclo, para manter a faixa de onda verde da progressão [Junior 2007].

A operação em rede tem a extensão mais ampla dos controles, pois considera o desempenho de um conjunto de interseções semafóricas de uma determinada área. A operação em rede é uma extensão da operação em corredores que considera a progressão para os veículos em todas as direções de tráfego. Este modo de operação é eficaz quando não existe sentido de trafego dominante na rede. O sistema permite a utilização de controles de tempo fixo, atuado ou adaptativo. Dependendo da complexidade e da quantidade de variáveis envolvidas na otimização, o adaptativo pode demandar um tempo computacional muito elevado para retornar a resposta em tempo real [FHWA 1996].

Capítulo 4

Comparação entre UML, SPT e MARTE

Neste capítulo, serão abordados conceitos imprescindíveis para o melhor entendimento do estudo de caso desta aplicação. Na Seção 4.1, o *profile* MARTE é detalhado. Nesta seção, os seus principais pacotes são apresentados. Na Seção 4.2, o *profile* MARTE é analisado sob um ponto de vista mas crítico, analisando seus pontos fortes e fracos, benefícios e suas dificuldades. Finalmente, na Seção 4.3, a UML, o SPT e o MARTE são comparados segundo aspectos relevantes para ambientes de software de tempo real.

4.1 MARTE

Como descrito anteriormente (subseções 2.2.3, 2.2.4 e 2.2.5), o *profile* MARTE foi criado como tentativa de resolver os problemas da UML e do SPT para modelagem de software de tempo real. Os esforços para o desenvolvimento do MARTE começaram a partir do lançamento da UML 2.0, em 2005. Naquele momento, o *profile* SPT precisava de uma nova versão para atender as inovações da UML 2.0. Consequentemente, um novo *RFP (Request For Proposals)* foi feito pela *OMG (Object Management Group)* procurando por um novo *profile* para modelagem de software de tempo real. Para responder a esse RFP, membros da OMG decidiram desenvolver em conjunto uma submissão. Esses membros da OMG ficaram conhecidos como consórcio *ProMARTE*. O *ProMARTE* consistia nas seguintes empresas e universidades: *Alcatel, ARTISAN Software Tools, Carleton University, CEA LIST, ESEO, ENSIETA, France Telecom, IBM, INRIA, INSA from Lyon, Lockheed Martin, MathWorks, Mentor Graphics Corporation, NoMagic, Software Engineering Institute (Carnegie-Mellon University), Softeam, Telelogic AB, Thales, Tri-Pacific Software e Universidad de Cantabria.* Como resultado do trabalho do *ProMARTE*, foi submetida e aceita uma versão *Beta* do MARTE em 2007. Após atualizações e correções, a versão final (1.0) foi lançada em Novembro de 2009 [OMG 2009]. A versão atual do

MARTE é a versão 1.1 [OMG 2011].

MARTE é um *profile* UML para a modelagem de sistemas embutidos e de tempo real. Ele consiste em um conjunto de extensões de conceitos da UML fornecendo aos desenvolvedores um conjunto de construções para modelar sistemas embutidos de tempo real. As extensões fornecidas pelo MARTE são focalizadas em estereótipos, restrições, valores rotulados e enumerações. Como ilustrado na Figura 4.1, MARTE é organizado em pacotes. Os pacotes serão melhor explicados nas subseções seguintes. O propósito das próximas subseções não é explicar detalhadamente as extensões do MARTE, mas apresentar os conceitos principais de cada pacote. Para maiores detalhes de cada extensão referente a cada pacote, recomenda-se uma leitura da especificação da OMG [OMG 2011].

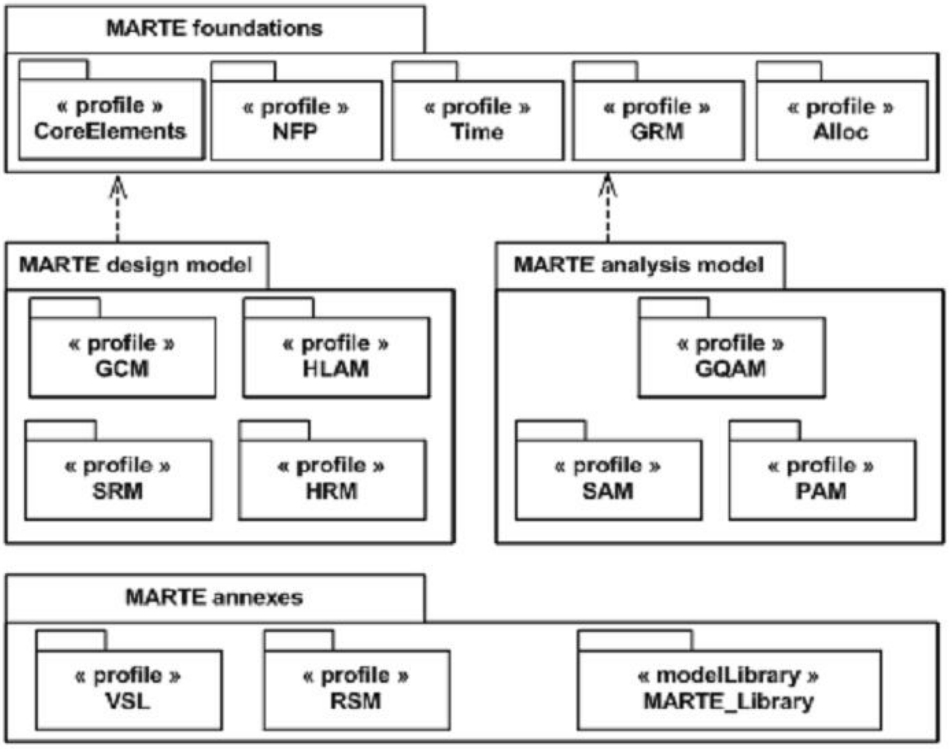

Figura 4.1: O *profile* MARTE [OMG 2011]

4.1.1 Fundamentos do MARTE

O pacote Fundamentos do MARTE (*MARTE Foundations*) define conceitos fundamentais para sistemas embutidos e de tempo real. O *MARTE Foundations* é composto dos subpacotes Elementos Principais *(CoreElements - Core Elements)*, Propriedades não-funcionais *(NFPs - Non-Functional Properties Modeling)*, Modelagem de Tempo (*Time - Time Modeling*), Modelagem de Recursos Genéricos (*GRM - Generic Resource Modeling*) e Modelagem de Alocação (*Alloc - Allocation Modeling*). Esses conceitos fundamentais são refinados (relação de dependência na UML) para propósito de análise no pacote *MARTE Analysis Model* e para propósito de projeto no pacote *MARTE Design Model* [Mraidha

et al. 2008].

O pacote *CoreElements* apresenta conceitos utilizados como base para descrição da maioria dos elementos da especificação do MARTE. Os conceitos apresentados neste pacote são úteis para definição de conceitos mais elaborados apresentados na especificação.

O pacote *NFP* fornece um *framework* genérico para anotar os modelos UML com requisitos não-funcionais. Para representar os valores o pacote *NFP* utiliza o pacote *VSL*.

O pacote *Time* descreve um *framework* para representar conceitos relacionados a tempo e mecanismos apropriados para modelar sistemas embutidos de tempo real. Existem três classes de abstração de tempo usadas no MARTE. A primeira classe é a causal, cujo interesse é na noção de precedência e dependência entre eventos. A segunda classe é a síncrona, cujo interesse é em adicionar conceitos para modelar simultaneidade entre eventos. A terceira é o físico, cujo interesse é adicionar características para modelar tempo físico. Um dos conceitos mais importantes utilizados no pacote *Time* é o conceito de Clock. Qualquer evento pode ser considerado como um *clock*, como por exemplo, o início/término de ações, envio/recebimento de mensagens ou uma transição sendo disparada. MARTE também faz a distinção entre tempo físico e tempo lógico. O tempo físico é o tempo considerado pelas leis da natureza e o tempo lógico é qualquer evento repetitivo, como o ciclo de um processador. A Figura 4.2 mostra a classificação de tempo físico/lógico e discreto/contínuo. O principal destaque da figura refere-se a não existência do *clock* lógico contínuo e a representação do *clock* lógico como um evento discreto.

	Discreto	Contínuo
Lógico	Clock Lógico	Não usado
Físico	Clock Físico Discreto	Clock Físico Contínuo

Figura 4.2: *Clock* físico e *Clock* lógico

O pacote *GRM* tem como objetivo oferecer conceitos necessários para modelar uma plataforma genérica para a execução de aplicações embarcadas de tempo real. O *GRM* inclui características para modelar a plataforma de execução em diferentes níveis de detalhes e modelar recursos de hardware e software.

O pacote *Alloc* contém extensões para a modelagem de alocação de recursos. A alocação de elementos da aplicação funcional em recursos disponíveis (plataforma de execução) é um dos principais interesses dos sistemas embutidos de tempo real.

4.1.2 Modelo de análise do MARTE

O pacote Modelo de Análise do MARTE (*MARTE Analysis Model*) oferece conceitos para análise. A análise de modelos pode detectar problemas nas primeiras fases de desenvolvimento e reduzir riscos e custos. O *MARTE Analysis Model* é composto dos subpacotes Modelagem de Análise Quantitativa Genérica *(GQAM - Generic Quantitative Analysis Modeling)*. O *GQAM* é usado para especificar Modelagem de Análise de Escalonabilidade *(SAM - Schedulability Analysis Modeling)* e Modelagem de Análise de Desempenho *(PAM - Performance Analysis Modeling)*.

O pacote *GQAM* inclui domínios especializados no qual a análise é baseada no comportamento do software, como desempenho e escalonabilidade, energia, memória, confiabilidade, disponibilidade e segurança. Os conceitos do *GQAM* são especializados em representação de escalonabilidade, no *SAM*, e em desempenho, no *PAM*.

O pacote *SAM* é um pacote do MARTE criado especificamente para análise de escalonabilidade. Como é conhecido, em sistemas de tempo real, a influência do escalonamento no tempo e no desempenho é crucial para garantir o tempo de resposta e o processamento de recursos. O pacote *SAM* oferece duas funções importantes. A primeira é calcular a escalonabilidade do sistema ou de uma parte de um software. A segunda é ajudar a análise de sensitividade para determinar como o sistema pode ser melhorado.

O pacote *PAM* tem o objetivo de modelar desempenho. Medidas de desempenho são medidas estatísticas, como *throughput* médio, atraso e probabilidade de um tempo de resposta falhar. Parâmetros de entrada para a análise de desempenho podem ser probabilísticos, como uma chegada aleatória de um processo. As técnicas de análise de desempenho comuns incluem simulações, modelos de fila estendida e modelos de estados discretos como as Redes de Petri Estocásticas.

4.1.3 Modelo de projeto do MARTE

O pacote Modelo de Projeto do MARTE (*MARTE Design Model*) fornece suporte necessário para realizar uma especificação detalhada de um projeto de software de tempo real. O *MARTE Design Model* é composto dos subpacotes Modelo de Componentes Genéricos *(GCM - Generic Component Model)*, Modelagem de Aplicação de Alto-Nível *(HLAM - High-Level Application Modeling)* e Modelagem de Recursos Detalhados *(DRM - Detailed Resource Modeling)*.

O pacote *GCM* apresenta conceitos adicionais identificados como necessários para modelar artefatos no contexto de software de tempo real utilizando uma abordagem baseada em componentes. Uma das principais razões da criação do pacote *GCM* é a representação e comunicação entre portas. O pacote propõe várias extensões para este fim.

O pacote *HLAM* fornece conceitos de modelagem de alto nível para tratar características de modelagem de software de tempo real. Em comparação com aplicações de domínio

comercial, por exemplo, os sistemas de tempo real precisam realizar a modelagem de características quantitativas, como *deadline*, períodos e, características qualitativas relacionadas a comportamento, como comunicação e concorrência. O pacote *HLAM* fornece extensões para tratar esses aspectos.

O pacote *DRM* tem como objetivo fornecer um conjunto de extensões para modelar hardware e software. O *DRM* é dividido em *SRM* e *HRM*. O pacote *SRM* especifica um conjunto de artefatos de modelagem que podem ser usados para descrever uma abordagem de desenvolvimento para sistemas de tempo real. Mais especificamente, o pacote busca descrever recursos e serviços de software em ambientes multi-tarefas. O pacote *HRM* fornece mecanismos para modelar as partes de hardware dos sistemas embutidos. Como o hardware tem várias arquiteturas e uma grande quantidade de componentes existentes e está continuamente sendo modificado por novas tecnologias emergentes, modelar esse domínio requer uma linguagem altamente expressiva.

4.1.4 Anexos do MARTE

O *profile MARTE*, em sua versão 1.1, tem oito anexos, nomeados de Anexos A a H. O Anexo A, denominado *Guidance Example for Use of MARTE*, oferece um guia em alto nível para a utilização do MARTE. O Anexo B, denominado *Value Specification Language (VSL)*, apresenta a linguagem *VSL* utilizada pelo pacote *NFP* para representar valores de restrições, propriedades e aspectos dos estereótipos particularmnte relacionados com aspectos não-funcionais. Essa linguagem de expressão pode ser usada por usuários do *profile* em valores rotulados, corpo de restrições e em qualquer elemento UML associado com a especificação de valores. A *VSL* trata como especificar parâmetros, variáveis, constantes, expressões em formas textuais, lógicas, aritméticas, coleções de valores e intervalos, por exemplo.

O Anexo C, denominado *Clock Handling Facilities*, fornece uma sintaxe abstrata para especificar valores e dependências entre *clocks*. Esse anexo utiliza as linguagens formais *Clocked Value Specification Language (CVSL)* e *Clock Constraint Specification Language (CCSL)* para fins de representação de valores e dependências, respectivamente. Mais detalhes sobre as linguagens formais podem ser encontrados em [André et al. 2007] [Mallet 2008] [André e Mallet 2008] [André e Mallet 2009].

O Anexo D, denominado *Normative MARTE Model Libraries*, define uma biblioteca de tipos primitivos que incluem operações predefinidas geralmente usadas no domínio de sistemas embutidos de tempo real.

O Anexo E, denominado *Repetitive Structure Modeling (RSM)*, propõe a definição de construções de alto nível para auxiliar na modelagem de sistemas embutidos de computação intensiva. Como exemplo desses sistemas, pode-se citar o domínio de aplicações de processamento de sinais e processamento de imagens. Este domínio de aplicações realiza

cálculos de maneira intensiva, possivelmente de forma paralela e distribuída.

O Anexo F, denominado *Domain Class Descriptions*, descreve os conceitos das extensões propostas em outros pacotes de maneira detalhada. O Anexo G, denominado *Bibliography*, fornece a bibliografia que auxiliou a elaboração da especificação. O Anexo H, denominado *Mapping SPT on MARTE*, é voltado para desenvolvedores que querem migrar do SPT para o MARTE. Este anexo fornece uma associação entre as extensões fornecidas pelo SPT e quais são as extensões fornecidas pelo MARTE que tem significado semelhante.

4.2 Fraquezas avaliadas do MARTE

Apesar das melhorias trazidas pelo MARTE, o estudo detalhado e a aplicação das suas extensões revelaram alguns problemas passíveis de melhorias para próximas versões. O documento que propõe o MARTE [OMG 2011], em sua versão atual, é bastante longo (754 páginas) e de leitura complexa. Existem diversos aspectos complexos não bem explicados anteriormente e são apresentados aos leitores de maneira direta. Além de bom conhecimento do metamodelo MOF e de UML, são necessários sólidos conhecimentos de sistemas de tempo real, embutidos e distribuídos para facilitar a leitura.

A especificação do MARTE é redundante em diversas partes. Na aplicação do MARTE, frequentemente, surge a dúvida de qual estereótipo aplicar. Em alguns cenários, é possível aplicar vários estereótipos para representar uma única característica, sendo que estes estereótipos diferem muito sutilmente entre si. Por exemplo, estereótipo *<<RtSpecification>>* (pacote *HLAM*) tem uma semântica muito próxima a semântica existente no estereótipo *<<TimeInstantObservation>>* (pacote *Time*), deixando a sua utilização ambígua. Existe confusão na utilização dos estereótipos do *SRM* com os estereótipos do *GRM*.

A redundância descrita no parágrafo anterior é muito ocasionada pelo excesso de estereótipos presentes na especificação. A maioria das extensões presentes no MARTE são estereótipos e esses, por vezes, podem deixar os diagramas complexos. Se o *profile* utilizasse novos diagramas como extensões provavelmente facilitaria o seu uso e disseminação.

Assim como a UML [Berkenkotter 2003] e o SPT [OMG 2005b], o *profile* MARTE é uma linguagem semi-formal. A falta de formalismo em certos pontos pode não garantir a corretude esperada em modelos de desenvolvimento de software de tempo real e corrobora no aumento da ambiguidade e da redundância nos diagramas anotados com as extensões do *profile*.

Uma alternativa para superar muitos dos problemas citados anteriormente seria a utilização de explicações detalhadas para os exemplos apresentados. Entretanto, este ponto é falho. O *profile* mostra bons exemplos para ilustrar os conceitos teóricos, mas os exemplos não são precedidos de explicações sobre conceitos relevantes para o seu entendimento e não há detalhamento suficiente dos exemplos para um entendimento profundo dos conceitos

sob um ponto de vista prático.

Devido a aspectos como redundância, complexidade e a falta de formalismo, a aplicação do *profile* MARTE é dificultada. Sabe-se que a utilização de guias ou processos na aplicação de técnicas de engenharia de software facilitam a sua implantação. A utilização de um guia ou um método de aplicação poderia facilitar a utilização do MARTE, indicando quais extensões são mais adequadas em determinados cenários.

A utilização de ferramentas de engenharia de software *(Computer-Aided Software Engineering - CASE)* faz parte das atividades dos engenheiros de software. Porém, as ferramentas que suportam a utilização do MARTE ainda são raras. Além disso algumas das ferramentas existentes são demasiadamente complexas e não confiáveis, como exemplo o *Papyrus UML* [Gérard 2011].

Pelas razões descritas nos parágrafos anteriores, a utilização do MARTE de maneira eficiente e profissional exige bastante trabalho e experiência.

4.3 Comparativo entre UML, SPT e MARTE

A comparação entre UML, SPT e MARTE ainda não está clara na presente literatura. As principais razões para isso são:

- A baixa aceitação do SPT pela comunidade. Como descrito anteriormente (Seção 2.2.4), apesar do SPT ter sido padronizado pela OMG, ele não foi bem aceito pela comunidade de software de tempo real. Isso acarretou uma diminuição na sua utilização e comparações com abordagens concorrentes e/ou complementares;
- A recente padronização do MARTE. O MARTE foi padronizado no final de 2009 e ainda não existem muitas aplicações do MARTE no cenário de software de tempo real. A sua pouca utilização dificulta comparações com abordagens concorrentes e/ou complementares.

Com objetivos de permitir ampla visão do potencial do MARTE, a Tabela 4.1 apresenta uma comparação entre essas especificações da OMG. Essa tabela é preenchida baseando-se em aspectos importantes no projeto de software de tempo real. A primeira coluna especifica algumas características que precisam ser modeladas em um projeto de software de tempo real ou características genéricas presentes na linguagem, como a modelagem de tempo e o alinhamento com a UML 2.X. As colunas são anotadas com os símbolos ○, ◐ e ●. Células anotadas com o símbolo ○ significam que a característica não está presente na linguagem de modelagem ou a característica não é satisfatória. Células anotadas com ● significam que a característica está altamente presente na linguagem de modelagem, isto é, sua aplicação é satisfatória. Células anotadas com ◐ significam que a característica ainda não é bem conhecida ou não foi completamente avaliada ou ainda a linguagem não oferece suporte para esta característica.

O preenchimento da tabela foi realizado segundo alguns critérios. As colunas referentes a UML e ao SPT foram preenchidas considerando a análise de artigos presentes na literatura. A coluna referente ao MARTE foi preenchida considerando a análise da literatura e a utilização do MARTE juntamente com a UML realizada pelo autor deste trabalho. A opinião do autor foi emitida apenas na coluna referente ao MARTE. Células onde não existem referências para afirmações consideram as próprias especificações da OMG como fundamentação.

Os principais destaques da tabela neste trabalho são as referências para problemas da UML e do SPT e pontos fortes, pontos fracos e desafios do MARTE. De forma geral, a UML é fraca para modelagem de software de tempo real porque a linguagem não é eficiente em representar as características do ambiente de tempo real, como por exemplo, modelagem de tempo, a modelagem de processos e a modelagem de recursos. Como descrito anteriormente (Subseção 2.2.3), a UML é adequada para modelagem de software de propósito geral e, apesar de ser bastante utilizada na modelagem de software de tempo real, a linguagem não é adequada para este fim.

O *profile* SPT, apesar de ter sido padronizado para a modelagem de software de tempo real, apresenta vários problemas para a modelagem de software de tempo real. A modelagem de tempo, processos e recursos no SPT ainda é passível de melhorias. O SPT não é bem adequado a CBSE e não atende completamente a UML 2.X (na primeira versão do documento). Em linhas gerais, o SPT tem características melhores do que a UML para modelar software de tempo real, mas os muitos problemas encontrados (Subseção 2.2.4) tornaram o *profile* pouco utilizado na academia e na indústria de software.

A tabela mostra os pontos fortes do MARTE para modelagem de software de tempo real. Essas características foram descritas em trabalhos anteriores, como indicado na tabela, e pela aplicação do *profile* MARTE no estudo de caso mostrado no Capítulo 6. Por exemplo, o projeto baseado em componentes (*Component Based Software Engineering - CBSE*) foi melhorado com a utilização do MARTE, assim como a modelagem de tempo, recursos, processos e requisitos não-funcionais (desempenho e escalonabilidade).

Característica	**UML [OMG 2010b]**	**SPT [OMG 2005b]**	**MARTE [OMG 2011]**
Modelagem de tempo	◐ [Graf et al. 2006] [Demathieu et al. 2008]	◐ [OMG 2007]	●
Modelagem de processos	○ [Henderson-Sellers 2005]	◐	● [Demathieu et al. 2008]
Modelagem de recursos	◐	◐ [OMG 2007]	● [Demathieu et al. 2008]
Modelagem de alocação de recursos	◐ [Henderson-Sellers 2005]	◐ [Demathieu et al. 2008] [OMG 2007]	●
Modelagem de hardware	○ [Süß et al. 2008] [OMG 2010c]	○ [Demathieu et al. 2008]	◐ [Boutekkouk et al. 2009]
Modelagem de desempenho	○ [Henderson-Sellers 2005]	◐ [Petriu e Woodside 2004]	● [Boutekkouk et al. 2009]
Modelagem de escalonabilidade	○ [Douglass 2004] [Henderson–Sellers 2005]	◐	●
Modelagem de tempo de execução	○ [Demathieu et al. 2008]	◐ [OMG 2007] [Demathieu et al. 2008]	● [Demathieu et al. 2008]
Modelagem de gerenciamento de tarefas	○ [Berkenkotter 2003] [Henderson-Sellers 2005]	●	● [Demathieu et al. 2008]
Modelagem de requisitos	○ [Henderson-Sellers 2005]	○	○ [Boutekkouk et al. 2009]
CBSE	○	○ [Petriu e Woodside 2004] [OMG 2007]	● [Boutekkouk et al. 2009]
Alinhamento com UML 2.X	não se aplica	◐ [Demathieu et al. 2008] [OMG 2007]	●
Formalismo	○ [Berkenkotter 2003] [Henderson-Sellers 2005]	○	○ [Boutekkouk et al. 2009]
Consistência dos diagramas	○ [Jacobson 2004] [Staines 2005] [Henderson-Sellers 2005]	○	○ [Jin et al. 2011]
Facilidade de uso	◐ [Henderson-Sellers 2005]	○ [OMG 2007]	○
Ferramentas disponíveis	●	○	○

Tabela 4.1: Comparação entre UML, SPT e MARTE

Alguns desafios do MARTE são os mesmos desafios encontrados na UML e no SPT, como aumentar o nível de formalismo e a criação de diagramas mais consistentes. Outro exemplo, é a modelagem de requisitos em um alto nível de abstração. As linguagens estudadas são adaptadas para modelar cenários de requisitos, frequentemente em um baixo nível de abstração. Outro *profile* UML, a SysML [OMG 2010c], pode preencher essa lacuna [Soares et al. 2011]. Adicionalmente, existem muitas extensões no MARTE, como estereótipos, restrições e valores rotulados. Dessa forma, é difícil tornar-se um analista proficiente. Finalmente, há carência de ferramentas que possam implementar completamente as capacidades do MARTE.

Capítulo 5

Arquitetura em múltiplas visões usando MARTE

Este capítulo foi baseado no artigo *"Multiple View Architecture Model for Distributed Real-Time Systems Using MARTE"* apresentado na *20th International Conference on Information Systems Development (ISD 2011)* [Silvestre e Soares 2011]. O objetivo deste capítulo é propor um modelo de arquitetura de software em múltiplas visões para ser aplicado no projeto de sistemas distribuídos de tempo real, no qual MARTE e UML são utilizados como linguagens de modelagem. A arquitetura deve ser vista como uma arquitetura complementar a outras arquiteturas. O foco é descrever as visões de arquitetura mais importantes considerando projetos de sistemas distribuídos de tempo real. A Seção 5.1 faz uma introdução à arquitetura de software, destacando os seus conceitos principais e a sua importância. A Seção 5.2 apresenta os conceitos de visões arquiteturais. A Seção 5.3 apresenta os detalhes da arquitetura em múltiplas visões proposta.

5.1 Introdução à arquitetura de software

Com o aumento do tamanho dos sistemas de software, os algoritmos e a estrutura de dados de computação não constituem mais o maior problema do *design* [Garlan e Shaw 1994]. A arquitetura de software é uma das partes mais problemáticas do *design* e é considerada como um dos fatores técnicos mais importantes para garantir um projeto de sucesso [Brown e McDermid 2007].

Arquitetura de software é a organização fundamental de um sistema, incluindo seus componentes, o relacionamento entre esses componentes e com o ambiente e os princípios que definem seu *design* e a evolução dos componentes [Hilliard 2000]. Em termos simples, a descrição da arquitetura de um sistema envolve elementos que compõem o sistema e como eles interagem, incluindo os pontos de interação.

A arquitetura de software é parte do *design*, mas com foco nos elementos princi-

pais, como as classes principais e suas comunicações, padrões, *frameworks*, camadas, sub-sistemas, componentes e interfaces [Kruchten 2003]. A arquitetura também está preocupada no modo como os elementos são distribuídos, o modo como eles se comunicam e como o ambiente do sistema é descrito. Um equívoco comum é que a arquitetura de software refere-se apenas a estrutura dos elementos e seus relacionamentos [Kruchten 2003]. Arquitetura de software refere-se também ao comportamento do software e a comunicação entre elementos das interfaces.

A arquitetura de software é o produto do desenvolvimento que fornece maior retorno sobre o investimento com respeito a qualidade, o cronograma e o custo. O desenvolvimento de uma arquitetura correta facilita outras etapas do ciclo de vida, como implementação e manutenção. O desenvolvimento de uma arquitetura errada é um meio para a construção de um software errado [Bass et al. 2003].

Decisões realizadas na definição da arquitetura de software tem impactos a longo prazo no *design* e nos principais atributos de qualidade do software, como o desempenho, evolução e segurança. Essas decisões refletirão nos modelos do *design* e na implementação do software. Além disso, a descrição da arquitetura é um meio efetivo de comunicação entre diferentes *stakeholders* [Soares 2010].

Um conceito bem aceito na área de desenvolvimento de software é que ter uma descrição de arquitetura bem definida é um dos fatores essenciais para um software de sucesso [Kruchten et al. 2006]. O conhecimento da estrutura do sistema e o relacionamento entre seus componentes ajuda no desenvolvimento de software e facilita a comunicação entre *stakeholders*. Sem a utilização de uma arquitetura apropriada para o problema sendo desenvolvido, o projeto irá falhar [Garlan et al. 2010].

Grandes sistemas de software raramente são construídos de maneira improvisada [Boehm e Turner 2003]. A arquitetura de software é útil para criar condições necessárias para melhorar a reusabilidade de software. O reúso de partes existentes tem grande potencial de entregar sistemas confiáveis.

A manutenção e a evolução de sistemas é facilitada quando a arquitetura do software está bem definida e divulgada para os *stakeholders* [Lindgren et al. 2008]. Grandes sistemas de software devem ser flexíveis para permitir a sua evolução, aumentar a sua longevidade e facilitar a sua manutenibilidade. Resultados empíricos e experiências práticas mostram que sistemas com uma arquitetura de software bem definida são resilientes a mudanças [Booch et al. 2007]

A pesquisa na área de engenharia de software vem crescendo rapidamente nos últimos anos e muitos pesquisadores e desenvolvedores reconhecem a importância em ter uma descrição da arquitetura de software bem definida para suportar as atividades como gerenciamento de projetos, *design* e implementação. Por outro lado, uma definição de arquitetura pobre é conhecida como um dos principais riscos técnicos no projeto de software de tempo real [Garlan et al. 2010]. Resumidamente, a arquitetura de software é um

elemento crítico no ciclo de vida de um software.

5.2 Visões em arquitetura de software

Como consequência da existência dos diversos interesses nos objetivos alcançados pelo software, a arquitetura também possuirá diversos interessados. No entanto, uma vez que os interessados no sistema têm diferentes preocupações e níveis de conhecimento, a arquitetura não deve ser exposta da mesma maneira para interessados diferentes. Para resolver esse problema, surge o conceito de visões arquiteturais [Barbosa 2009].

Visões são diferentes formas de observar um mesmo problema com a finalidade de melhor entendê-lo para atribuir-lhe a solução mais adequada [Varoto 2002]. Essas formas estão relacionadas ao modo como:

- As pessoas que desempenham papéis diferentes dentro do processo de desenvolvimento de software veem o problema;
- Cada entidade (componente) da arquitetura de software pode ser observada (perspectivas diferentes).

Uma visão arquitetural é uma representação da informação (ou parte dela) contida na arquitetura de forma que atenda às necessidades de um ou mais interessados. Ela facilita o entendimento da arquitetura por parte do interessado, uma vez que vai filtrar e formatar a informação de acordo com as necessidades e preocupações do interessado em questão [Barbosa 2009].

Existem informações independentes umas das outras e não é possível representá-las em um mesmo diagrama. As visões arquiteturais são a forma de detalhar e representar separadamente cada uma destas informações dissociadas [Varoto 2002].

A arquitetura de software em múltiplas visões também é útil ao arquiteto. Quando um arquiteto faz o *design*, ele usa o conceito de visões arquiteturais para endereçar as diferentes preocupações do sistema. Dessa maneira, ele divide o problema do *design* em problemas menores e, consequentemente, menos complexos: ele endereça cada atributo de qualidade que será alcançado por essa arquitetura. Atacando uma visão por vez, o arquiteto pode [Barbosa 2009]:

- Definir as partições lógicas, ou seja, os módulos funcionais que comporão o sistema e considerar uma visão lógica do sistema;
- Definir as partições dinâmicas do sistema, ou seja, quais processos, *threads* e protocolos estarão presentes no sistema;
- Definir as partições do ponto de vista de implementação, ou seja, quais classes, pacotes e bibliotecas comporão o sistema;

- Definir onde as partes dinâmicas executarão, ou seja, onde e em quais máquinas os diversos "executáveis" do software estarão implantados, além de como eles vão se comunicar.

Alguns dos principais esquemas de visões de arquitetura são as visões da OMT [Rumbaugh et al. 1991], as visões do RM-ODP [Bietz et al. 1993], as visões de Booch [Booch 1994], o modelo em múltiplas visões "4+1" [Kruchten 1995] (Figura 5.1), as visões de Zachman [Zachmann 1997], as visões da arquitetura de referência ANSI/IEEE 1471-2000 [Hilliard 2000] (Figura 5.2) e as visões de Leist e Zellner [Leist e Zellner 2006]. Nos próximos parágrafos serão detalhados a arquitetura em múltiplas visões "4+1" e ANSI/IEEE 1471-2000.

A arquitetura em múltiplas visões "4+1" é composta das visões lógica, de implementação, de processos e de implantação. A visão de casos de uso é usada para ilustrar a arquitetura e representa a visão +1.

A visão lógica está concentrada na funcionalidade que o sistema disponibiliza para o usuário final. Os diagramas UML usados para representar a visão lógica são o diagrama de classes, diagrama de comunicação e diagrama de sequência. A visão de implementação ilustra o sistema do ponto de vista do programador. Os diagramas UML usados para representar a visão de implementação são o diagrama de componentes e o diagrama de pacotes. A visão de processos permite visualizar as partes dinâmicas do sistema, explicar os processos e como eles se comunicam, focando no comportamento do sistema. Um diagrama UML usado para representar a visão de processos é o diagrama de atividades. A visão de implantação mostra a arquitetura física do sistema. O diagrama UML normalmente usado para descrever esta visão é o diagrama de implantação. A visão de casos de uso descreve a arquitetura do sistema segundo os seus cenários de utilização. O diagrama UML usado para descrever a visão de casos de uso é o diagrama de caso de uso.

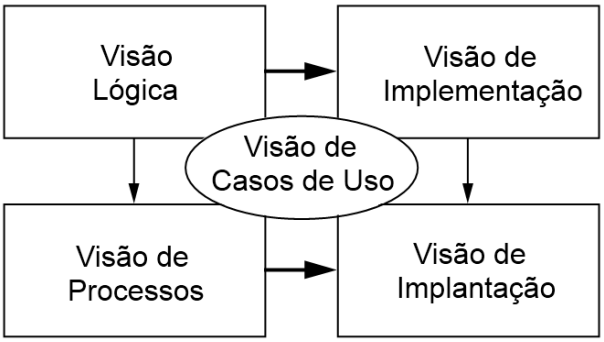

Figura 5.1: O modelo de visão "4+1"

As visões da arquitetura de referência ANSI/IEEE 1471-2000 descrevem práticas recomendadas para a representação de uma arquitetura de software em múltiplas visões. A arquitetura proposta endereça atividades de criação, análise e registro das arquiteturas em termos das descrições arquiteturais. Um *framework* conceitual para descrição de

arquitetura é estabelecido.

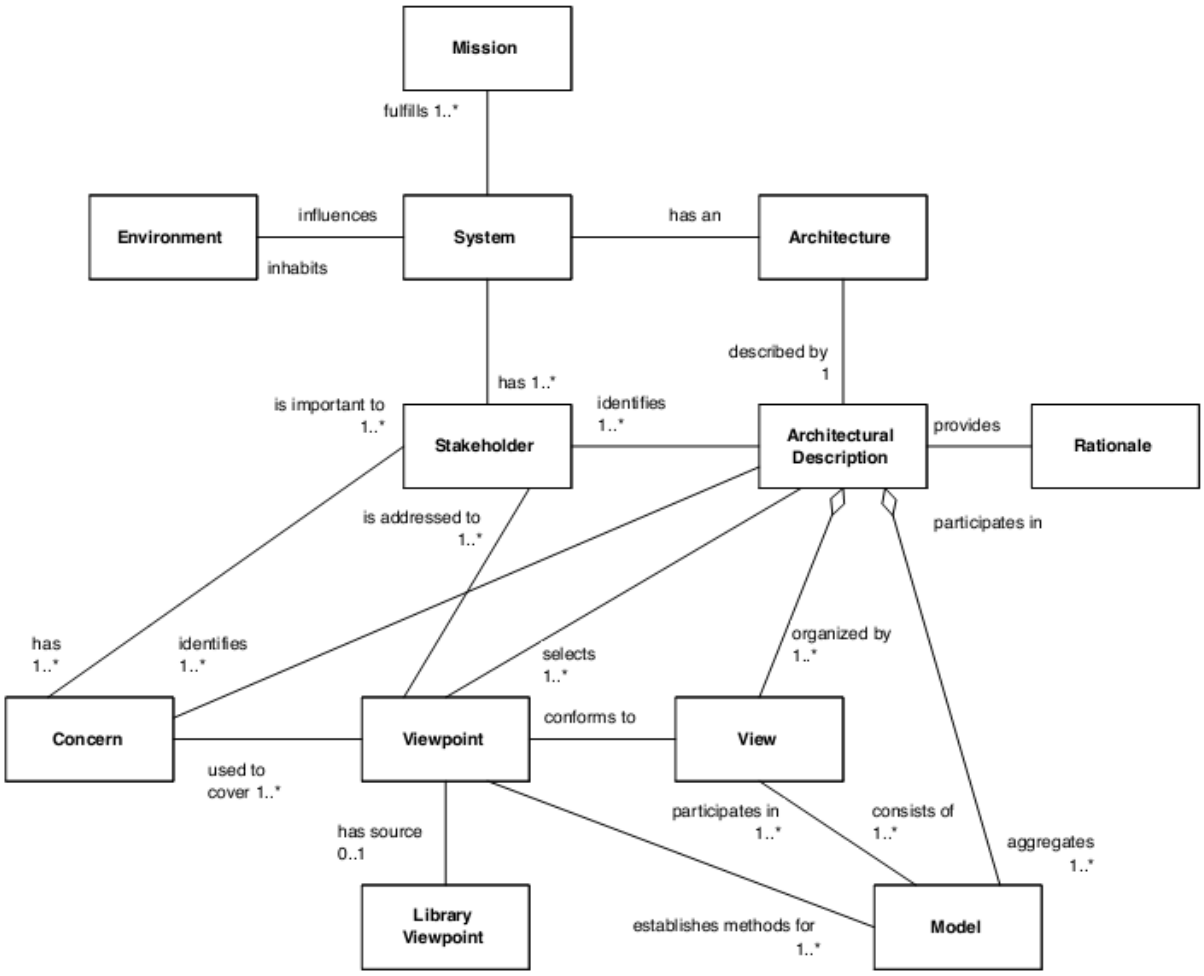

Figura 5.2: O modelo de visão IEEE-Std-1471-2000

5.3 Arquitetura em múltiplas visões usando MARTE

A arquitetura proposta (Figura 5.3) é baseada em múltiplas visões, inspiradas em trabalhos anteriores [Soni et al. 1995] [Kruchten 1995]. A arquitetura de referência ANSI/IEEE 1471-2000 também especifica uma arquitetura organizada em múltiplas visões. As visões escolhidas nessa pesquisa são a visão de Processos, visão de Tempo, visão de Recursos Compartilhados e visão de Alocação de Recursos. Essas visões são relacionadas a sistemas distribuídos de tempo real e não foram bem descritas em trabalhos anteriores. As visões propostas devem ser usadas juntamente com outros modelos de arquitetura em múltiplas visões (as visões não são mutualmente exclusivas).

A visão de Processos é usada porque ela captura questões como concorrência, sincronização, aspectos dos requisitos não-funcionais do projeto, como desempenho e disponibilidade e trata questões de integridade e tolerância a falhas [Kruchten 1995]. Os requisitos não-funcionais apresentados são importantes para ser representados em sistemas distribuí-

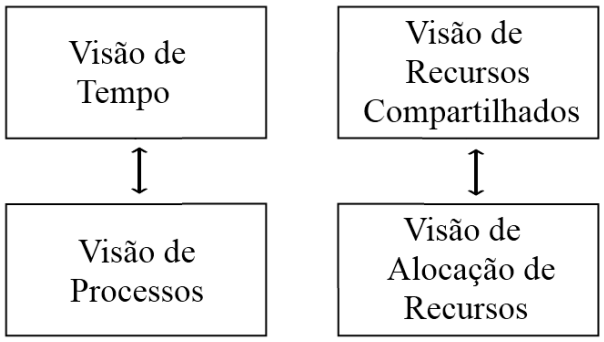

Figura 5.3: Arquitetura em multiplas visões proposta

dos de tempo real. A visão de Tempo é aplicada para representar uma das características mais importantes em um software de tempo real: o tempo. A visão de Recursos Compartilhados representa a descrição dos recursos compartilhados. A definição dos recursos é importante para tratar aspectos como concorrência, elementos distribuídos e paralelismo. A visão de Alocação de Recursos representa a alocação de elementos funcionais da aplicação em recursos disponíveis. Em muitos casos os recursos não estão disponíveis, assim um mecanismo específico precisa gerenciar a alocação de recursos.

As próximas subseções descrevem a arquitetura em múltiplas visões usando o MARTE. A Subseção 5.3.1 detalha a visão de processos referente a arquitetura proposta. A Subseção 5.3.2 detalha a visão de tempo da arquitetura proposta. A Subseção 5.3.3 descreve a visão de representação de recursos para a arquitetura proposta. A Subseção 5.3.4 detalha a visão de alocação de recursos para a arquitetura proposta. A Subseção 5.3.5 faz uma conclusão da arquitetura proposta e também a relaciona brevemente com outras arquiteturas.

5.3.1 Visão de processos

A visão de Processos deve tratar eventos, concorrência e aspectos de sincronização do projeto. O pacote *CoreElements* define três estereótipos principais no contexto da visão de processos. O estereótipo *Configuration* representa um conjunto de elementos ativos do sistema, o estereótipo *Mode* indentifica um segmento operacional dentro do sistema, e, o estereótipo *ModeBehavior* especifica um conjunto de modos mutualmente exclusivos.

O pacote *NFP* define o estereótipo *NfpConstraint* que suporta expressões textuais para especificar asserções considerando, por exemplo, desempenho e escalonamento. Ele inclui relacionamentos com outras características por meio de variáveis matemáticas, lógicas e expressões temporais. Além disso, o pacote *VSL* fornece estereótipos para representar quantidades *(TupleType, ChoiceType, CollectionType)* e intervalos de valores *(IntervalType)* fornecidos pelo pacote *NFP*.

O pacote *GCM* fornece conceitos para modelar troca de mensagens entre portas e mensagens estendidas. Alguns dos estereótipos mais importantes são o *DataPool*, usado para

especificar a política de armazenamento; o estereótipo *FlowPort*, usado para especificar a natureza do fluxo; o estereótipo *ClientServerPort*, usado para especificar características fornecidas e requeridas e o estereótipo *DataEvents* usado para especificar mensagens recebidas.

O pacote *Alloc* define extensões para representar a alocação de elementos nos recursos disponíveis da plataforma de execução. A construção mais importante do pacote *Alloc* são os estereótipos. *Assign* é um estereótipo para associar elementos de um contexto lógico (elementos do modelo da aplicação), para elementos descritos em um contexto mais físico (elementos do modelo da plataforma de execução). *NfpRefine* é um estereótipo para associar um elemento do modelo abstrato para elementos do modelo refinado.

O pacote *HLAM* fornece estereótipos para tratamento de concorrência. Para tratar essa característica, o *profile* MARTE usa o estereótipo *RtUnit*. MARTE define estereótipos para representar tipos de concorrência, seja síncrona ou assíncrona *(CallConcurrencyKind, ConcurrencyKind e SynchronizationKind)*. Como exemplo de concorrências comuns que podem ser representadas, estão a concorrência de processos, a concorrência de memória e a concorrência de dispositivos de entrada e saída.

Entre os estereótipos mais importantes do pacote GQAM, estão os estereótipos relacionados a comunicação *(GaCommChannel e GaCommHost)* e representação de recursos *(GaResourcesPlatform)*. Os principais estereótipos definidos pelo pacote *SAM* são usados para analisar recursos compartilhados *(SaSharedResource)* e para medir escalonabilidade, interrupções e processamento *(SaExecHost)*. Os principais estereótipos definidos pelo pacote *PAM* são *PaRunTInstance, PaStep e PaCommStep*.

O pacote *SRM* define vários estereótipos para modelagem de software de tempo real. Entre eles, estereótipos para modelar interrupções, sincronização de recursos, políticas para acessar um recurso e modelar concorrência de recursos.

Os conceitos de tempo, recursos e alocação de recursos relacionados com a visão de processos são descritos nas próximas seções.

5.3.2 Visão de tempo

Mesmo com a UML 2.0 e o novo Diagrama de Tempo, a modelagem de tempo não está claramente definida na UML [OMG 2010b]. O modelo de tempo na UML é muito simplista [Graf et al. 2006]. Como o tempo é um aspecto crítico em determinados ambientes, a especificação da UML (Capítulo 13, Superestrutura) aconselha o uso de um modelo de tempo mais sofisticado para sistemas distribuídos [OMG 2010b]. Por esse motivo, o *profile* MARTE apresenta tempo de uma maneira mais precisa e clara. Tanto modelos de tempos discretos quanto modelos de tempo contínuos são tratados no MARTE. Os *Clocks* são usados para acessar a estrutura do tempo. Os *Clocks* podem ser cronométricos (referem-se a tempo físico) ou podem ser lógicos.

Um *clock* é um conjunto de instantes finito ou infinito, podendo representar um evento temporal e momentos da sua ocorrência. Um *clock* tem uma unidade e os instantes podem ser rotulados. Esses instantes em um *clock* são totalmente ordenados por *clocks* discretos no tempo, assim, eles pode ser indexados por números naturais. Uma estrutura de tempo é composta de um conjunto de *clocks* com relação de precedência entre eles. Precedência é uma relação binária entre os *clocks*. Dessa relação, as seguintes novas relações podem ser derivadas: coincidência, precedência estrita, independência e exclusão.

MARTE descreve extensões necessárias para suportar os conceitos de tempo. Alguns conceitos resultam em novos estereótipos, outros especializam estereótipos definidos para o pacote *NFPs*, outros ainda não precisam de extensão. Os estereótipos *ClockType* e *TimedDomain* representam conceitos relacionados a estrutura de tempo apresentadas na visão do domínio de Tempo. Os estereótipos *TimedValueSpecification* fazem referências a *Clocks*. O pacote de modelagem de Tempo apresenta dois novos estereótipos especializando o estereótipo *NfpConstraint*: *TimedConstraint* e *ClockConstraint*. O estereótipo *TimedEvent* representa eventos cuja ocorrência é explicitamente relacionada a um *clock*. O estereótipo *TimedProcessing* representa atividades que tem início e fim conhecido, e cujas durações são explicitamente relacionada a *clocks*.

5.3.3 Visão de recursos compartilhados

Os principais aspectos do MARTE para representação de recursos estão no pacote *GRM*. O conceito central do *GRM* é a noção de um Recurso. Um Recurso representa uma entidade persistente fisicamente ou logicamente que oferece um ou mais serviços. Recursos e seus serviços são o meio disponível para realizar os serviços esperados e/ou satisfazer os requisitos pelos quais o sistema em consideração foi projetado.

O pacote *GRM* define como os elementos do modelo do domínio estendem metaclasses do metamodelo UML para representar recursos. A definição mais importante no pacote *GRM* é o estereótipo *Resource*. Ele fornece a representação para um recurso genérico de uma maneira holística. O estereótipo *Resource* auxilia na representação de mecanismos para conectar e entregar dados, meios para transportar informações de um lugar para outro, concorrência de recursos, liberação de recursos, uso de recursos, políticas de escalonamento e diferentes formas de memória (memória principal, disco rígido, entre outros).

5.3.4 Visão de alocação de recursos

Alocação compreende tanto a distribuição espacial quanto aspectos de escalonamento temporal para mapear algoritmos em recursos e computações disponíveis. MARTE usa a palavra *allocation* no lugar da palavra *deployment* (como usado na UML). *Deployment* implica em uma distribuiçãoo física, enquanto *allocation* pode ser lógico ou físico.

Um exemplo de alocação é o escalonamento de processos em um processador [Mallet e de Simone 2008]. O mecanismo de alocação proposto pelo MARTE é muito próximo a estrutura de alocação da SysML (*Systems Modeling Language* [OMG 2010c]) porque ele aloca partes lógicas para partes mais físicas [André et al. 2007]. Entretanto, MARTE deixa explícito que tanto a parte lógica quanto a parte física podem ser de natureza estrutural ou comportamental.

O pacote *Alloc* define estereótipos para a modelagem de alocação de recursos. Dois dos estereótipos mais importantes são: *Allocate* e *Allocated*. *Allocate* é usado para identificar um recurso que pode ser alocado. *Allocated* é aplicado para qualquer elemento que tem um relacionamento de alocação com outro elemento. A utilização dos estereótipos em mais detalhes é apresentada no próximo capítulo.

5.3.5 Conclusões sobre a arquitetura em múltiplas visões

Este capítulo apresentou conceitos e questões essenciais sobre arquiteturas de software, destacando aspectos relacionados à sua representação e importância. Além disso, foi proposta uma arquitetura em múltiplas visões usando MARTE para sistemas distribuídos de tempo real.

Uma arquitetura de software inclui o conjunto de decisões significativas sobre a organização de um software, seleção dos elementos estruturais e suas interfaces, comportamento entre esses elementos, composição destes elementos estruturais e de comportamento em subsistemas maiores e estilo arquitetural que guia a organização.

O *profile* MARTE foi proposto para resolver algumas questões da UML relacionadas ao projeto de software de tempo real. MARTE oferece noções interessantes para a UML para modelar aspectos como alocação de recursos e requisitos não-funcionais. Uma atenção especial é dada para restrições temporais, em que três tipos de representações são propostas: causal (interessada na precedência dos *clocks*), *clocked* (adiciona a noção de simultaneidade) e física (modelagem da duração).

Nas seções anteriores, uma arquitetura em múltiplas visões usando UML e MARTE foi proposta. As quatro visões foram identificadas e, para cada uma, MARTE pode ser usada como linguagem de modelagem junto com a UML. Um resumo dos estereótipos do MARTE utilizados para a modelagem das visões arquiteturais neste trabalho é apresentado na Tabela 5.1.

Os estereótipos do MARTE deixam a noção de tempo mais clara que a forma apresentada na UML. Além da visão de tempo, a arquitetura proposta inclui a visão de processos, a visão de representação de recursos e a visão de alocação de recursos. A arquitetura proposta é complementar a outras arquiteturas, não as substitui. A razão é que o foco da arquitetura é descrever as visões mais importantes considerando o projeto de sistemas distribuídos de tempo real. Desta forma, a arquitetura deve ser combinada com outros

modelos de arquitetura em múltiplas visões para descrever outras visões não mencionadas anteriormente, como por exemplo a visão de casos de uso.

Pacote	**Tempo**	**Processos**	**Recursos Compartilhados**	**Alocação de Recursos**
Core Elements		*Configuration*, *Mode* e *ModeBehavior*		
NFP		*NfpConstraint*		
Time	*ClockType*, *Clock*, *TimedValueSpecification*, *TimedDomain*, *NfpConstraint*, *TimedConstraint*, *ClockConstraint*, *TimedEvent* e *TimedProcessing*	*ClockType*, *Clock*, *TimedValueSpecification*, *TimedDomain*, *NfpConstraint*, *TimedConstraint*, *ClockConstraint*, *TimedEvent* e *TimedProcessing*		
GRM		*Resource*	*Resource*	
Alloc		*Assign*, *NfpRefine*, *Allocate* e *Allocated*		*Allocate* e *Allocated*
GCM		*DataPool*, *FlowPort*, *ClientServerPort* e *DataEvents*		
HLAM		*RtUnit*, *CallConcurrencyKind*, *ConcurrencyKind* e *SynchronizationKind*		
SRM				
HRM				
GQAM		*GaCommChannel*, *GaCommHost* e *GaResourcesPlatform*		
SAM		*SaSharedResource* e *SaExecHost*		
PAM		*PaRunTInstance*, *PaStep* e *PaCommStep*		

Tabela 5.1: Representação dos estereótipos usados nas visões propostas

Capítulo 6

Modelagem de sistemas de controle de tráfego usando MARTE

Este capítulo tem como principal objetivo detalhar a aplicação do *profile* MARTE juntamente com a UML na modelagem de um software de controle de sinais de trânsito. A modelagem realizada neste capítulo utiliza as visões arquiteturais para software de tempo real propostas no capítulo anterior. Para o melhor entendimento do estudo de caso, este capítulo está dividido em três seções. A Seção 6.1 é responsável por especificar as principais características referentes ao projeto, com o objetivo de ajudar o leitor a entender melhor o problema. As Seções 6.2 e 6.3 descrevem a utilização de diagramas UML juntamente com as extensões do MARTE para modelar o problema do estudo de caso. A Seção 6.4 realiza uma conclusão sobre a modelagem realizada. Este capítulo é baseado no artigo *"Modeling Road Traffic Signals Control using UML and the MARTE Profile"* aceito na *12th International Conference on Computational Science and Its Applications (ICCSA 2012)* [Silvestre e Soares 2012].

6.1 Requisitos do projeto

A implantação de uma interseção semafórica é uma atividade complexa em que várias decisões de projeto precisam ser tomadas de acordo com os objetivos principais da interseção. As decisões neste ponto impactarão o desempenho e fluidez de veículos na interseção. Para este trabalho, é necessário decidir se serão usados semáforos de duas ou quatro fases, qual a quantidade de cruzamentos que será utilizada no projeto, se existirá tempo prioritário para alguma das vias, a existência ou não de pontos onde pedestres podem fazer pedidos de passagem e se existirão laços detectores de fluxo de veículos.

A primeira tarefa é decidir a quantidade de fases existentes no cruzamento. As abordagens mais comuns para uma interseção em cruz são a utilização de duas ou de quatro fases. Neste trabalho, foi escolhida a abordagem de duas fases para diminuir o tempo de

espera na interseção. A escolha da interseção de duas fases apresenta duas consequências imediatas: a limitação de conversão à esquerda em qualquer uma das vias e a escolha de um intervalo para passagem de pedestres. A Figura 6.1 representa o exposto.

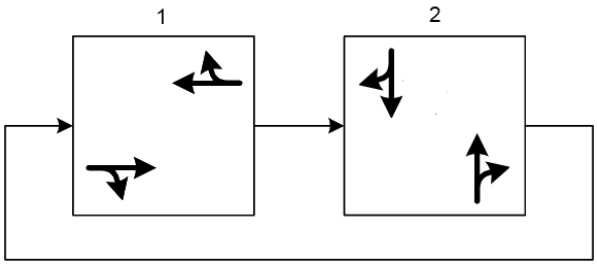

Figura 6.1: Diagrama de fases

A limitação de conversão à esquerda é preferida devido a complexidade e fatores que influenciam o controle da interseção. Para sinais de quatro fases poderia ser necessário esperar um grande tempo para cruzar a interseção. Os condutores poderiam ter de esperar as outras três vias serem liberadas, além de requisições de pedestres e esperar mais tempo se uma via for prioritária. A escolha pela interseção de duas fases garante que as vias na mesma direção e sentidos contrários estejam disponíveis por mais tempo. A escolha de duas fases não permite o fluxo de pedestres no cruzamento enquanto uma das fases estiver no estado de verde. Para resolver esse problema é necessário que em algum momento as duas fases estejam em estado de vermelho. Para este projeto, foi padronizado que, após as duas fases de verde, haverá uma fase onde o semáforo estará vemelho para as vias e será permitido o cruzamento em segurança dos pedestres.

Outra tarefa é definir a quantidade de cruzamentos (e por conseguinte a quantidade de semáforos) que farão parte do projeto. Neste trabalho, foi considerada uma rede de cruzamentos, sendo um semáforo para cada cruzamento. A utilização de apenas um cruzamento restringiria a complexidade do sistema, e desta forma, não seria possível modelar uma característica desejável em cruzamentos, como a onda verde (vários sinais verdes em sequência). Uma rede de cruzamentos é exemplificada nas Figuras 6.2, 6.3, 6.4 e 6.5.

A Figura 6.2 mostra uma região com alto fluxo de veículos na cidade de Uberlândia - MG. Esta região será utilizada como um exemplo de rede de cruzamentos. A Figura 6.3 centraliza mais a área específica da região exemplificada. A Figura 6.4 mostra a região de maneira discretizada destacando os semáforos existentes em diversos cruzamentos. A Figura 6.5 mostra um exemplo de uma rede de semáforos que pode ser considerada para modelagem deste estudo de caso.

A permissão de virada também é um fator de suma importância em um projeto semafórico. A escolha de um semáforo de dois tempos limita as opções de fluxo para uma via. Com o semáforo de dois tempos o condutor tem permissão para seguir apenas dois percursos: o condutor pode seguir em frente ou pode virar à direita. Com um semáforo de

Figura 6.2: Exemplo de uma malha urbana

Figura 6.3: Foco em uma área específica da malha urbana

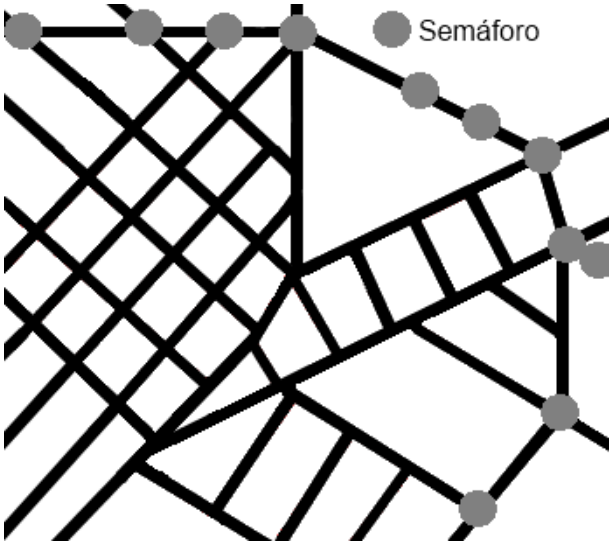

Figura 6.4: Foco em uma área específica da malha urbana com visualização de pontos de semáforo

quatro tempos o condutor teria permissão para seguir por três percursos: virar à esquerda, seguir reto ou virar à direita. Entretanto, como descrito anteriormente, esta abordagem tem limitações no tempo de espera e no tratamento das conversões.

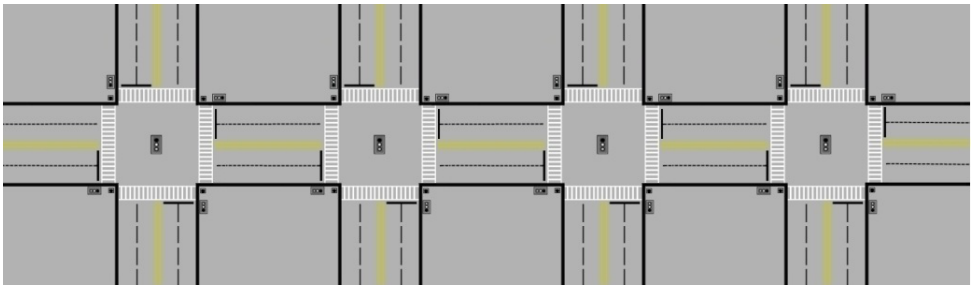

Figura 6.5: Interseção em rede

Para o tratamento da complexidade da interseção é considerada a definição de tempo prioritário para o semáforo das vias. A noção de tempo prioritário é importante para tratar algumas situações presentes no trânsito tradicionalmente, como por exemplo o cruzamento entre uma avenida movimentada e uma rua menos movimentada. Esta abordagem pode ser utilizada para tratar eventos que podem ocorrer no trânsito como a diferença de fluxo de veículos em um período do dia, diferença de fluxo de veículos em um período do ano ou em algum evento pontual.

Uma etapa importante de um projeto de interseção é decidir sobre o fluxo de pedestres. Para facilitar o fluxo de pedestres, a modelagem deste estudo de caso permitirá a passagem prioritária de pedestres pelo cruzamento. Assim que os pedestres fizerem o pedido de passagem, esse pedido será analisado para verificar quando pode ser atendido. Algumas variáveis devem ser consideradas para este propósito como o tempo de verde do semáforo da via e outros pedidos de pedestres nas vias concorrentes. O sistema foi dotado de quatro pontos de requisição de passagem de pedestres (*push button*), sendo cada um dotado de dois painéis, totalizando oito possibilidades de requisição.

A modelagem realizada pressupõe que existem sensores em todas as quatro vias da interseção. Os sensores são utilizados com o objetivo de analisar o fluxo de veículos no local. Os sensores fazem a contagem do número de veículos provenientes de uma via chegando na interseção em um determinado momento. Através desta análise é possível definir políticas específicas para determinados momentos e eventos que podem ocorrer e que impactam o fluxo da interseção.

Existem diversas políticas de controle da interseção, como a política de tempo fixo, a política de tempo atuado e a política de tempo adaptativo. A definição de qual política será utilizada será feita arbitrariamente pelo controlador da interseção ou de maneira automática dependendo das considerações de fluxo de veículos apresentadas anteriormente.

A Figura 6.6 representa uma visão geral em alto nível da utilização do sistema e os seus impactos nos semáforos. A figura foca nas interações que os usuários realizam sobre o controlador e nas mudanças refletidas nos semáforos. As principais interações são representadas pela manutenção remota do sistema, a mudança de política de gerenciamento de tráfego, a chegada de veículos na interseção e a requisição de pedestres por passagem. Essas interações sobre o controlador da interseção são responsáveis pela alteração dos

estados dos semáforos, tanto o semáforo dos veículos quanto o semáforo dos pedestres.

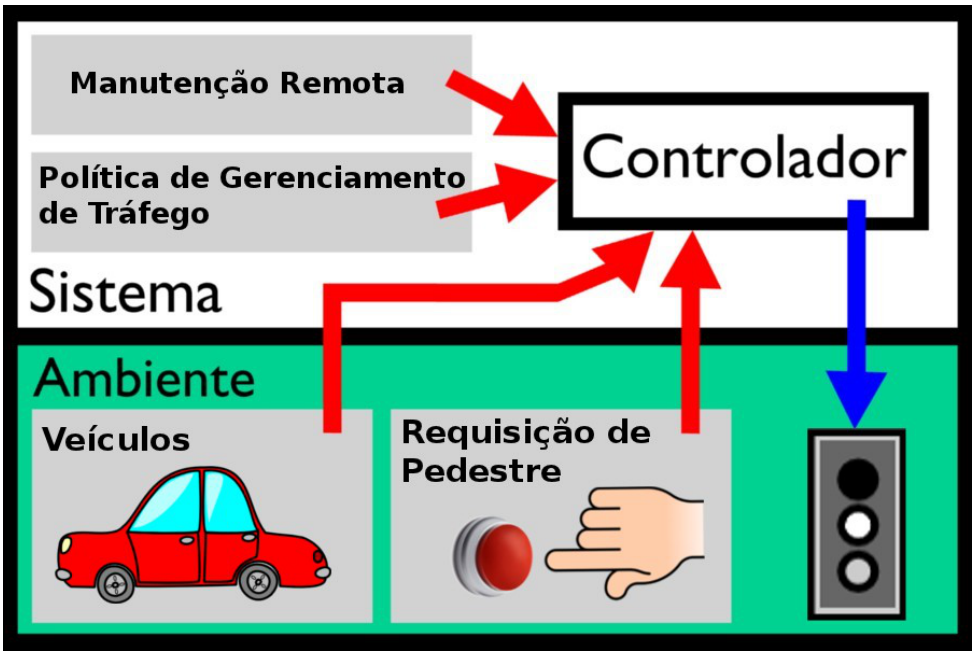

Figura 6.6: Controlador da interseção

Uma lista de requisitos funcionais em nível de usuário foi definida para o problema abordado. Os requisitos são apresentados na Tabela 6.1.

Requisitos Funcionais em Nível de Usuário
R1 - O sistema deve controlar o padrão de tráfego de veículos na interseção.
R2 - O sistema deve controlar o padrão de tráfego de pedestres na interseção.
R3 - O sistema deve armazenar o fluxo de veículos das vias.
R4 - O sistema deve controlar o padrão de tráfego relacionado a cada via.
R5 - O sistema deve permitir política de gerenciamento de tráfego fixo.
R6 - O sistema deve permitir política de gerenciamento de tráfego atuado.
R7 - O sistema deve permitir política de gerenciamento de tráfego adaptativo.
R8 - O sistema deve permitir sincronização de semáforos.
R9 - O sistema deve permitir escolher uma via prioritária.
R10 - O sistema deve permitir detecção de presença de pedestres.
R11 - O sistema deve permitir manutenção presencial.
R12 - O sistema deve permitir manutenção remota.
R13 - O sistema deve manter o histórico do tráfego de veículos das vias.
R14 - O sistema deve manter o histórico das políticas de tráfego nos períodos do ano.
R15 - O sistema deve ser capaz de implementar novas políticas de tráfego.
R16 - O sistema deve armazenar os incidentes ocorridos no cruzamento.
R17 - O sistema deve permitir a operação automática dos semáforos.
R18 - O sistema deve armazenar os incidentes ocorridos no software e no hardware.

Tabela 6.1: Requisitos Funcionais em Nível de Usuário com Alto Nível de Abstração

O diagrama de casos de uso correspondente aos cenários de utilização do sistema é ilustrado na Figura 6.7. A Figura 6.8 mostra um diagrama de classes em nível de análise (apenas identificação das classes) para os requisitos apresentados anteriormente.

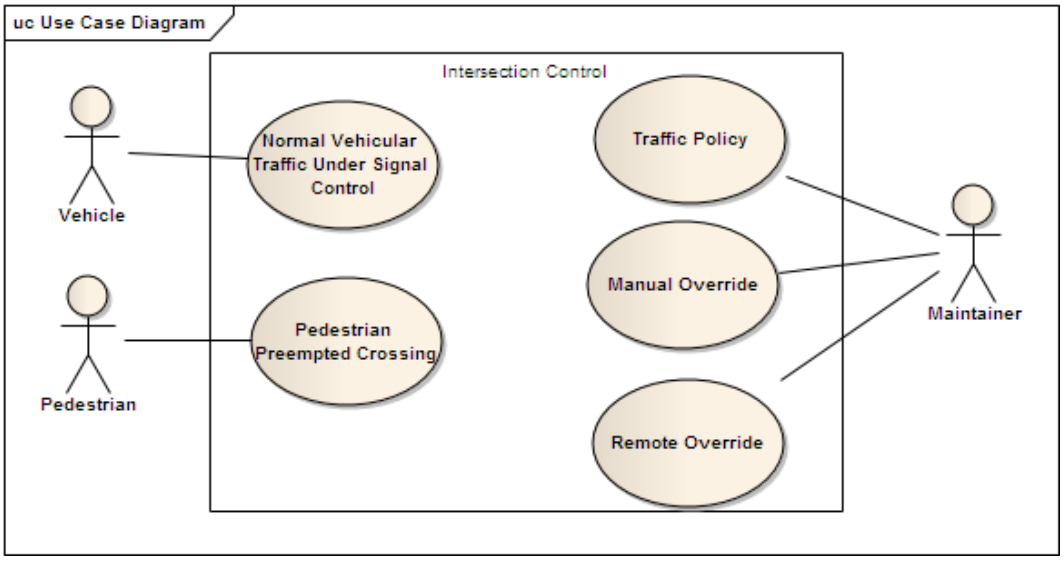

Figura 6.7: Diagrama de casos de uso

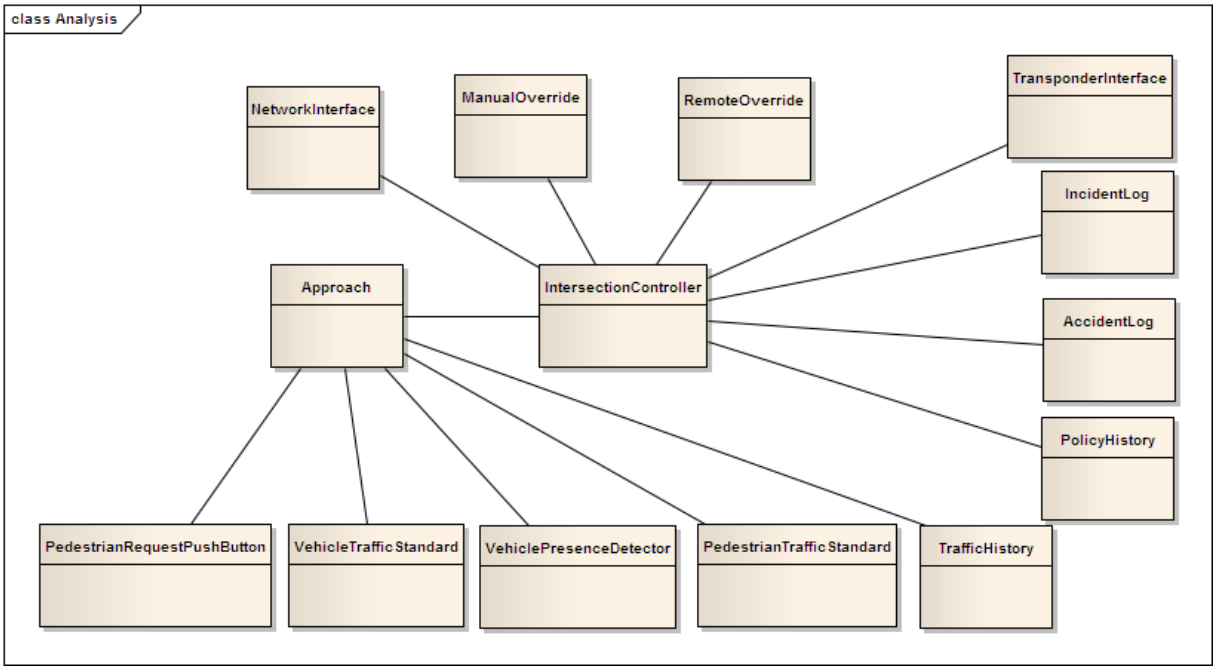

Figura 6.8: Diagrama de classes de análise

6.2 Modelagem estrutural

O projeto estrutural descrito neste capítulo é realizado através da modelagem das visões arquiteturais lógica (proposta por [Kruchten 2003]), visões de recursos compartilhados e alocação de recursos propostas no capítulo anterior. O projeto estrutural é descrito neste trabalho com a utilização do diagrama de classes e do diagrama de implantação. O diagrama de classes utilizando a UML e as extensões do MARTE representa a visão lógica. Os diagramas de implantação utilizando a UML e as extensões do MARTE representam as visões de recursos compartilhados e alocação de recursos.

O diagrama de classes representa os requisitos funcionais e um conjunto de abstrações

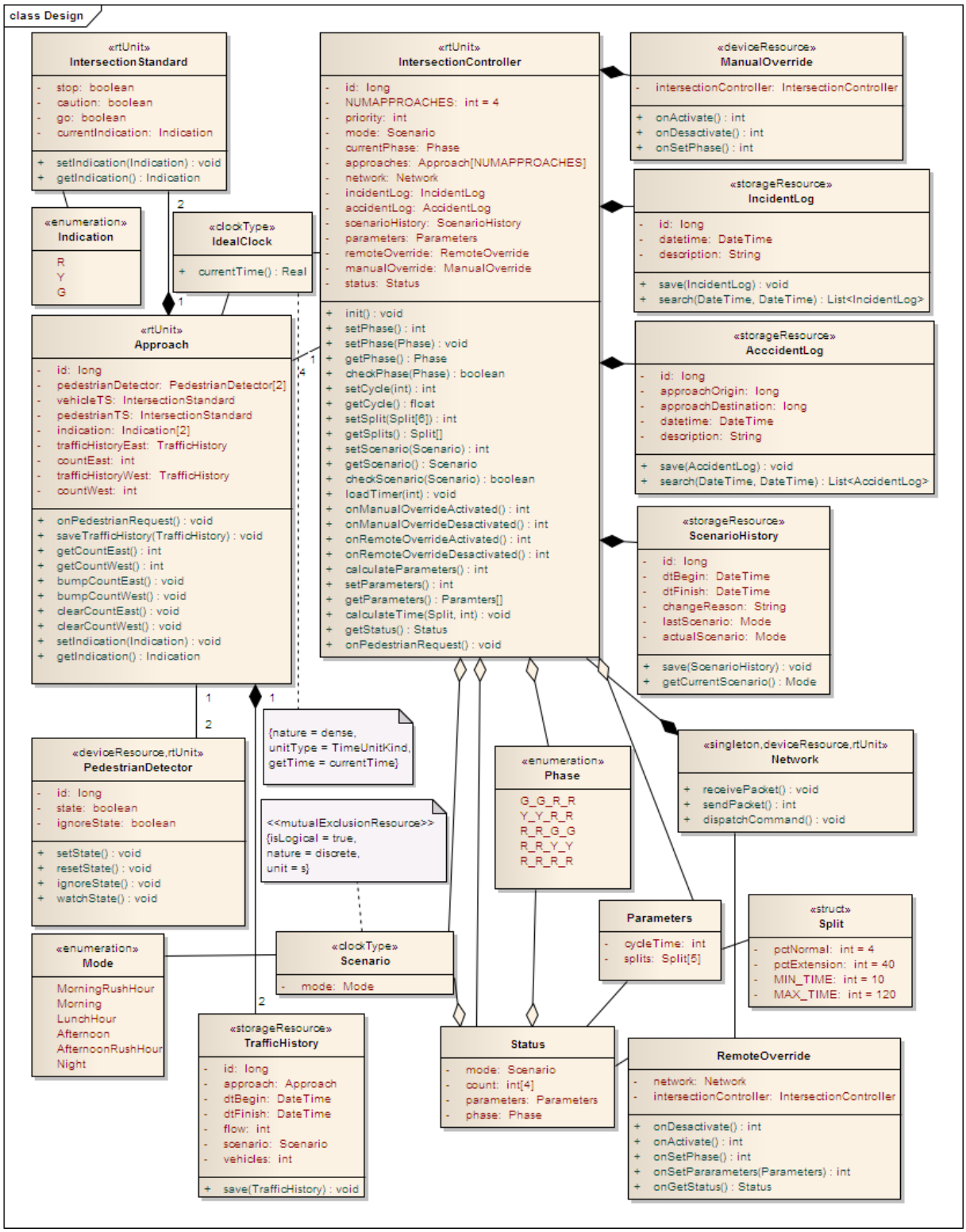

Figura 6.9: Diagrama de classes com extensões do MARTE

na forma de objetos. O diagrama de classes UML (em nível de projeto) utilizando as extensões do MARTE é descrito na Figura 6.9. A estrutura do diagrama de classes é centralizada principalmente nas classes *IntersectionController* e *Approach*. A classe *IntersectionController*, controladora da interseção, é um *singleton* responsável por gerenciar as principais características dos sinais de trânsito. Ela contém relacionamentos com as

vias e com as fases. A classe *Approach*, que representa as vias, é responsável por gerenciar as características individuais de cada via da interseção. Ela contém referências para semáforos e painéis de requisição de pedestres.

A aplicação do *profile* MARTE no diagrama de classes é concentrada na utilização de novos estereótipos. O estereótipo <<*rtUnit*>> é usado para representar classes ativas. O estereótipo <<storageResource>> é usado para representar classes de entidade. O estereótipo <<deviceResource>> representa um dispositivo externo que pode ser manipulado ou invocado pela plataforma de execução.

O estereótipo <<clockType>> está relacionado com o tempo, uma característica crucial em sistemas de tempo real. O estereótipo cria um novo *clock*. Neste estudo de caso, dois novos tipos de *clock* são criados e adicionados ao diagrama de classes. O primeiro é o tipo predefinido *IdealClock*. O *IdealClock* modela o tempo ideal e abstrato usado nas leis da física, isto é, ele é uma forma de tempo denso. O *IdealClock* deveria ser importado em modelos que se referem a tempo físico. O segundo é usado para representar um cenário. O *Scenario* representa um cenário de utilização do sistema e as possíveis políticas que podem ser aplicadas ao controle dos sinais de trânsito da interseção. Os vários valores - referências a quantidades - mostrados no diagrama de classes são descritos usando o pacote *VSL* (subseção 4.1.4).

A representação de recursos e sua posterior alocação são características importantes no desenvolvimento de software de tempo real. A definição de recursos é importante para tratar aspectos como concorrência, elementos distribuídos e paralelismo. A alocação de recursos representa a alocação de elementos da aplicação funcional em recursos disponíveis. O diagrama mais adequado da UML para representação de recursos e posterior alocação desses recursos é o diagrama de implantação. O diagrama de implantação modela a configuração de tempo de execução em uma visão estática e também visualiza os componentes distribuídos na aplicação. A Figura 6.10 descreve um diagrama de implantação para representar recursos e as Figuras 6.11 e 6.12 descrevem diagramas de implantação para representar a alocação de recursos.

A Figura 6.10 refere-se à arquitetura física do software para o controlador da interseção. Ela contém os principais componentes do sistema, como o sistema operacional, o software da aplicação, a rede e componentes físicos onde o sistema é executado.

Neste estudo de caso, o estereótipo <<communicationMedia>> é usado em várias partes do diagrama de implantação. Ele representa o meio de transportar dados de um lugar para outro. Desta forma, o seu uso representa o fluxo de dados. Os estereótipos <<deviceResource>> e <<storageResource>> são os mesmos descritos no diagrama de classes. O estereótipo <<computingResource>> representa dispositivos de processamento - tanto virtuais quanto físicos - capazes de armazenar e executar programas. Neste estudo de caso, por exemplo, o sistema operacional de tempo real é considerado um <<computingResource>>.

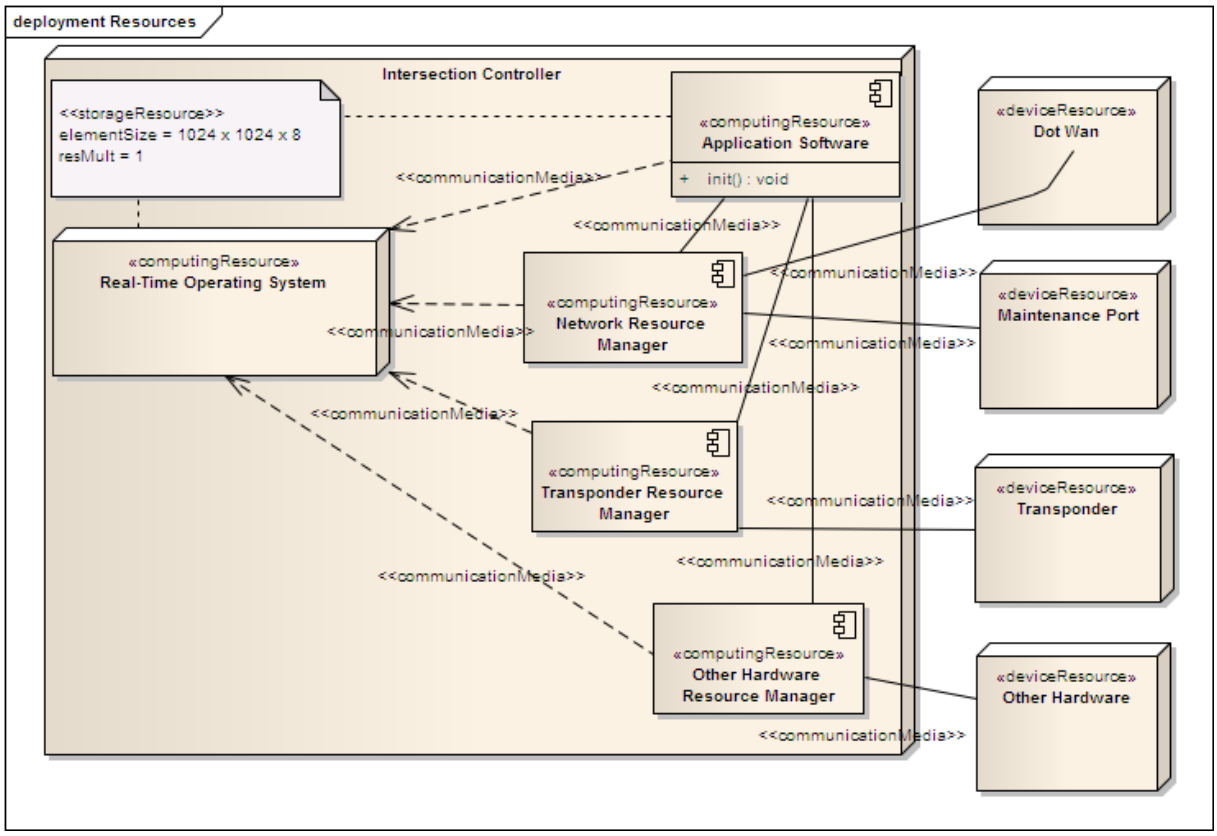

Figura 6.10: Diagrama de implantação com extensões do MARTE para representação de recursos

A Figura 6.11 mostra a alocação de recursos aplicada sobre a arquitetura física do software para o controlador da interseção. A figura contém as mesmas extensões presentes na Figura 6.10 e as extensões para representar a alocação de recursos. Para isto, o estereótipo <<allocate>> é usado. O estereótipo é uma ponte entre elementos de um contexto mais lógico para elementos de um contexto mais físico. O atributo *nature* é uma enumeração que define literais usados para especificar o propósito da alocação. O valor rotulado $\{nature = timeScheduling\}$ indica que a alocação consiste de uma ordenação temporal/comportamental para os fornecedores, sendo a ordenação dada pelos clientes. Além disso, a figura mostra os valores rotulados $\{kind = executionPlatform\}$ e $\{kind = application\}$. O atributo *kind* é uma enumeração que diferencia o objetivo fim da alocação. O valor rotulado $\{kind = executionPlatform\}$ identifica uma alocação do lado da plataforma de execução da aplicação. O valor rotulado $\{kind = application\}$ identifica que a alocação fim está sendo feita no lado da alocação da aplicação.

A Figura 6.12 mostra um exemplo de alocação de recursos em três camadas. A primeira camada descreve a aplicação, a segunda camada representa um sistema operacional de tempo real e a última camada mostra as partes de hardware.

A camada do topo é a visão da aplicação. Essa visão tem os mesmos componentes mostrados na Figura 6.10. O estereótipo <<allocated>> é aplicado a elementos que tem pelo menos um relacionamento de alocação com outro elemento. O valor rotulado $\{kind$

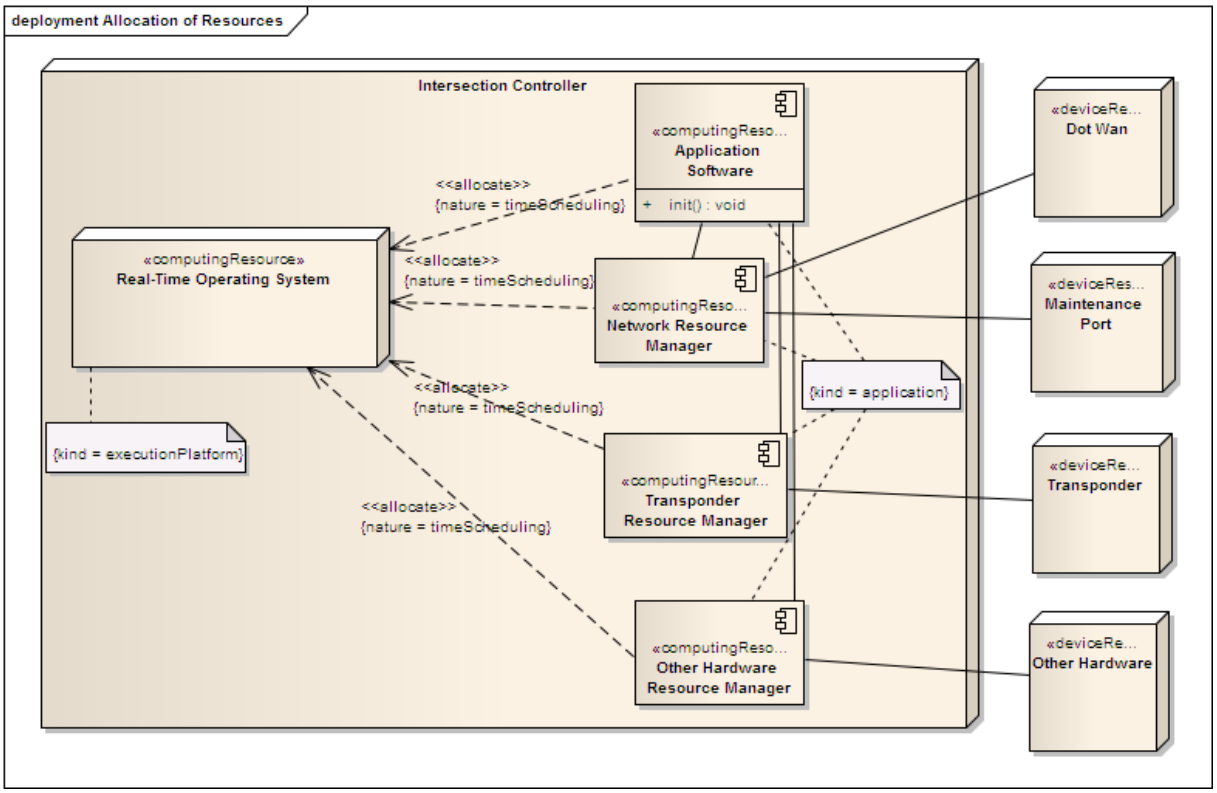

Figura 6.11: Diagrama de implantação com extensões do MARTE para representação de alocação de recursos

$= application\}$ foi descrito anteriormente.

A camada intermediária, o sistema operacional de tempo real, não é o foco deste estudo de caso. Ele suporta a alocação em diferentes níveis de abstração. O sistema operacional de tempo real está relacionado ao hardware através do estereótipo <<allocate>> e do valor rotulado $\{nature = timeScheduling\}$ (descritos anteriormente).

A camada inferior no diagrama representa as partes do hardware. CPU, memória, barramento e disco são anotados com estereótipos supracitados, <<computingResource>>, <<storageResource>>, <<allocated>> e <<communicationMedia>>. Todos os elementos tem o valor rotulado $\{kind = executionPlatform\}$. Além disso, essa camada mostra o relacionamento <<allocate>> entre os componentes. Esse estereótipo é anexado com os valores rotulados $\{nature = spatialDistribution\}$ e $\{nature = timeScheduling\}$. O *timeScheduling* foi descrito anteriormente e o *spatialDistribution* indica que os fornecedores são distribuídos nos clientes. Distribuição espacial é a alocação de computações para elementos de processamento e de dados para memórias.

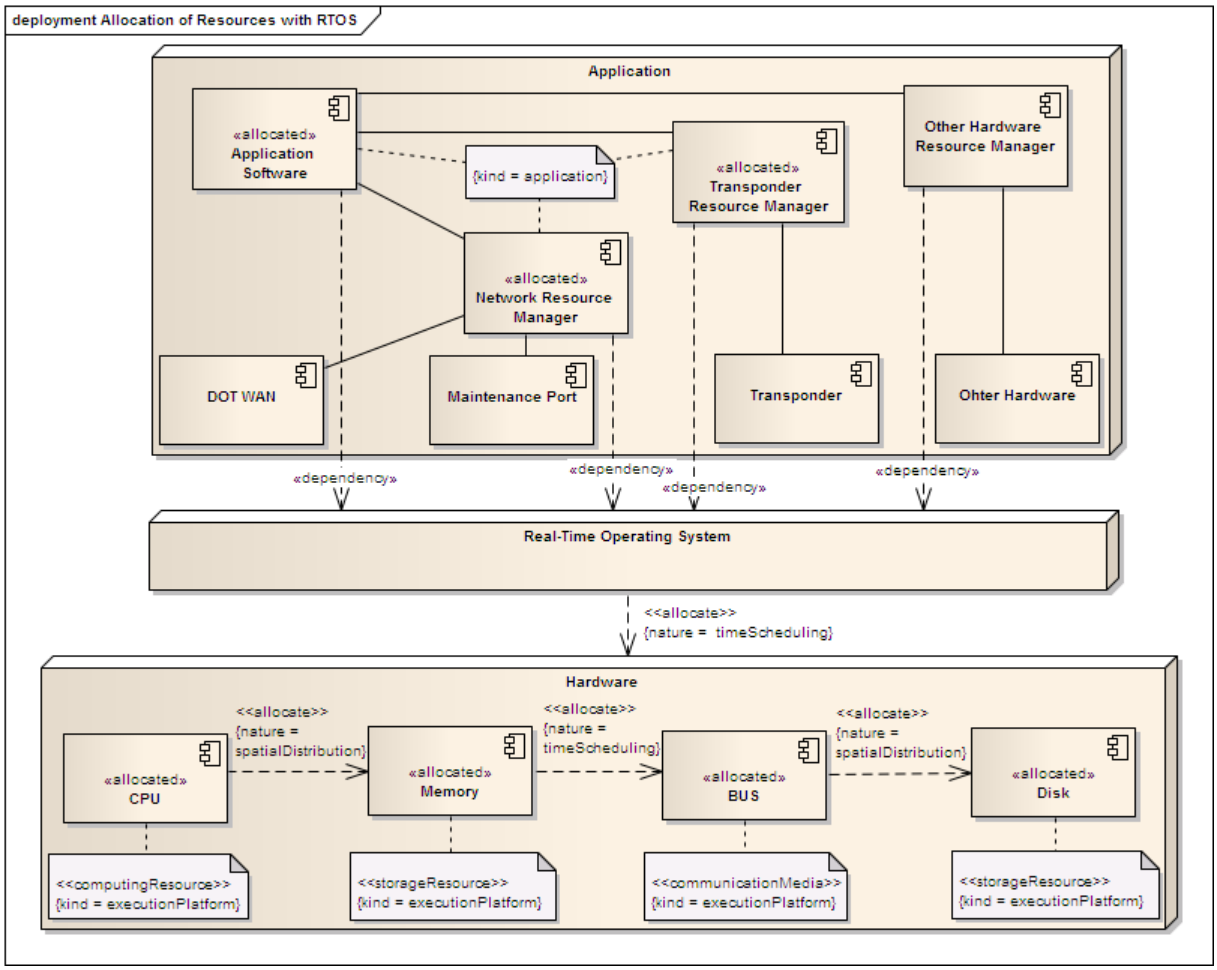

Figura 6.12: Diagrama de implantação com extensões do MARTE para representação de alocação de recursos

6.3 Modelagem comportamental

O projeto comportamental descrito neste capítulo é realizado através da modelagem das visões arquiteturais de tempo e de processos propostas no capítulo anterior. O projeto comportamental é descrito neste trabalho com a utilização do diagrama de sequência, diagrama de máquina de estados e diagrama de tempo. Os diagramas de sequência utilizando a UML e as extensões do MARTE representam as visões de tempo e de processos. Os diagramas de máquina de estados utilizando a UML e as extensões do MARTE representam a visão de processos. O diagrama de tempo utilizando a UML e as extensões do MARTE representa a visão de tempo.

O diagrama de sequência é aplicado para modelar o fluxo dentro do sistema em uma maneira visual, com foco na identificação do seu comportamento. As Figuras 6.13 e 6.14 mostram a utilização do diagrama de sequência utilizando as extensões do MARTE.

O diagrama de sequência da Figura 6.13 mostra a operação normal do sistema controlador de sinais de trânsito. O diagrama descreve a operação desde a primeira ação até o fluxo normal do software.

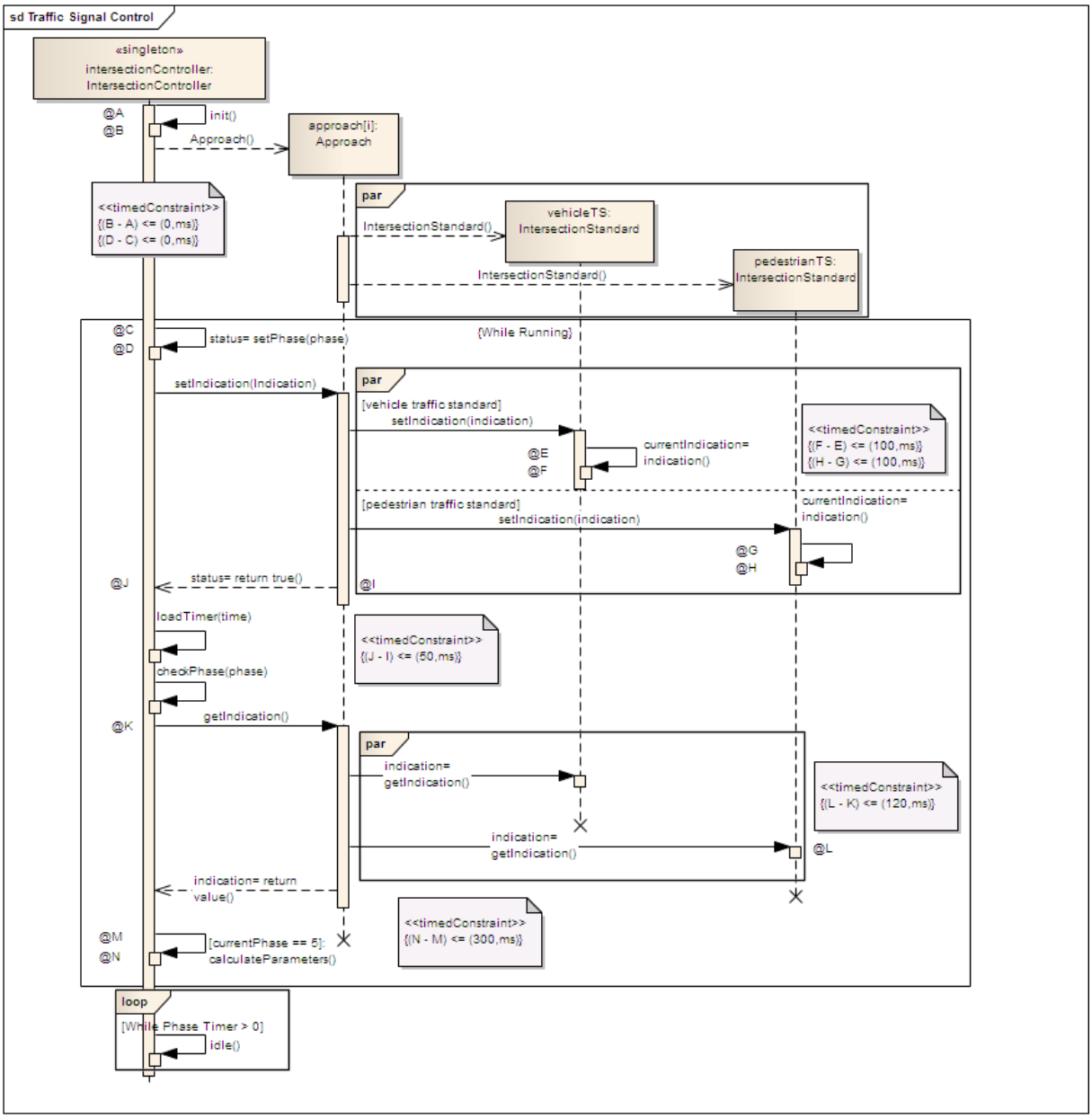

Figura 6.13: Diagrama de sequência com extensões do MARTE representando a operação do sistema

Para esse propósito, o diagrama de sequência mostra as classes *IntersectionController*, *Approach* e *IntersectionStandard*. A classe *IntersectionController* gerencia o software. A classe *Approach* representa uma via específica na interseção e a classe *IntersectionStandard* representa os sinais de trânsito. O semáforo de veículos é representado pelo objeto *vehicleTS* e o semáforo de pedestres é representado pelo objeto *pedestrianTS*. A figura mostra a inicialização do sistema, a criação de objetos e a troca de mensagens entre os objetos.

O diagrama de sequência é concentrado na representação de tempo. Para a realização deste objetivo, o estereótipo <<timedConstraint>> é usado junto com o pacote *VSL*. O estereótipo <<timedConstraint>>, representado pela classe *TimedConstraint*, é

uma superclasse abstrata de *TimedInstantConstraint* e *TimedDurationConstraint*. Este estereótipo permite restringir quando um evento pode ocorrer, restringir a duração da execução de um evento ou restringir o intervalo entre eventos.

O diagrama de sequência da Figura 6.14 utiliza basicamente as mesmas extensões do diagrama de sequência da Figura 6.13 ou extensões apresentadas anteriormente, como o estereótipo <<deviceResource>> aplicado ao objeto *pedestrianDetector*. A figura mostra os métodos disparados quando o pedestre pressiona o botão pedindo passagem. A classe *PedestrianDetector* é responsável por gerenciar os pedidos de passagem do pedestre.

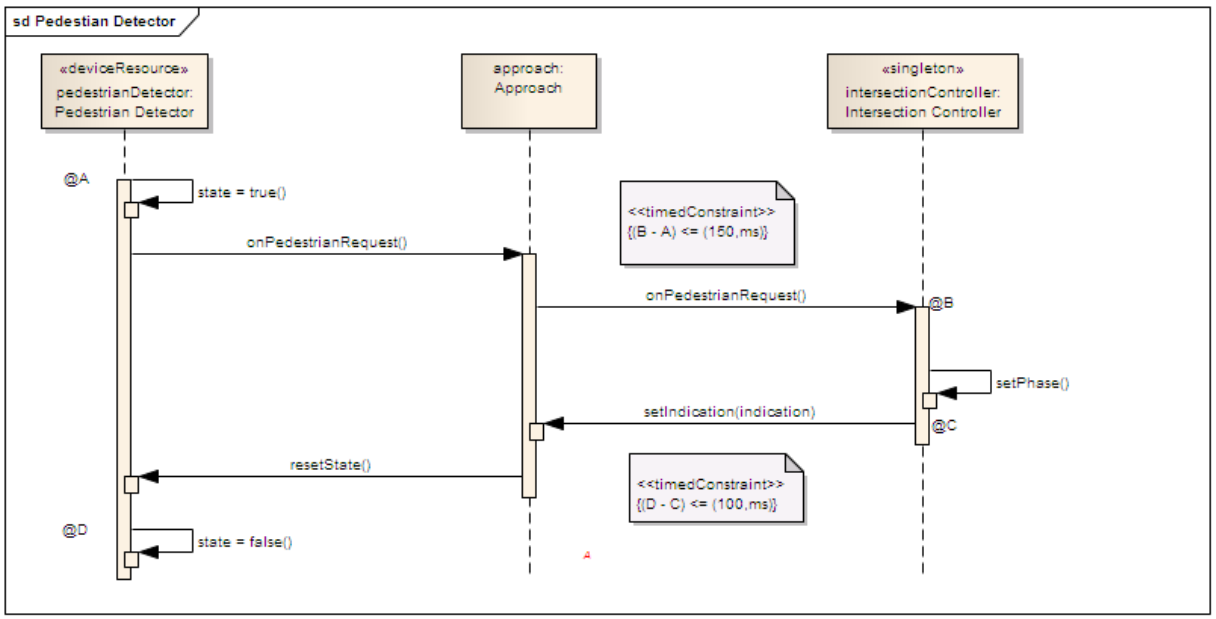

Figura 6.14: Diagrama de sequência com extensões do MARTE representando a interação com os pedestres

O diagrama de máquina de estados é útil no domínio de sistemas de tempo real. Ele permite a descrição do comportamento do sistema em termos dos estados e das transições entre esses estados. As Figuras 6.15, 6.16, 6.17 e 6.18 mostram diagramas de máquinas de estados UML utilizando as extensões do MARTE.

A aplicação do *profile* MARTE no diagrama de máquina de estados é concentrada em novos estereótipos. Os estados são representados pelos estereótipos <<*mode*>>. Esse estereótipo identifica um segmento operacionado dentro da execução de um sistema. As transições entre estados é representada pelo estereótipo <<*modeTransition*>>. Esse estereótipo descreve o sistema modelado que está sob troca de estados. A máquina de estados é representada por dois estereótipos. O estereótipo <<*modeBehavior*>> especifica um conjunto de modos mutualmente exclusivos, isto é, apenas um modo pode estar ativo em um dado instante do tempo. O estereótipo <<*timedProcessing*>> representa atividades que tem início/fim conhecido ou uma duração conhecida.

O diagrama de máquina de estados na Figura 6.15 mostra os estados *Waiting Signal* e *Pedestrian Crossing*. O primeiro representa o momento em que o pedestre está esperando

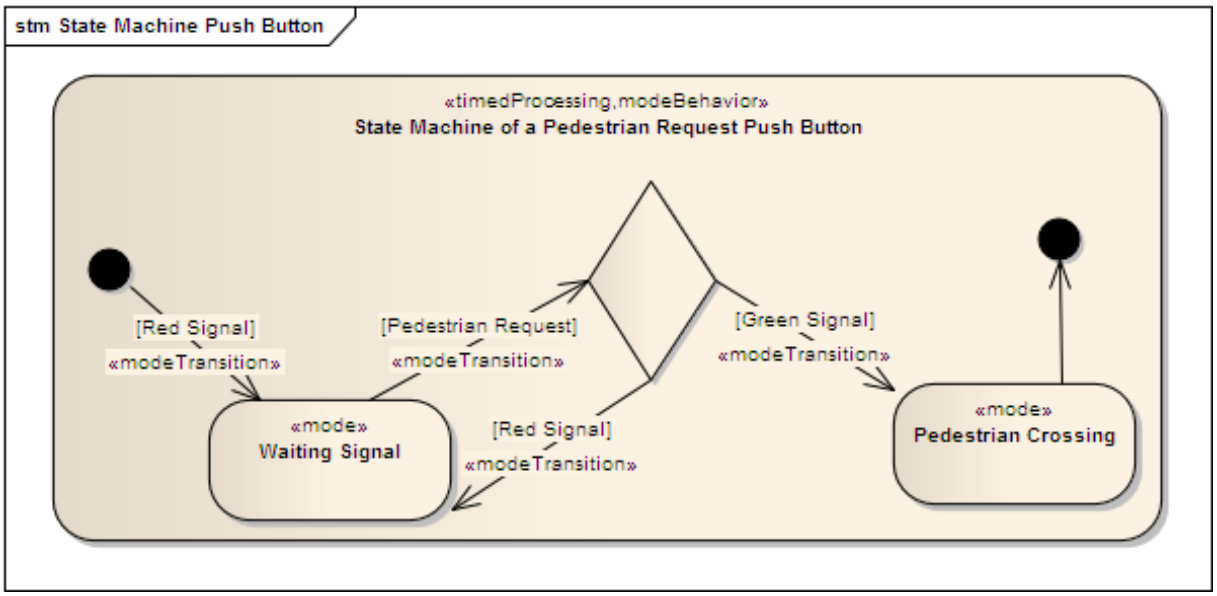

Figura 6.15: Diagrama de máquina de estados com extensões do MARTE

pelo sinal de trânsito. O segundo representa o momento no qual o pedestre está atravessando o sinal. A transição entre os dois estados representa como resposta a requisição do pedestre por passagem, ou seja, quando o pedestre pressiona o botão lateral requisitando a permissão de passagem.

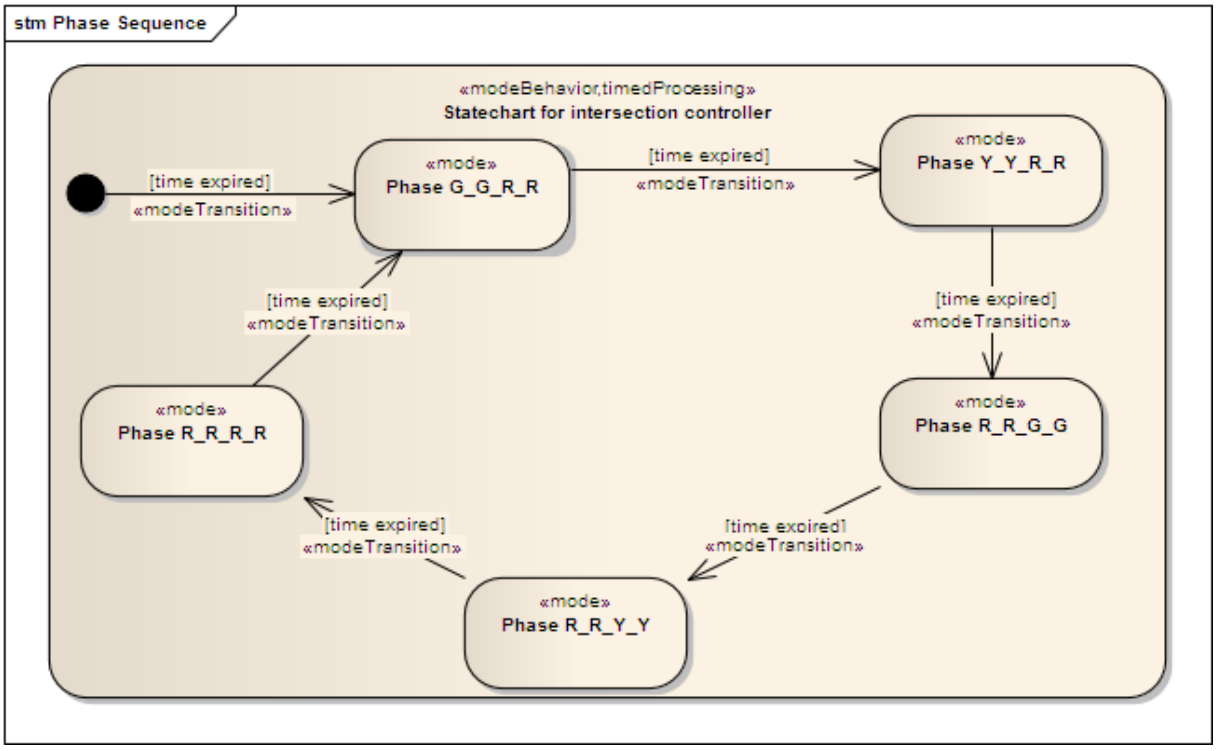

Figura 6.16: Diagrama de máquina de estados com extensões do MARTE

O diagrama de máquina de estados da Figura 6.16 mostra os vários estados de um semáforo e as transições feitas após um período de tempo determinado. O estado *Phase G_G_R_R* representa o momento em que o semáforo está na fase de verde e seu concorrente

está na fase de vermelho. O estado *Phase Y_Y_R_R* representa o momento em que o semáforo está na fase de amarelo e o seu concorrente continua na fase de vermelho. Os semáforos *Phase R_R_G_G* e *Phase R_R_Y_Y* representam o oposto do primeiro e do segundo semáforo supracitados, respectivamente. O estado *Phase R_R_R_R* representa o momento em que os dois semáforos estão na fase de vermelho, permitindo a passagem dos pedestres.

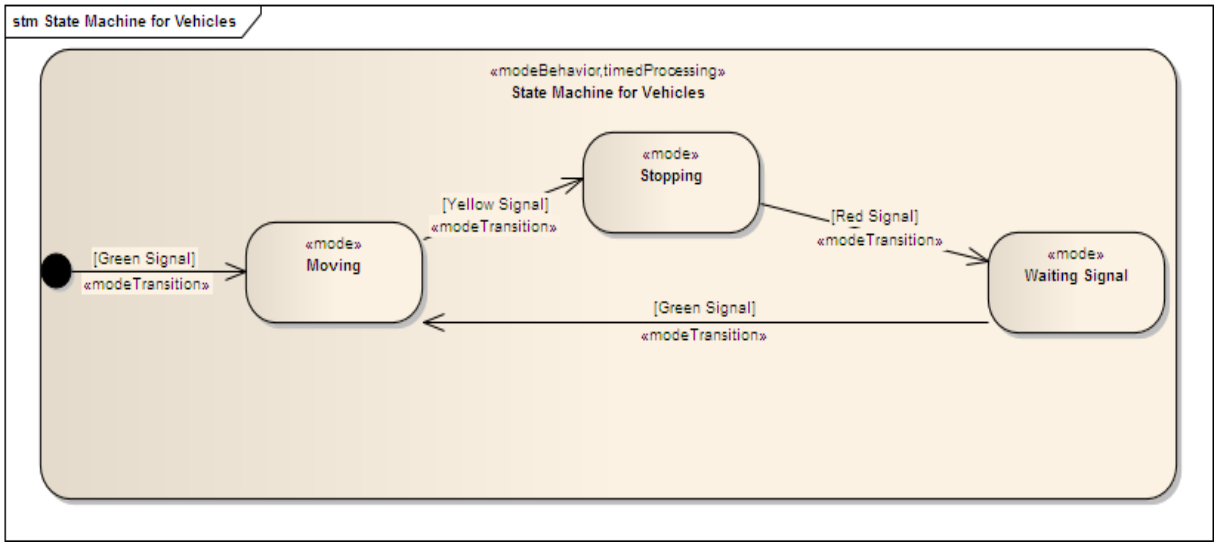

Figura 6.17: Diagrama de máquina de estados com extensões do MARTE

O diagrama de máquina de estados da Figura 6.17 representa o fluxo de veículos e a sua chegada na interseção. Em um dado momento, os veículos estão se movendo. Neste momento, os veículos estão no estado *Moving*. Se o semáforo está exibindo o sinal de amarelo então é o momento de começar a parar, modificando os seus estados para *Stopping*. Após o sinal vermelho ser mostrado, é o momento de esperar, movendo o estado para *Waiting Signal*. Após esperar um determinado tempo a fase de verde do semáforo será acionada e o semáforo pode recomeçar o ciclo indo para o estado de *Moving*.

O diagrama de máquina de estados da Figura 6.18 mostra o estado de sincronismo. O estado de sincronismo pode ser utilizado quando estados de regiões diferentes estejam de alguma forma sincronizados, por vezes sendo necessário que o estado de uma região espere por um estado de outra. Neste exemplo, há dois sinais de trânsito. No momento em que o sinal de um muda, o outro deve mudar também, automaticamente. Assim, o estado do primeiro sinal de trânsito é representado como *Open Signal*, e do segundo, como *Closed Signal*. Quando o primeiro sinal de trânsito recebe a ordem de trocar seu sinal, este gera duas transições, uma para trocar seu próprio estado e outra para avisar ao estado de sincronismo para alterar o estado do segundo sinal. Assim, no momento em que o estado do primeiro sinal muda para *Closed Signal*, o estado do segundo sinal automaticamente terá que mudar para *Open Signal*.

O diagrama de tempo é usado para explorar os comportamentos de um ou mais objetos em um dado período do tempo. Diagramas de tempo são frequentemente usados no

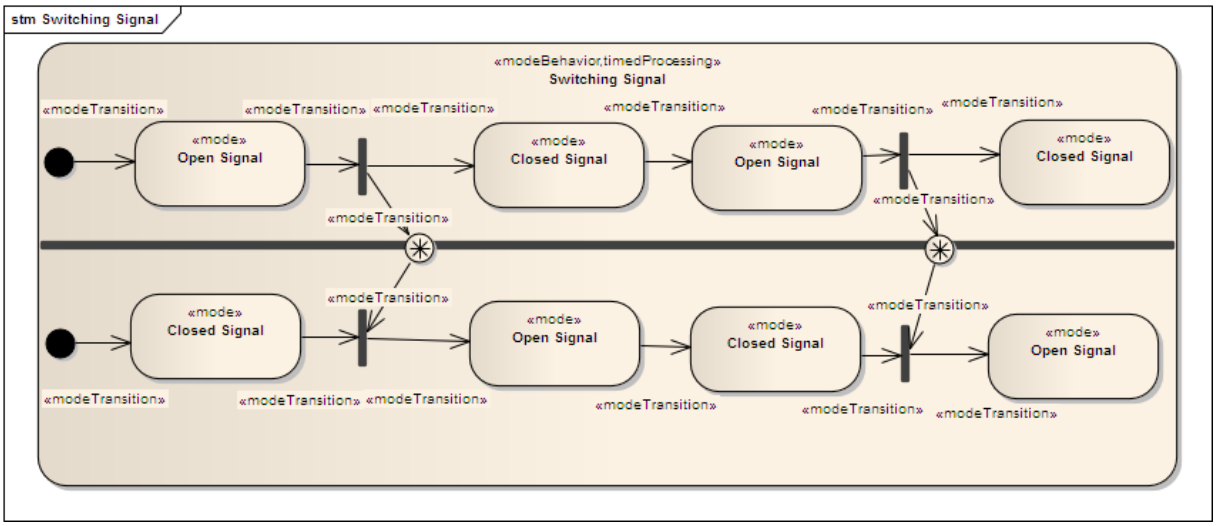

Figura 6.18: Diagrama de máquina de estados com extensões do MARTE

projeto de software de tempo real. Um diagrama de tempo UML para sinais de trânsito é descrito na Figura 6.19. Ele mostra que pedestres tem o direito de passagem quando ambos os sinais de trânsito em sentidos diferentes estão no estado vermelho. A Figura 6.20 complementa o diagrama de tempo.

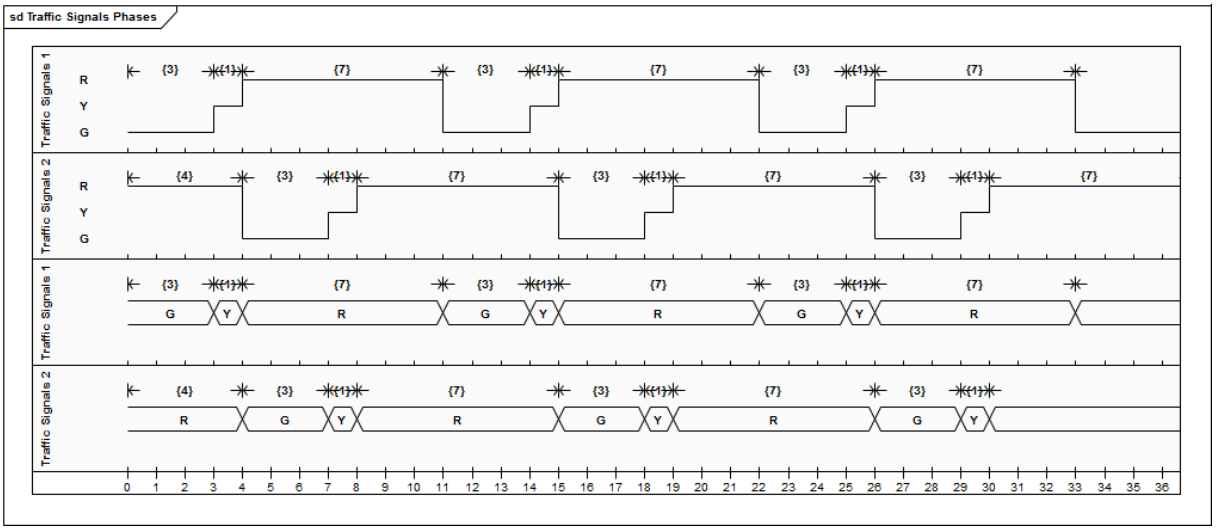

Figura 6.19: Diagrama de tempo

A Figura 6.20 representa os semáforos como elementos do tipo <<*clockType*>> e sua representação de seu objeto estereotipado com <<*clock*>>. Além disso, estabelece uma restrição entre os dois *clocks*. A ocorrência de um *clock* deve ser exclusiva em relação a ocorrência de outro *clock*.

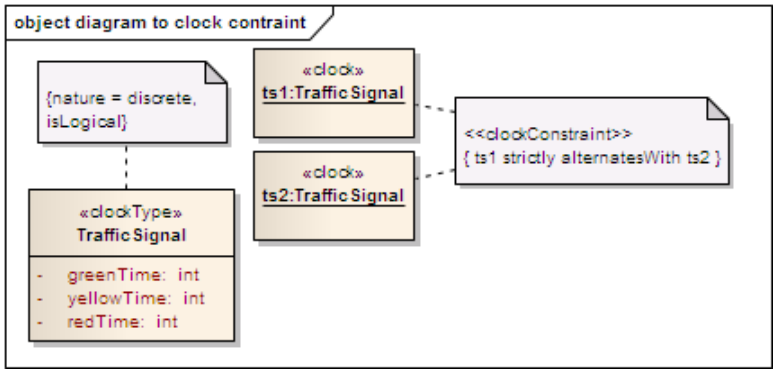

Figura 6.20: Restrições temporais para o diagrama de tempo

6.4 Conclusões sobre a modelagem de software utilizando o *profile* MARTE

Nas seções anteriores foi modelado o software controlador da interseção proposto neste trabalho. A Seção 6.1 definiu os requisitos pertencentes ao estudo de caso. A Seção 6.2 realizou a modelagem estrutural da aplicação. Para isso, foram utilizadas as visões arquiteturais de recursos compartilhados e alocação de recursos (propostas no Capítulo 5) e a visão lógica da arquitetura de referência "4+1". Para a representação das visões arquiteturais foram utilizados os diagramas da UML de classes e de implantação. A Seção 6.3 realizou a modelagem comportamental da aplicação. Para isso, foram utilizadas as visões arquiteturais de tempo e de processos (propostas no Capítulo 5). Para a representação das visões arquiteturais foram utilizados os diagramas UML de sequência, máquina de estados e de tempo. Em todas as visões, diversos estereótipos do *profile* MARTE foram aplicados na especificação dos modelos UML.

Capítulo 7

Conclusões

O desenvolvimento de software de tempo real é uma atividade complexa, desafiadora e passível de inúmeros problemas não encontrados no desenvolvimento de software de propósito geral. Sabe-se que a modelagem de software pode ser aplicada como tentativa de diminuir os problemas no desenvolvimento de software, uma vez que os modelos ajudam a descobrir os problemas antes da sua implementação. A modelagem também pode ser aplicada no contexto de software de tempo real. Entretanto, a modelagem de software de tempo real apresenta algumas características não presentes na modelagem de software de propósito geral. Desta forma, a modelagem de software de tempo real é uma área com diversas oportunidades de pesquisa (Capítulo 1).

Diversas linguagens de modelagem foram aplicadas para a modelagem de software de tempo real, como as extensões da análise estruturada, métodos formais, UML e SPT. Entretanto, essas linguagens não podem ser consideradas um padrão nesta área, devido principalmente às dificuldades existentes nestas linguagens para representar características de tempo real (Capítulo 2). Neste cenário, a OMG propôs um novo *profile* que estende a UML e substitui o SPT para modelagem de software de tempo real (Capítulo 4). Este *profile* adiciona extensões a UML, como estereótipos, valores rotulados e restrições para modelar software de tempo real.

Um exemplo de um sistema de tempo real é um sistema controlador de semáforos de trânsito (Capítulo 3). Esse sistema tem características representativas de sistemas de tempo real, como exemplo, o tempo rigoroso para a execução de tarefas (Capítulo 2). Para avaliar a capacidade do MARTE em modelar software de tempo real, neste trabalho, MARTE foi utilizado junto com a UML (complementando alguns aspectos historicamente considerados fracos na UML, incluindo modelagem de tempo, a modelagem de recursos e a modelagem de processos) para o projeto de software de um sistema controlador de semáforos de trânsito (Capítulo 6). Uma arquitetura de referência de software de tempo real (Capítulo 5) foi proposta para servir de base para a aplicação dos diagramas UML propostos segundo visões estáticas e dinâmicas (Capítulo 6).

A utilização das extensões do *profile* MARTE no estudo de caso proposto neste traba-

lho revelaram algumas melhorias não presentes em linguagens de modelagem anteriores. MARTE representou melhor características como modelagem de tempo, processos e recursos do que a UML e o SPT (Capítulos 4 e 6).

Apesar das melhorias trazidas pelo MARTE, o estudo detalhado e a aplicação das suas extensões revelaram alguns problemas passíveis de melhorias para próximas versões (Capítulo 4). A linguagem é considerada bastante complexa e a quantidade excessiva de estereótipos gera confusão em sua aplicação. Devido a alta quantidade de estereótipos alguns deles são redundantes. Por exemplo, é possível anotar uma única característica da UML com diversas extensões. Existem ainda poucas ferramentas computacionais que adicionam as extensões do *profile*. As ferramentas gratuitas, por exemplo, ainda precisam de melhorias em vários aspectos, como melhor representação das características do MARTE, configuração mais simples e melhor usabilidade. Devido a sua complexidade, tornar-se proficiente na linguagem é um desafio.

É importante ressaltar que MARTE é um *profile* extensível. Isso significa que novos estereótipos podem ser criados e adicionados ao *profile* quando necessário, o que também aumenta sua complexidade. A aplicação industrial do MARTE ainda é muito incipiente e, mesmo na academia, a linguagem não é bem conhecida. Isso significa que mais avaliações e discussões da sua aplicabilidade são necessárias. Uma primeira tentativa de tentar comparar UML, SPT e MARTE foi mostrada neste trabalho, mas ainda é necessário um número maior de aplicações e comparações, sendo esta uma área com inúmeras oportunidades de pesquisa.

7.1 Contribuições

Após a execução deste trabalho, algumas contribuições foram realizadas para a comunidade científica. As principais foram:

- Proposta de uma arquitetura de software em múltiplas visões para ser aplicada no projeto de sistemas distribuídos de tempo real, no qual MARTE e UML são utilizados como linguagens de modelagem. A arquitetura proposta é complementar a outros modelos de arquitetura em múltiplas visões;
- A utilização do *profile* MARTE em um estudo de caso. Além disso, pela primeira vez a UML juntamente com o *profile* MARTE foi utilizada para modelagem de sinais de trânsito, de acordo com a revisão de literatura;
- Comparação entre as linguagens utilizadas para modelagem de software de tempo real. Especificamente, a comparação entre UML, SPT e MARTE.

A pesquisa também teve como resultado a publicação de artigos em congressos científicos:

- *"Multiple View Architecture Model for Distributed Real-Time Systems Using MARTE"* para a *20th International Conference on Information Systems Development (ISD 2011)*, apresentado em Agosto de 2011 na Escócia [Silvestre e Soares 2011];
- *"Modeling Road Traffic Signals Control using UML and the MARTE Profile"* para a *12th International Conference on Computational Science and Its Applications (ICCSA 2012)* [Silvestre e Soares 2012].

7.2 Trabalhos futuros

Levando em consideração o escopo de sistemas de tempo real, o domínio de aplicação da área de transportes e o escopo do *profile* MARTE, foi necessário focar a pesquisa em algumas partes e não detalhar outras partes. O foco dado foi a utilização do *profile* MARTE para modelagem de sistemas de controle de sinais trânsito. Dentre outras pesquisas correlatas derivadas desta, pode-se citar:

- Realizar a modelagem de outros estudos de caso em outros domínios de aplicação para tentar comprovar as conclusões alcançadas neste trabalho. Como estudos de caso futuros, pode-se citar a modelagem de software de áreas médica, financeira e de manufatura;
- Como descrito no Capítulo 4, o MARTE utiliza as linguagens formais *CVSL* e *CCSL* para representação de tempo. Como trabalho futuro, pode-se aplicar essas linguagens formais para representar tempo no sistema de semáforos de trânsito ou em outros estudos de caso, além do desenvolvimento de um compilador para verificação da sintaxe e semântica de modelos especificados nestas linguagens;
- Outro trabalho futuro derivado diretamente deste é realizar a integração do MARTE com a SysML. A SysML tem algumas características não presentes no MARTE para modelagem de sistemas, como exemplo, o diagrama de requisitos e melhor representação do hardware. Essas características precisam ser trabalhadas juntas em um estudo de caso. O trabalho de [Espinoza et al. 2009] tenta realizar esta combinação de maneira conceitual, mas não prática;
- Pode-se realizar a implementação de *plugins* que aceitem as extensões do MARTE para a modelagem de software. Para isto, pode-se implementar *plugins* para ferramentas de modelagem conhecidas, como ArgoUML e Eclipse.
- Realizar a integração deste estudo de caso e de outros estudos de caso com a prática do desenvolvimento de software de tempo real é um trabalho futuro relevante. Pode-se fornecer a modelagem utilizando o MARTE aos desenvolvedores de software e avaliar a qualidade dos diagramas segundo um ponto de vista essencialmente prático.

Referências Bibliográficas

[EUT 2001] (2001). White Paper, European Transportation Policy for 2010: Time to Decide. Technical report, Office for Official Publications of the European Communities, Luxembourg.

[Abrial 2006] Abrial, J.-R. (2006). Formal Methods in Industry: Achievements, Problems, Future. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pp. 761–768.

[Abrial 2007] Abrial, J.-R. (2007). Formal Methods : Theory Becoming Practice. volume 41, pp. 619–628.

[Addouche et al. 2004] Addouche, N., Antoine, C., e Montmain, J. (2004). UML Models for Dependability Analysis of Real-Time Systems. In *Proceedings of the IEEE Systems, Man and Cybernetics (SMC)*, pp. 5209–5214.

[Adler 1988] Adler, M. (1988). An Algebra for Data Flow Diagram Process Decomposition. *IEEE Transactions on Software Engineering*, 14:169–183.

[Alabiso 1988] Alabiso, B. (1988). Transformation of Data Flow Analysis Models to Object Oriented Design. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications*, OOPSLA '88.

[Andrade 2008] Andrade, C. (2008). *Controle de Sistemas Max-Plus Lineares Sujeitos a Restricões no Estado*. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica - UFMG.

[André e Mallet 2008] André, C. e Mallet, F. (2008). Clock Constraints in UML/MARTE CCSL. Research Report RR-6540, INRIA.

[André e Mallet 2009] André, C. e Mallet, F. (2009). Specification and Verification of Time Requirements with CCSL and Esterel. In *ACM SIGPLAN Notices*, volume 44, pp. 167–176.

[André et al. 2007] André, C., Mallet, F., e de Simone, R. (2007). Modeling Time(s). In *Proceedings of the 10th International Conference on Model Driven Engineering Languages and Systems (MODELS '07)*, pp. 559–573. Springer Verlag.

[Arnold et al. 2003] Arnold, G., Henriques, P., e Fonseca, J. (2003). A Development Approach to Industrial Robots Programming. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR' 2003)*, pp. 167–172.

[Arrayangkool e Unakul 2009] Arrayangkool, A. e Unakul, A. (2009). A Flexible Intelligent Transportation System Architecture Model with Object Oriented Methodology and UML. In *Proceedings of the 9th International Conference on Communications and Information Technologies*, ISCIT'09, pp. 741–746.

[Axelsson 2000] Axelsson, J. (2000). Real-World Modeling in UML. In *Proceedings of the 13th International Conference on Software and Systems Engineering and their Applications, Paris*, pp. 196–206.

[Barbosa 2009] Barbosa, G. (2009). Um Livro-texto para o Ensino de Projeto de Arquitetura de Software. Master's thesis, Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.

[Bass et al. 2003] Bass, L., Clements, P., e Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition.

[Bennett e Field 2004] Bennett, A. e Field, A. (2004). Performance Engineering with the UML Profile for Schedulability, Performance and Time: a Case Study. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2004)*, pp. 67 – 75.

[Berkenkotter 2003] Berkenkotter, K. (2003). Using UML 2.0 in Real-Time Development a Critical Review. In *International Workshop on SVERTS: Specification and Validation of UML Models for Real Time and Embedded Systems*.

[Bezerra 2007] Bezerra, B. (2007). *Semáforos: Gestão Técnica, Percepção do Desempenho, Duração dos Tempos*. PhD thesis, Escola de Engenharia de São Carlos - USP.

[Bianco et al. 2002] Bianco, V. D., Lavazza, L., e Mauri, M. (2002). A Formalization of UML Statecharts for Real-Time Software Modeling. In *Integrated Design and Process Technology, IDPT-2002*.

[Bietz et al. 1993] Bietz, A., Berry, A., Lister, A., e Raymond, K. (1993). Introduction to Open Distributed Processing. In *Proceedings of the ACS Queensland Branch Conference–Overcoming Isolation: The Human–Computer Connection*, pp. 57–68.

[Bihari e Gopinath 1992] Bihari, T. E. e Gopinath, P. (1992). Object-Oriented Real-Time Systems: Concepts and Examples. *Computer*, 25:25–32.

[Bisbal et al. 1999] Bisbal, J., Lawless, D., Wu, B., e Grimson, J. (1999). Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues. Technical report, Video, Recommendation ITU-T H.262, ISO/IEC.

[Björklund e Lilius 2002] Björklund, D. e Lilius, J. (2002). From UML Behavioral Descriptions to Efficient Synthesizable VHDL. In *Proceedings of the 20th IEEE NORCHIP Conference*, pp. 45–50.

[Boehm e Turner 2003] Boehm e Turner, R. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Bonnefoi et al. 2007] Bonnefoi, F., Hillah, L., Kordon, F., e Renault, X. (2007). Design, Modeling and Analysis of ITS Using UML and Petri Nets. In *Intelligent Transportation Systems Conference (ITSC' 2007)*, pp. 314–319.

[Booch 1994] Booch, G. (1994). *Object-Oriented Analysis and Design with Applications (2nd ed.)*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

[Booch et al. 2007] Booch, G., Maksimchuk, R., Engle, M., Young, B., Conallen, J., e Houston, K. (2007). *Object-Oriented Analysis and Design With Applications*. Addison-Wesley Professional, New York, NY, USA, 3rd edition.

[Booch et al. 2005] Booch, G., Rumbaugh, J., e Jacobson, I. (2005). *Unified Modeling Language User Guide*. Addison-Wesley Professional, Redwood City, CA, USA, 2nd edition.

[Boutekkouk et al. 2009] Boutekkouk, F., Benmohammed, M., Bilavarn, S., e Auguin, M. (2009). UML 2.0 Profiles for Embedded Systems and Systems On a Chip (SOCs). *JOT (Journal of Object Technology)*, 8(1):710–715.

[Bowen e Hinchey 2006] Bowen, J. e Hinchey, M. (2006). Ten Commandments of Formal Methods...Ten Years Later. *Computer*, 39(1):40 – 48.

[Brave 1993] Brave, Y. (1993). Control of Discrete Event Systems Modeled as Hierarchical State Machines. volume 38, pp. 1803–1819.

[Bray 2002] Bray, I. K. (2002). *An Introduction to Requirements Engineering*. Addison Wesley, Boston, MA, USA.

[Brown e McDermid 2007] Brown, A. e McDermid, J. (2007). The Art and Science of Software Architecture. *Software Architecture*, pp. 237–256.

[Broy 2006] Broy, M. (2006). The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems. *Computer*, 39:72–80.

[Bruda e Akl 2001] Bruda, S. D. e Akl, S. G. (2001). Real-Time Computation: A Formal Definition and Its Applications. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IPDPS '01, pp. 137–. IEEE Computer Society.

[Bucci et al. 1995] Bucci, G., Campanai, M., e Nesi, P. (1995). Tools for Specifying Real-Time Systems. *Real-Time Systems*, 8(2-3):117–172.

[Burback 1998] Burback, R. (1998). A Distributed Architecture Definition Language: a DADL. `http://infolab.stanford.edu/~burback/dadl/control.html`. Acessado em 25 de novembro de 2011.

[Burmester e Giese 2003] Burmester, S. e Giese, H. (2003). The Fujaba Real-Time Statechart PlugIn. In Giese, H. e Zündorf, A. (editores), *Proceedings of the First International Fujaba Days 2003*, Technical Report. University of Paderborn.

[Burns e Wellings 2001] Burns, A. e Wellings, A. J. (2001). *Real-Time Systems and Programming Languages: ADA 95, Real-Time Java, and Real-Time POSIX*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

[Carro e Wagner 2003] Carro, L. e Wagner, F. (2003). Sistemas Computacionais Embarcados. In *Jornadas de Atualização em Informática, Campinas*.

[Casciato e Cass 1962] Casciato, L. e Cass, S. (1962). Pilot Study of the Automatic Control of Traffic Signals by a General Purpose Electronic Computer. *Highway Research Board Bulletin*.

[Cassandras e Lafortune 2010] Cassandras, C. G. e Lafortune, S. (2010). *Introduction to Discrete Event Systems*. Springer-Verlag, New York, Incorporated, 2nd edition.

[Cheng e Krishnakumar 1993] Cheng, K. T. e Krishnakumar, A. S. (1993). Automatic Functional Test Generation Using the Extended Finite State Machine Model. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, pp. 86–91.

[Clarke et al. 1986] Clarke, E. M., Emerson, E. A., e Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263.

[Clarke e Wing 1996] Clarke, E. M. e Wing, J. M. (1996). Formal Methods: State of the Art and Future Directions. *ACM Computing Survey*, 28:626–643.

[Coulouris et al. 2011] Coulouris, G., Dollimore, J., Kindberg, T., e Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition.

[Datta e Dutta 1990] Datta, T. e Dutta, U. (1990). Traffic Signal Installation and Accident Experience. volume 60, pp. 39–42.

[Davol 2001] Davol, A. (2001). *Modeling of Traffic Signal Control and Transit Signal Priority Strategies in a Microscopic Simulation Laboratory*. PhD thesis, Massachusetts Institute of Technology.

[DeMarco 1979] DeMarco, T. (1979). *Structured Analysis and System Specification*, pp. 409–424. Yourdon Press, Upper Saddle River, NJ, USA.

[Demathieu et al. 2008] Demathieu, S., Thomas, F., André, C., Gérard, S., e Terrier, F. (2008). First Experiments Using the UML Profile for MARTE. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, pp. 50–57. IEEE Computer Society.

[Deng et al. 2005] Deng, L. Y., Liang, H. C., Wang, C.-T., Wang, C.-S., e Hung, L.-P. (2005). The Development of the Adaptive Traffic Signal Control System. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops - Volume 02*, ICPADS '05, pp. 634–638. IEEE Computer Society.

[Denning 1997] Denning, P. (1997). A New Social Contract for Research. *Communications of the ACM*, 40(2):132–134.

[Dotoli et al. 2008] Dotoli, M., Fanti, M. P., e Mangini, A. M. (2008). Real Time Identification of Discrete Event Systems using Petri nets. *Automatica*, 44(5):1209 – 1219.

[Douglass 2004] Douglass, B. P. (2004). *Real Time UML: Advances in the UML for Real-Time Systems*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 3rd edition.

[Du et al. 2007] Du, Y. Y., Jiang, C. J., e Zhou, M. C. (2007). Modeling and Analysis of Real-Time Cooperative Systems Using Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 37(5):643 –654.

[Easterbrook et al. 2008] Easterbrook, S., Singer, J., Storey, M., e Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. *Guide to Advanced Empirical Software Engineering*, pp. 285–311.

[Espinoza et al. 2009] Espinoza, H., Cancila, D., Selic, B., e Gérard, S. (2009). Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ECMDA-FA '09, pp. 98–113. Springer-Verlag.

[Fantechi e Ferrari 2009] Fantechi, A. e Ferrari, A. (2009). Panel Discussion on Formal Methods in Commercial Software Development Tools. *Formal Methods for Industrial Critical Systems*, pp. 4–6.

[FHWA 1995] FHWA (1995). Improving Traffic Signal Operations, Federal Highway Administration. Technical report, Department of Transportation, Washington, DC, USA.

[FHWA 1996] FHWA (1996). Traffic Control Systems Handbook, Federal Highway Administration. Technical report, Department of Transportation, Washington, DC, USA.

[Fidge e Lister 1992] Fidge, C. e Lister, A. (1992). Disciplined Approach to Real-Time Systems Design. *Information and Software Technology*, 34(9):603 – 610.

[Figueiredo 2005] Figueiredo, L. (2005). *Sistemas Inteligentes de Transporte*. PhD thesis, Universidade do Porto.

[Flyvbjerg 2006] Flyvbjerg, B. (2006). Five Misunderstandings About Case-Study Research. *Qualitative inquiry*, 12(2):219.

[Forward e Lethbridge 2002] Forward, A. e Lethbridge, T. (2002). The Relevance of Software Documentation, Tools and Technologies: a Survey. In *Proceedings of the 2002 ACM Symposium on Document Engineering*, pp. 26–33.

[Fowler 2003] Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

[Gane e Sarson 1977] Gane, C. e Sarson, T. (1977). *Structured Systems Analysis: Tools and Techniques*. McDonnell Douglas Systems Integration Company, Upper Saddle River, NJ, USA.

[Garlan et al. 2010] Garlan, D., Bachmann, F., Ivers, J., Stafford, J., Bass, L., Clements, P., e Merson, P. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, Boston, MA, USA, 2nd edition.

[Garlan e Shaw 1994] Garlan, D. e Shaw, M. (1994). An Introduction to Software Architecture. Technical report, Pittsburgh, PA, USA.

[Gérard 2011] Gérard, S. (2011). Papyrus UML. `http://www.papyrusuml.org/`. Acessado em 16 de junho de 2011.

[Gérard e Terrier 2003] Gérard, S. e Terrier, F. (2003). *UML for Real-Time: Which Native Concepts to Use?*, pp. 17–51. Kluwer Academic Publishers, Norwell, MA, USA.

[Ghosh e Lee 2000] Ghosh, S. e Lee, T. (2000). *Intelligent Transportation Systems: New Principles and Architectures*. CRC Press, New York, NY, USA.

[Giese e Burmester 2003] Giese, H. e Burmester, S. (2003). Real-Time Statechart Semantics. Technical Report tr-ri-03-239, Lehrstuhl für Softwaretechnik, Universität Paderborn, Paderborn, Germany.

[Gogolla 2004] Gogolla, M. (2004). Benefits and Problems of Formal Methods. In Llamosí, A. e Strohmeier, A. (editores), *Reliable Software Technologies - Ada-Europe 2004*, Lecture Notes in Computer Science, pp. 1–15. Springer Berlin.

[Goguen 1997] Goguen, J. A. (1997). Formal Methods: Promises and Problems. *IEEE Software*, 14:73–85.

[Graf et al. 2006] Graf, S., Ober, I., e Ober, I. (2006). A Real-Time Profile for UML. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(2):113–127.

[Hall 1990] Hall, A. (1990). Seven Myths of Formal Methods. *Software, IEEE*, 7(5):11–19.

[Hall 1999] Hall, R. (1999). *Handbook of Transportation Science*, volume 23. Kluwer Academic Publishers.

[Harel 1987] Harel, D. (1987). Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274.

[Hatley e Pirbhai 1987] Hatley, D. J. e Pirbhai, I. A. (1987). *Strategies for Real-Time System Specification*. Dorset House Publishing Co., Inc., New York, NY, USA.

[Henderson-Sellers 2005] Henderson-Sellers, B. (2005). UML - the Good, the Bad or the Ugly? Perspectives from a panel of experts. *Software and Systems Modeling*, 4(1):4–13.

[Henzinger e Sifakis 2007] Henzinger, T. A. e Sifakis, J. (2007). The Discipline of Embedded Systems Design. *Computer*, 40:32–40.

[Hevner et al. 2004] Hevner, A. R., March, S. T., Park, J., e Ram, S. (2004). Design Science in Information Systems Research. *MIS Quaterly*, 28:75–105.

[Hilliard 2000] Hilliard, R. (2000). IEEE-Std-1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems.

[Hölzl et al. 2008] Hölzl, M., Rauschmayer, A., e Wirsing, M. (2008). Software-Intensive Systems and New Computing Paradigms. chapter Engineering of Software-Intensive Systems: State of the Art and Research Challenges, pp. 1–44. Springer-Verlag.

[Hopcroft et al. 1979] Hopcroft, J., Motwani, R., e Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*, volume 3. Addison-Wesley Reading, MA.

[Huang 2006] Huang, Y.-S. (2006). Design of Traffic Light Control Systems Using Statecharts. *The Computer Journal*, 49(6):634–649.

[Huszerl et al. 2000] Huszerl, G., Kosmidis, K., Cin, M. D., Majzik, I., e Pataricza, A. (2000). Quantitative Analysis of UML Statechart Models of Dependable Systems. *The Computer Journal, Vol*, 45:260–277.

[ITSAmerica 2007] ITSAmerica (2007). `http://www.itsa.org/`. Acessado em 17 de novembro de 2011.

[Jacobson 1992] Jacobson, I. (1992). *Object-Oriented Software Engineering*. ACM, New York, NY, USA.

[Jacobson 2004] Jacobson, I. (2004). Use cases - Yesterday, Today, and Tomorrow. *Software and System Modeling.*, 3(3):210–220.

[Jaffe et al. 1991] Jaffe, M., Leveson, N., Heimdahl, M., e Melhart, B. (1991). Software Requirements Analysis for Real-Time Process-Control Systems. *IEEE Transactions on Software Engineering*, 17(3):241–258.

[Jin et al. 2011] Jin, W., Wang, H., e Zhu, M. (2011). Modeling MARTE Sequence Diagram with Timing Pi-Calculus. In *ISORC*, pp. 61–66.

[Joseph 1992] Joseph, M. (1992). Problems, Promises and Performance: Some Questions for Real-time System Specification. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pp. 315–324. Springer-Verlag.

[Junior 2007] Junior, A. D. A. (2007). *Método de Coordenação Semafórica para Corredores de Transporte Coletivo*. PhD thesis, PET Programa de Engenharia de Transporte - COPPE/UFRJ.

[Kelly 1987] Kelly, J. C. (1987). A Comparison of Four Design Methods for Real-Time Systems. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pp. 238–252. IEEE Computer Society Press.

[Kemmerer e Ghezzi 1992] Kemmerer, R. A. e Ghezzi, C. (1992). Guest Editors' Introduction: Specification and Analysis of Real-Time Systems. *IEEE Transactions on Software Engineering*, 18(9):766–767.

[Khalsa 1989] Khalsa, G. K. (1989). Using Object Modeling to Transform Structured Analysis into Object Oriented Design. In *Proceedings of the Sixth Washington Ada Symposium on Ada*, WADAS '89, pp. 201–212.

[King e Goldblatt 1975] King, G. e Goldblatt, R. (1975). Relationship of Accident Patterns to Type of Intersection Control. *Transportation Research Record*, (540):1–12.

[Kirner e Davis 1996] Kirner, T. G. e Davis, A. M. (1996). Requirements Specification of Real-Time Systems: Temporal Parameters and Timing-Constraints. *Information and Software Technology*, 38(12):735 – 741.

[Koonce et al. 2008] Koonce, P., Rodegerdts, L., Lee, K., Quayle, S., Beaird, S., Braud, C., Bonneson, J., Tarnoff, P., e Urbanik, T. (2008). Traffic Signal Timing Manual. Technical report.

[Kopetz 2011] Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Publishing Company, Incorporated, Norwell, MA, USA, 2nd edition.

[Kruchten 1995] Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Software,*, 12(6):42–50.

[Kruchten 2003] Kruchten, P. (2003). *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

[Kruchten et al. 2006] Kruchten, P., Obbink, H., e Stafford, J. (2006). The Past, Present, and Future for Software Architecture. *IEEE Software*, 23:22–30.

[Kumar e Goel 2010] Kumar, M. e Goel, S. (2010). Specifying Safety and Critical Real-Time Systems in Z. In *International Conference on Computer and Communication Technology (ICCCT)*, pp. 596 –602.

[Kutz 2004] Kutz, M. (2004). *Handbook of Transportation Engineering*. McGraw-Hill Professional.

[Laplante 2004] Laplante, P. A. (2004). *Real-Time System Design and Analysis*. John Wiley & Sons, Piscataway, NJ,USA.

[Larsen et al. 1996] Larsen, P. G., Fitzgerald, J., e Brookes, T. (1996). Applying Formal Specification in Industry. *IEEE Software*, 13:48–56.

[Lavazza et al. 2001] Lavazza, L., Quaroni, G., e Venturelli, M. (2001). Combining UML and formal notations for modelling real-time systems. In *ACM SIGSOFT Software Engineering Notes*, volume 26, pp. 196–206. ACM.

[Lee 1989] Lee, A. (1989). A Scientific Methodology for MIS Case Studies. *MIS Quarterly*, pp. 33–50.

[Lee e Yannakakis 1996] Lee, D. e Yannakakis, M. (1996). Principles and Methods of Testing Finite State Machines - a Survey. In *Proceedings of the IEEE, vol. 84*. IEEE.

[Lee 2002] Lee, D.-T. (2002). Evaluating Real-Time Software Specification Languages. *Computer Standards & Interfaces*, 24:395–409.

[Leist e Zellner 2006] Leist, S. e Zellner, G. (2006). Evaluation of Current Architecture Frameworks. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pp. 1546–1553.

[Lin et al. 2003] Lin, L., Nan, T., Xiangyang, M., e Fubing, S. (2003). Implementation of Traffic Lights Control based on Petri Nets. In *Intelligent Transportation Systems*, volume 2, pp. 1087–1090. IEEE.

[Lindgren et al. 2008] Lindgren, M., Norstrom, C., Wall, A., e Land, R. (2008). Importance of Software Architecture During Release Planning. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, WICSA '08, pp. 253–256. IEEE Computer Society.

[Liu et al. 2009] Liu, H., Jiang, Z., e Fung, R. Y. K. (2009). Performance Modeling, Real-Time Dispatching and Simulation of Wafer Fabrication Systems using Timed Extended Object-Oriented Petri Nets. *Computing Industry Engineering*, 56:121–137.

[Mallet 2008] Mallet, F. (2008). Clock Constraint Specification Language: Specifying Clock Constraints with UML/MARTE. *Innovations in Systems and Software Engineering*, 4(3):309–314.

[Mallet e de Simone 2008] Mallet, F. e de Simone, R. (2008). MARTE: a Profile for RT/E Systems Modeling, Analysis and Simulation? In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pp. 43:1–43:8. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Martins 1996] Martins, J. A. (1996). *Transporte, Uso do Solo e Auto-Sustentabilidade*. PhD thesis, PET Programa de Engenharia de Transporte - COPPE/UFRJ.

[Mathalone 1997] Mathalone, S. (1997). A Behaviorally-Based Methodology for Modeling System Specifications. *SIGSOFT Software Engineering Notes*, 22:39–42.

[Matzoros e Van Vliet 1992] Matzoros, A. e Van Vliet, D. (1992). A Model of Air Pollution from Road Traffic, Based on the Characteristics of Interrupted Flow and Junction Control: Part I–model description. *Transportation Research Part A: Policy and Practice*, 26(4):315–330.

[McUmber e Cheng 1999] McUmber, W. e Cheng, B. (1999). UML-Based Analysis of Embedded Systems Using a Mapping to VHDL. In *Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering*, pp. 56–63. IEEE.

[Mraidha et al. 2008] Mraidha, C., Tanguy, Y., Jouvray, C., Terrier, F., e Gerard, S. (2008). An Execution Framework for MARTE - Based Models. In *Proceedings 13th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 222 –227.

[Mueller 1970] Mueller, E. (1970). Aspects of the History of Traffic Signals. *IEEE Transactions on Vehicular Technology*, 19(1):6–17.

[Murata 1989] Murata, T. (1989). Petri nets: Properties, Analysis and Applications. *IEEE Journal*, 77(4):541–580.

[Neill e Laplante 2003] Neill, C. J. e Laplante, P. A. (2003). Specification of Real-Time Imaging Systems Using the UML. *Real-Time Imaging*, 9:125–137.

[Nickel et al. 2000] Nickel, U., Niere, J., e Zündorf, A. (2000). The FUJABA Environment. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pp. 742–745. ACM.

[Nielsen e Shumate 1987] Nielsen, K. W. e Shumate, K. (1987). Designing Large Real-Time Systems with Ada. *Communications of the ACM*, 30:695–715.

[OMG 2003] OMG (2003). UML Profile for Schedulability, Performance, and Time, Version 1.0. Technical Report formal/2003-09-01, OMG.

[OMG 2005a] OMG (2005a). OMG Unified Modeling Language (OMG UML) Infrastructure, Version 2.0. Technical Report formal/2005-07-05, OMG.

[OMG 2005b] OMG (2005b). UML Profile for Schedulability, Performance, and Time, Version 1.1. Technical Report formal/2005-01-02, OMG.

[OMG 2007] OMG (2007). MARTE Tutorial: UML profile for develop for Real-Time and Embedded systems. Technical Report formal/2007-03-28, OMG.

[OMG 2009] OMG (2009). UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems Version, 1.0. Technical Report formal/2009-11-02, OMG.

[OMG 2010a] OMG (2010a). OMG Unified Modeling Language (OMG UML) Infrastructure, Version 2.3. Technical Report formal/2010-05-03, OMG.

[OMG 2010b] OMG (2010b). OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.3. Technical Report formal/2010-05-03, OMG.

[OMG 2010c] OMG (2010c). Systems Modeling Language (SysML) - Version 1.2.

[OMG 2011] OMG (2011). UML Profile for MARTE: Modeling and Analysis of Real-time Embedded Systems Version, 1.1. Technical Report formal/2011-06-02, OMG.

[Oshana 2005] Oshana, R. (2005). *DSP Software Development Techniques for Embedded and Real-Time Systems (Embedded Technology)*. Newnes.

[PAIS 2004] PAIS, R. M. C. (2004). Geração de executores e analisadores de Redes de Petri. Master's thesis, Dissertação (Mestrado). Faculdade de Ciências e Tecnologia, Universidade de Lisboa, Lisboa, 2004.

[Papageorgiou et al. 2003] Papageorgiou, M., Diakaki, C., Dinopoulou, V., Kotsialos, A., e Wang, Y. (2003). Review of Road Traffic Control Strategies. *Proceedings of the IEEE*, 91(12):2043–2067.

[Peters 1987] Peters, L. (1987). *Advanced Structured Analysis and Design*. Prentice-Hall, Incorporation, Upper Saddle River, NJ, USA.

[Peters 1989] Peters, L. (1989). Timing Extensions to Structured Analysis for Real Time Systems. In *Proceedings of the 5th International Workshop on Software Specification and Design*, IWSSD '89, pp. 83–90. ACM.

[Petriu e Woodside 2004] Petriu, D. C. e Woodside, M. (2004). Extending the UML Profile for Schedulability Performance and Time (SPT) for Component-Based Systems.

[Plat et al. 1992] Plat, N., van Katwijk, J., e Toetenel, H. (1992). Application and Benefits of Formal Methods in Software Development. *Software Engineering Journal*, 7(5):335 –346.

[Pressman 2010] Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, USA, 7th edition.

[Ranjini et al. 2011] Ranjini, K., Kanthimathi, A., e Yasmine, Y. (2011). Design of Adaptive Road Traffic Control System through Unified Modeling Language. *International Journal of Computer Applications*, 14(7):36–41. Published by Foundation of Computer Science.

[Robertson et al. 1996] Robertson, D., Wilson, N., e Kemp, S. (1996). The Effects of Co-ordinated and Isolated Signal Control on Journey Times and Exhaust Emissions Along the A12 in London. *Traffic engineering & control*, 37(1):4–9.

[Roess et al. 2004] Roess, R., Prassas, E., e McShane, W. (2004). *Traffic Engineering*. Pearson/Prentice Hall, Upper Saddle River, NJ, USA.

[Rossi e Sein 2003] Rossi, M. e Sein, M. (2003). Design Research Workshop: a Proactive Research Approach. *Action Research*, 2005:1–20.

[Rouse 2003] Rouse, W. B. (2003). Engineering Complex Systems: Implications for Research in Systems Engineering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 33(2):154–156.

[Rumbaugh et al. 1991] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., e Lorensen, W. (1991). *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Runeson e Höst 2009] Runeson, P. e Höst, M. (2009). Guidelines for Conducting and Reporting Case Study Research in Software Engineering. *Empirical Software Engineering*, 14:131–164.

[Sanden 1989] Sanden, B. (1989). An Entity-Life Modeling Approach to the Design of Concurrent Software. *Communications of the ACM*, 32:330–343.

[Selic 1999] Selic, B. (1999). Turning clockwise: Using UML in the Real-Time Domain. *Communications ACM*, 42:46–54.

[Selic 2006] Selic, B. (2006). Tutorial: an Overview of UML 2. In *Proceedings of the 28th International conference on Software engineering*, ICSE '06, pp. 1069–1070. ACM.

[Shaw 1992] Shaw, A. (1992). Communicating Real-Time State Machines. *IEEE Transactions on Software Engineering*, 18(9):805 –816.

[Shaw 2000] Shaw, A. C. (2000). *Real-Time Systems and Software*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

[Shaw 2003] Shaw, M. (2003). Writing Good Software Engineering Research Papers: Minitutorial. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pp. 726–736. IEEE Computer Society.

[Silvestre e Soares 2011] Silvestre, E. e Soares, M. S. (2011). Multiple View Architecture Model for Distributed Real-Time Systems Using MARTE. In *Proceedings of the 20th International Conference on Information Systems Development*, ISD '2011, pp. 1–9.

[Silvestre e Soares 2012] Silvestre, E. e Soares, M. S. (2012). Modeling Road Traffic Signals Control using UML and the MARTE Profile. In *Proceedings of the 12th International Conference on Computational Science and Its Applications. Aceito*, ICCSA '2012. Aceito.

[Sinha et al. 2000] Sinha, V., Doucet, F., Siska, C., Gupta, R., Liao, S., e Ghosh, A. (2000). YAML: A Tool for Hardware Design Visualization and Capture. In *Proceedings of the 13th International Symposium on System Synthesis*, pp. 9–14. IEEE Computer Society.

[Sirikijpanichkul et al. 2005] Sirikijpanichkul, A., van Vuren, T., e Tenekeci, G. (2005). Application of Microsimulation Program for Traffic Management Scheme Appraisal Case Study: Perry Barr - Birmingham, UK. In *The 15th International Road Federation (IRF) World Meeting 2005*. International Road Federation – IRF.

[Sjoberg et al. 2007] Sjoberg, D., Dyba, T., e Jorgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research.

[Smith e Hayes 2000] Smith, G. e Hayes, I. J. (2000). Structuring Real-Time Object-Z Specifications. In *Proceedings of the Second International Conference on Integrated Formal Methods*, IFM '00, pp. 97–115, London, UK. Springer-Verlag.

[Soares 2010] Soares, M. d. S. (2010). *Architecture-Driven Integration of Modeling Languages for the Design of Software-Intensive Systems*. PhD thesis, Delft University of Technology.

[Soares e Vrancken 2008] Soares, M. S. e Vrancken, J. (2008). Responsive Traffic Signals Designed With Petri Nets. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC' 2008)*, pp. 1942–1947.

[Soares e Vrancken 2009] Soares, M. S. e Vrancken, J. L. M. (2009). Empirical Evaluation of UML in Practice - Experiences in a Traffic Control Company. In Cordeiro, J. e Filipe, J. (editores), *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS)*, pp. 313–319.

[Soares et al. 2011] Soares, M. S., Vrancken, J. L. M., e Verbraeck, A. (2011). User Requirements Modeling and Analysis of Software-Intensive Systems. *Journal of Systems and Software*, 84(2):328–339.

[Sommerville 2010] Sommerville, I. (2010). *Software Engineering*. Pearson Addison Wesley, Boston, MA, USA, 9th edition.

[Soni et al. 1995] Soni, D., Nord, R. L., e Hofmeister, C. (1995). Software Architecture in Industrial Applications. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95.

[Spivey 1989] Spivey, J. M. (1989). *The Z notation: a reference manual*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Srinivasan e Choy 2006] Srinivasan, D. e Choy, M. (2006). Cooperative Multi-Agent System for Coordinated Traffic Signal Control. In *IEEE Proceedings: Intelligent Transport Systems*, volume 153, pp. 41–50.

[Staines 2005] Staines, A. S. (2005). A Comparison of Software Analysis and Design Methods for Real Time Systems. In *World Academy of Science, Engineering and Technology*, pp. 55–59.

[Stankovic 1988] Stankovic, J. A. (1988). Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. *Computer*, 21:10–19.

[Stewart 1999] Stewart, D. (1999). Twenty-Five Most Common Mistakes with Real-Time Software Development. In *Tutorial Presented at the 1999 Embedded Systems Conference*.

[Stoel e Karrfalt 1995] Stoel, C. e Karrfalt, J. (1995). VIOOL for Hardware/Software Codesign. In *Systems Engineering of Computer Based Systems, 1995., Proceedings of the 1995 International Symposium and Workshop on*, pp. 333–340. IEEE.

[Süß et al. 2008] Süß, J., Fritzson, P., e Pop, A. (2008). The Impreciseness of UML and Implications for ModelicaML. In *Proceedings of the 2nd International Workshop on Equation-Based Object-Oriented Languages and Tools*.

[Sussman 2005] Sussman, J. (2005). *Perspectives on Intelligent Transportation Systems (ITS)*. Springer.

[Tanenbaum e Steen 2006] Tanenbaum, A. S. e Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition.

[Tatsuta 1995] Tatsuta, T. (1995). Practical Verification for Requirements Model of Hatley/Pirbhai. *SIGSOFT Software Engineering Notes*, 20:94–97.

[Thramboulidis 2004] Thramboulidis, K. (2004). Using UML in Control and Automation: a Model Driven Approach. In *2nd IEEE International Conference on Industrial Informatics (INDIN '04)*, pp. 587 –593.

[Tian 2002] Tian, Z. (2002). *Capacity Analysis of Traffic-Actuated Intersections*. PhD thesis, Massachusetts Institute of Technology.

[Tichy e Kudak 2003] Tichy, M. e Kudak, M. (2003). Visualization of the Execution of Real-Time Statecharts. In *Proceedings of the First International Fujaba Days 2003*.

[Tolba et al. 2003] Tolba, C., Thomas, P., ElMoudni, A., e Lefebvre, D. (2003). Performances Evaluation of the Traffic Control in a Single Crossroad by Petri Nets. In *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, volume 2, pp. 157–160. IEEE.

[Toma et al. 2007] Toma, M., Busam, A., Ortmaier, T., Raczkowsky, J., Hopner, C., e Marmulla, R. (2007). UML Based Modeling of Medical Applications Workflow in Maxillofacial Surgery. *GMS CURAC*.

[Tsichritzis 1997] Tsichritzis, D. (1997). Beyond Calculation. chapter The Dynamics of Innovation. Copernicus, New York, NY, USA.

[Van Aken 2005] Van Aken, J. (2005). Management Research as a Design Science: Articulating the Research Products of Mode 2 Knowledge Production in Management. *British Journal of Management*, 16(1):19–36.

[Varoto 2002] Varoto, A. (2002). Visões em Arquitetura de Software. Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo.

[Vepsalainen et al. 2008] Vepsalainen, T., Hastbacka, D., e Kuikka, S. (2008). Tool Support for the UML Automation Profile - For Domain-Specific Software Development in Manufacturing. In *3rd International Conference on Software Engineering Advances (ICSEA '08)*, pp. 43 –50.

[Von Alan et al. 2004] Von Alan, R., March, S., Park, J., e Ram, S. (2004). Design science in information systems research. *Mis Quarterly*, 28(1):75–105.

[Vrancken et al. 2008] Vrancken, J. L. M., van den Berg, J., e dos Santos Soares, M. (2008). Human Factors in System Reliability: Lessons Learnt from the Maeslant Storm Surge Barrier in The Netherlands. *IJCIS*, 4(4):418–429.

[Ward e Mellor 1986] Ward, P. T. e Mellor, S. J. (1986). *Structured Development for Real-Time Systems*. Yourdon Press, Upper Saddle River, NJ, USA.

[Wastell et al. 2009] Wastell, D., Sauer, J., e Schmeink, C. (2009). Time for a Design Turn in IS Innovation Research? A Practice Report from the Home Front. *Information Technology & People*, 22(4):335–350.

[Whittle e Schumann 2002] Whittle, J. e Schumann, J. (2002). Statechart Synthesis from Scenarios: an Air Traffic Control Case Study. In *Proceedings of the Workshop at the 24th International Conference on Software Engineering (ICSE 2002)*.

[Wiering et al. 2004] Wiering, M., van Veenen, J., Vreeken, J., e Koopman, A. (2004). Intelligent Traffic Light Control. Technical report, Ericm News, European Research Consortium form Informatics and Mathematics.

[Williams 2006] Williams, R. (2006). *Real-Time Systems Development*. Butterworth-Heinemann, Newton, MA, USA.

[Wing 1990] Wing, J. (1990). A Specifier's Introduction to Formal Methods. *Computer*, 23(9):8, 10 –22, 24.

[Woodcock e Davies 1996] Woodcock, J. e Davies, J. (1996). *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Woodcock et al. 2009] Woodcock, J., Larsen, P. G., Bicarregui, J., e Fitzgerald, J. (2009). Formal Methods: Practice and Experience. *ACM Computing Surveys*, 41:19:1–19:36.

[Xu et al. 2003] Xu, J., Woodside, M., e Petriu, D. (2003). Performance Analysis of a Software Design Using the UML Profile for Schedulability, Performance and Time. In *In Proceedings 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS 03)*, pp. 291–310. Springer-Verlag.

[Yin 2008] Yin, R. K. (2008). *Case Study Research: Design and Methods*. Sage Publications, Incorporated, CA, 4th edition.

[Yongfeng et al. 2009] Yongfeng, Y., Bin, L., Deming, Z., e Tongmin, J. (2009). On modeling Approach for Embedded Real-Time Software Simulation Testing. *Journal of Systems Engineering and Electronics*, 20(2):420–426.

[Yourdon 1989] Yourdon, E. (1989). *Modern Structured Analysis*. Yourdon Press, Upper Saddle River, NJ, USA.

[Yourdon e Constantine 1979] Yourdon, E. e Constantine, L. L. (1979). *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1st edition.

[Zachmann 1997] Zachmann, J. A. (1997). Enterprise Architecture: The Issue Of The Century. *Database Programming and Design*, pp. 1–13.

[Zeng et al. 2007] Zeng, R., Li, G., e Lin, L. (2007). Adaptive Traffic Signals Control by Using Fuzzy Logic. In *Innovative Computing, Information and Control, 2007. ICICIC'07. Second International Conference on*, pp. 527–527. IEEE.

[Zhou et al. 2004] Zhou, Y., Chen, Y., e Lu, H. (2004). UML-Based Systems Integration Modeling Technique for the Design and Development of Intelligent Transportation Management System. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, volume 7, pp. 6061 – 6066.

[Zoughbi et al. 2007] Zoughbi, G., Briand, L., e Labiche, Y. (2007). A UML Profile for Developing Airworthiness-Compliant (RTCA DO-178B), Safety-Critical Software. *Model Driven Engineering Languages and Systems*, pp. 574–588.