Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departmento de Informática e Matemática Aplicada
Programa de Pós-Graduação em Sistemas e Computação
Doutorado Acadêmico em Sistemas e Computação

Tese de Doutorado

Uma Linguagem de Descrição Arquitetural baseada em uma Arquitetura de Referência para Sistemas Ubíquos

Carlos Alberto Nunes Machado

Natal-RN
Janeiro / 2015

Carlos Alberto Nunes Machado

Uma Linguagem de Descrição Arquitetural baseada em uma Arquitetura de Referência para Sistemas Ubíquos

Tese submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Doutor em Ciências da Computação (DSc.). *Linha de pesquisa*: Engenharia de Software

Orientadora

Prof[a]. Dr[a]. Thaís Vasconcelos Batista

PPgSC – Programa de Pós-Graduação em Sistemas e Computação
DIMAp – Departamento de Informática e Matemática Aplicada
CCET – Centro de Ciências Exatas e da Terra
UFRN – Universidade Federal do Rio Grande do Norte

Natal-RN

Janeiro / 2015

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial
Centro de Ciências Exatas e da Terra – CCET.

Machado, Carlos Alberto Nunes.
Uma linguagem de descrição arquitetural baseada em uma arquitetura de referência para sistemas ubíquos / Carlos Alberto Nunes Machado. - Natal, 2015.
164 f. : il.

Orientadora: Profa. Dra. Thaís Vasconcelos Batista.

Tese (Doutorado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Ciência da Computação.

1. Engenharia de software – Tese. 2. Computação ubíqua – Tese. 3. Revisão sistemática – Tese. 4. Arquitetura de referência – Tese. 5.Linguagem de descrição de arquitetura – Tese. 6. Experimento controlado – Tese. I. Batista, Thaís Vasconcelos. II. Título.

RN/UF/BSE-CCET CDU: 004.41

CARLOS ALBERTO NUNES MACHADO

Uma Linguagem de Descrição Arquitetural Baseada em uma Arquitetura de Referência para Sistemas Ubíquos

Esta Tese foi julgada adequada para a obtenção do título de doutor em Ciência da Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

Prof.ª Dr.ª Thais Vasconcelos Batista – UFRN
Orientador

Prof. Dr. Uirá Kulesza – UFRN
Coordenador do Programa

Banca Examinadora

Prof.ª Dr.ª Thais Vasconcelos Batista – UFRN
Presidente

Prof. Dr. Jair Cavalcanti Leite – UFRN

Prof. Dr. Uirá Kulesza – UFRN

Prof.ª Dr.ª Elisa Yumi Nakagawa – USP

Prof. Dr. Gustavo Henrique Matos Bezerra Motta – UFPB

Janeiro, 2015

Homenagem que o autor presta a uma ou mais pessoas.

Agradecimentos

Agradeço a DEUS todos os dias por ter me concedido força, saúde, autoestima e principalmente fé para que pudesse galgar tamanha vitória em minha vida acadêmica e pessoal, pois sem Sua ajuda não teria conseguido.

Agradeço à minha esposa, Bernadete de Lima Tabosa Machado, por ter me concedido força, incentivo, conselho e recomendações, além de manter a casa sob controle durante a minha ausência e me apoiado durante todos os dias de nossa convivência familiar, em meu trabalho e em todos os meus estudos: graduação, especialização, mestrado e finalmente o doutorado.

Agradeço também às minhas três filhas, Daniele Maria Tabosa Machado, Natalia Tabosa Machado e Karla Cristina Tabosa Machado, por terem participado e me acompanhado em todos os momentos que passei fora de casa para concluir esta caminhada. Agradeço a Deus também por ter me dado uma linda e saudável neta chamada Maria Gabriela.

Agradeço aos meus pais, Manuel Nunes Machado e Josefa Ramos Machado (in memoriam), por todo amor e ensinamentos a mim dispensados, pois foram eles que plantaram em mim a semente da dedicação ao estudo e da importância da educação.

Agradeço também a todos os meus irmãos e irmãs, em especial a Marinete Nunes Machado e Ernani Rodrigues de Carvalho Filho, meu cunhado e irmão, por terem me ajudado na graduação, acompanhado e torcido por esta vitória.

Agradeço à minha orientadora, Thaís Vasconcelos Batista, por ter acreditado em mim, por sua paciência e confiança e por ter sempre me mostrado o rumo certo a seguir durante as pesquisas, pois grande parte desta conquista é obra dela.

Agradeço ao Jair Cavalcante Leite por ter sido meu primeiro orientador, pelas suas

participações e contribuições nos artigos e também por ter aceitado participar de minhas bancas de qualificação e defesa, além de ter acompanhado todo os passos deste trabalho.

Agradeço ao amigo, Eduardo Alexandre Ferreira Silva, por ter participado dessa pesquisa auxiliando em todos os momentos, principalmente na escrita dos artigos, pois suas contribuições foram muito importantes para a conquista e conclusão desta tese.

O meu agradecimento também à Elisa Yumi Nakagawa por ter aceitado participar das minhas bancas de qualificação e defesa, além de seus conselhos durante a qualificação, os quais foram muito importantes, bem como sua participação e contribuição nos artigos.

Agradeço ao Uirá Kulesza, por também ter aceitado participar da minhas bancas de qualificação e defesa e, em especial, pelas suas contribuições na qualificação.

Agradeço também ao Gustavo Henrique Matos Bezerra Motta por ter sido meu orientador no mestrado, pela sua paciência e dedicação e também por ter aceitado participar da banca de defesa desta tese, bem como por ser um amigo no Departamento de Informática que sempre pude contar.

Agradeço também aos amigos José Antônio Gomes da Silva e Hamilton Soares da Silva por terem ajudado em todos os momentos em que precisei de assistência nos processos de afastamento para cursar o doutorado, e, principalmente, ao Amilton por ter me substituído nas disciplinas, possibilitando, assim, meu afastamento do Departamento de informática na UFPB.

Agradeço aos amigos Creão e Rogéria, por terem contribuído no afastamento do CCEN e estarem sempre a minha disposição nos momentos em que precisei de ajuda.

Gostaria de agradecer aos amigos e colegas Everton Cavalcante, Ana Luisa Medeiros e Thiago Pereira por contribuírem no desenvolvimento de partes desse trabalho. Também agradeço aos demais amigos que fazem ou fizeram parte do DIMAp.

Agradeço ao Macilon, Evandro, Liliane, Marília, Felipão, Isaac e Plácido pelo excelente convívio no laboratório de doutorado, ou melhor no subsolo do DIMAp. Além de Evandro, Macilon, Ronildo, Ícaro e Petrusca, com os quais não dividi apenas o laboratório, mas também o mesmo teto por um período de tempo.

Agradeço também aos funcionários do DIMAp, Hélida, Vamberto, Sr. Gaspar e Rita, por me servirem bem e serem grandes amigos durante toda a temporada do doutorado.

Este trabalho é dedicado à Bernadete, minha esposa e a Daniele, Natalia e Karla, minhas abençoadas filhas.

Uma Linguagem de Descrição Arquitetural baseada em uma Arquitetura de Referência para Sistemas Ubíquos

Autor: Carlos Alberto Nunes Machado

Orientadora: Prof[a]. Dr[a]. Thaís Vasconcelos Batista

Resumo

A computação ubíqua é um paradigma no qual dispositivos com capacidade de processamento e comunicação são embutidos nos elementos comuns de nossas vidas (casas, carros, máquinas fotográficas, telefones, escolas, museus, etc), provendo serviços com um alto grau de mobilidade e transparência. O desenvolvimento de sistemas ubíquos é uma tarefa complexa, uma vez que envolve várias áreas da computação, como Engenharia de Software, Inteligência Artificial e Sistemas Distribuídos. Essa tarefa se torna ainda mais complexa pela ausência de uma arquitetura de referência para guiar o desenvolvimento de tais sistemas. Arquiteturas de referência têm sido usadas para fornecer uma base comum e dar diretrizes para a construção de arquiteturas de softwares para diferentes classes de sistemas. Por outro lado, as linguagens de descrição arquitetural (ADLs) fornecem uma sintaxe para representação estrutural dos elementos arquiteturais, suas restrições e interações, permitindo expressar modelos arquiteturais de sistemas. Atualmente não há, na literatura, ADLs baseadas em arquiteturas de referência para o domínio de computação ubíqua. Neste sentido, permitir a modelagem arquitetural de aplicações ubíquas, esse trabalho tem como objetivo principal especificar UbiACME, uma linguagem de descrição arquitetural para aplicações ubíquas, bem como disponibilizar a ferramenta *UbiACME Studio*, que permitirá arquitetos de software realizar modelagens usando UbiACME. Para esse fim, inicialmente realizamos uma revisão sistemática, de forma a investigar na literatura relacionada com sistemas ubíquos, os elementos comuns a esses sistemas que devem ser considerados no projeto de UbiACME. Além disso, com base na revisão sistemática, definimos uma arquitetura de referência para sistemas ubíquos, RA-Ubi, que é a base para a definição dos elementos necessários para a modelagem arquitetural e, portanto, fornece subsídios para a definição dos elementos de UbiACME. Por fim, de forma a validar a

linguagem e a ferramenta, apresentamos um experimento controlado em que arquitetos modelam uma aplicação ubíqua usando *UbiACME Studio* e comparam com a modelagem da mesma aplicação em SySML.

Palavras-chave: Computação Ubíqua, Revisão Sistemática, Arquitetura de Referência, Linguagem de Descrição de Arquitetura, Experimento Controlado.

An Architectural Description Language based on a Reference Architecture for Ubiquitous Systems

Author: Carlos Alberto Nunes Machado

Supervisor: Prof[a]. Dr[a]. Thaís Vasconcelos Batista

Abstract

Ubiquitous computing is a paradigm in which devices with processing and communication capabilities are embedded in the common elements of our daily lives, providing services with a high degree of mobility and transparency. The development of ubiquitous systems is a complex task, since it involves several areas of computing, such as Software Engineering, Artificial Intelligence, and Distributed Systems. This task becomes even more complex by the absence of a reference architecture for guiding the development of such systems. Reference architectures have been used to provide a common basis and to provide guidelines for the development of software architectures for different classes of systems. On the other hand, architectural description languages (ADLs) provide a syntax for the structural representation of the architectural elements, their constraints and interactions, allowing the specification of architectural models. As far as we are concerned, there is no ADL based on a reference architecture for the ubiquitous computing domain. In order to allow the architectural modeling of ubiquitous applications, this work aims to provide UbiACME, an architectural description language for ubiquitous applications, as well as to provide the UbiACME Studio tool, to allow software architects modeling UbiACME specifications. For this purpose, we conducted a systematic review in order to investigate the literature related to ubiquitous systems, as well as the elements common to these systems that will be considered as a basis to the UbiACME project. In addition, based on the systematic review, we defined a reference architecture for ubiquitous systems, RA-Ubi, that is a basis for the definition of the elements needed for the architectural modeling. RA-Ubi supports the definition of the UbiACME elements. Thus, in order to validate the language and the tool, we conducted a controlled experiment where software architects modeled an ubiquitous application using UbiACME Studio and also with SysML.

Keywords : Ubiquitous Computing, Ubiquitous System, Architecture Reference, Systematic review, Ubiquitous Middleware.

Lista de Figuras

Lista de Tabelas

Lista de abreviaturas e siglas

UbiCom – Computação Ubíqua

SOA – Service Oriented Architectures

PCA – Pervasive Computing Architecture

PSC-RM – Pervasive Service Composition – Reference Model

ProSA-RA – Processo que visa a Construção, Representação e Avaliação de Arquiteturas de Referência

CoBrA – *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*

EMF – Eclipse Modeling Framework

DSL – Domain-Specific Language

Sumário

Capítulo 1

Introdução

O termo computação ubíqua (UbiCom), proposto por *Weiser* (WEISER, 1991), refere-se a sistemas nos quais as informações e recursos computacionais estão disponíveis em qualquer local, a qualquer momento e em qualquer dispositivo. A Figura 1.1 mostra como a história da computação pode ser dividida em três grandes fases que são caracterizadas pela forma como os usuários interagiam com os computadores. Na primeira fase, a era dos *mainframes*, grandes computadores serviam a vários usuários de uma organização. Na segunda fase, a era dos microcomputadores, caracterizou-se pela utilização de um computador por usuário e possibilitou a disseminação de aplicativos comerciais e de escritórios. A terceira fase, computação ubíqua, em que há uma variedade enorme de dispositivos eletrônicos com poder computacional, que fornecem conjuntamente serviços para o usuário (WEISER, 1991).

A idéia básica da computação ubíqua é que a computação move-se para fora das estações de trabalho e computadores pessoais (PCs) e torna-se pervasiva em nossa vida cotidiana. Por ser uma área emergente, termos como *computação ubíqua, computação pervasiva, computação nomádica, computação móvel* e outros tantos, têm sido usados, muitas vezes, como sinônimos, embora sejam diferentes conceitualmente e empreguem diferentes idéias de organização e gerenciamento dos serviços computacionais. Na medida em que a área evolui, esses conceitos vão sendo melhor compreendidos e suas definições tornam-se mais claras. A definição e diferenciação entre esses conceitos são discutidos a seguir (LYYTINEN; YOO, 2002). A **computação móvel** baseia-se na capacidade de mobilidade de dispositivos, levando os serviços computacionais, ou seja, o computador torna-se um dispositivo sempre presente que expande a capacidade de um usuário utilizar os serviços que um computador oferece, independentemente de sua localização. Combinada com a capacidade de

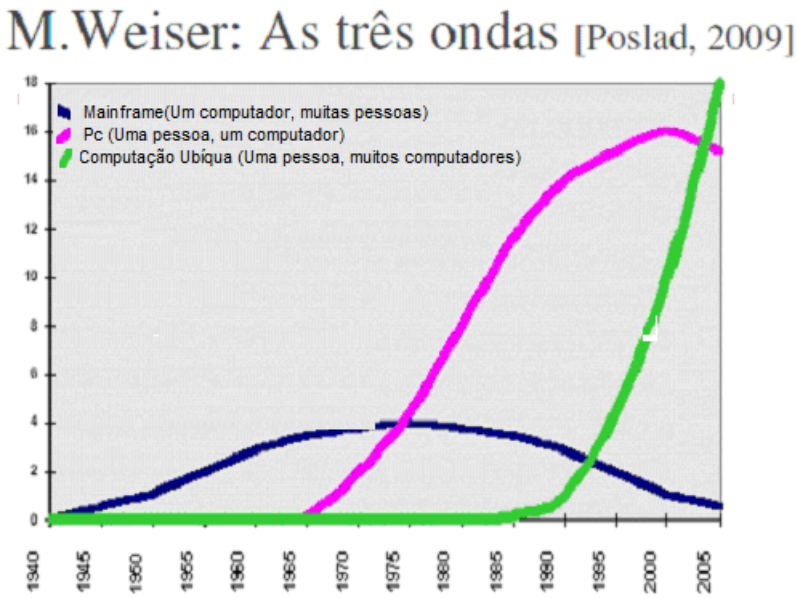

Figura 1.1: As Três Grande Fases da Computação

acesso, a computação móvel tem transformado a computação em uma atividade que pode ser transportada para qualquer lugar. A **computação pervasiva** implica que o computador está embarcado no ambiente de forma invisível para o usuário. O ambiente também pode e deve ser capaz de detectar outros dispositivos que venham a fazer parte dele. Desta interação surge a capacidade de computadores agirem de forma "inteligente" no ambiente no qual nos movemos, um ambiente povoado por sensores e serviços computacionais. A **computação ubíqua** beneficia-se dos avanços da computação móvel e da computação pervasiva. A computação ubíqua surge, então, da necessidade de se integrar mobilidade e computação pervasiva, ou seja, qualquer dispositivo computacional, enquanto em movimento, pode construir dinamicamente modelos computacionais dos ambientes nos quais nos movemos e configuramos seus serviços dependendo da necessidade.

Tendo em vista as definições mencionadas acima, o termo computação ubíqua será usado aqui como uma junção da computação pervasiva e da computação móvel. A justificativa de se realizar uma diferenciação desses termos é que um dispositivo que está embutido em um ambiente não necessariamente é móvel. Devido a isso, quando for utilizado o termo computação ubíqua, considerar-se-ão o alto grau de dispositivos embarcados da computação pervasiva juntamente com o alto grau de mobilidade da computação móvel, conforme mostra a Figura 1.2.

Portanto, a computação ubíqua usa uma grande variedade de dispositivos, sensores e redes integrados para formar um ambiente distribuído, altamente heterogêneo e integrado às atividades diárias dos usuários. Tipicamente, aplicações ubíquas recebem dados de sen-

Figura 1.2: Relação entre Computação Ubíqua, Pervasiva e Móvel.

sores, de dispositivos e de provedores de serviços, gerenciam ações de usuários e oferecem suporte à mobilidade. Tais aplicações são compostas por serviços, fornecidos por diversos provedores de serviços, e são cientes de contexto, ou seja, usam informações de contexto para a realização das suas tarefas. Segundo DEY e ABOWD, (2000), contexto é *"qualquer informação que pode ser usada para caracterizar a situação de uma entidade"*. Uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. Dentre essas informações, destacam-se os parâmetros de qualidade, que são atributos responsáveis por medições de qualidade acerca dos serviços utilizados e providos pelas aplicações (BATISTA et al., 2012). No cenário das aplicações ubíquas, é fundamental representar essas informações que dão suporte à ubiquidade desde as fases iniciais do projeto de desenvolvimento de software.

1.1 Motivação

Há vários desafios na computação ubíqua (KUMAR, 2009): (i) o suporte a vários tipos de eventos, como eventos da aplicação, mudanças no ambiente, eventos de troca de dados; (ii) a adaptação do sistema em tempo de execução, permitindo descoberta e localização de serviços; (iii) a integração de vários tipos de elementos computacionais, como sensores, atuadores e smartfones; e (iv) o gerenciamento das comunicações, incluindo questões de mobilidade e segurança. As soluções para esses problemas não são triviais e podem envolver o suporte de vários elementos em cooperação.

Essa natureza heterogênea e complexa torna difícil o projeto e implementação de sistemas ubíquos, e aumenta o custo de construí-los. De forma a sistematicamente organizar os principais elementos que fazem parte de um sistema ubíquo, suas responsabilidades e interações, provendo um entendimento comum da arquitetura de sistemas para esse domínio, é necessário se estabelecer uma *arquitetura de referência para sistemas ubíquos.* Segundo BASS, CLEMENTS e KAZMAN, (2012), uma arquitetura compreende os elementos

de software, as propriedades externamente visíveis desses elementos e o relacionamento entre as partes que a compõe. Arquiteturas de referência avaliam os elementos essenciais que compõem arquiteturas de um domínio específico, definindo um vocabulário de tipos e um conjunto de restrições referentes a esse domínio. Essas arquiteturas também provêem direções para a especificação de arquiteturas concretas pertencentes a um dado domínio de aplicação (ANGELOV et al., 2008) (NAKAGAWA; OQUENDO; BECKER, 2012). Por conseguinte, podem influenciar diretamente na qualidade e no projeto de todo um conjunto de arquiteturas concretas e no conjunto de sistemas derivados dessa arquitetura (ANGELOV; GREFEN; GREEFHORST, 2009). Em suma, uma arquitetura de referência (NAKAGAWA; ANTONINO; BECKER, 2011) cria um repositório de conhecimento arquitetural relacionado a um domínio específico, reduz o risco de se projetar uma arquitetura não viável, reduz o custo do projeto arquitetural e aumenta sua qualidade. Além disso, o uso de uma arquitetura de referência como base do desenvolvimento de um sistema reduz o tempo de desenvolvimento. Como sistemas ubíquos estão cada vez mais comuns no nosso quotidiano é essencial prover uma arquitetura de referência para capturar a essência dessa importante classe de sistemas, bem como garantir a padronização e interoperabilidade entre eles. Complexidade e interoperabilidade são aspectos críticos em sistemas ubíquos, que demandam o suporte de uma arquitetura de referência tanto para definição arquitetural quanto para evolução.

Além de arquiteturas de referências, no contexto de arquitetura de software é importante representar o modelo arquitetural das aplicações ubíquas. Para essa finalidade, as linguagens de descrição arquitetural (ADL - *Architecture Description Language*) (SHAW; GARLAN, 1996) (CLEMENTS, 1996) (MEDVIDOVIC; TAYLOR, 2000a) são bastantes usadas. ADLs são linguagens de alto nível para representar e analisar projetos arquiteturais, provendo um *framework* conceitual e uma notação sintática para caracterizar arquiteturas de software em termos de seus elementos e relacionamentos. Essas linguagens essencialmente oferecem abstrações para representação da arquitetura de um software através de componentes, conectores e configurações. Os *componentes* representam as funcionalidades do software, os *conectores* são elementos de comunicação entre componentes, e a *configuração* é utilizada para descrever o relacionamento entre componentes e conectores. Além de representar os componentes essenciais, uma ADL pode oferecer recursos para a especificações de informações adicionais, que possam ser utilizadas durante o projeto arquitetural ou em situações de adaptação quando a arquitetura precisa ser modificada. No caso de aplicações ubíquas, no entanto, a literatura não reporta ADLs específicas que disponibilizem abstrações para representar informações essenciais para tais aplicações, como

metadados, representação de contexto e especificação de condições de reconfiguração.

1.2 Questões de Pesquisa

Visando atender os desafios relacionados com a modelagem de aplicações ubíquas, as seguintes questões de pesquisa nortearam o desenvolvimento deste trabalho:

- **QP1:** Quais as características essenciais de aplicações ubíquas?
- **QP2:** Quais os elementos que caracterizam uma arquitetura para computação ubíqua e como eles se relacionam?
- **QP3:** Como representar adequadamente uma arquitetura de software de aplicações ubíquas considerando as características inerentes desse tipo de aplicação?

Com base nas questões de pesquisas supracitadas, elaboramos as seguintes Hipóteses:

- **H1:** A realização de uma revisão sistemática da literatura pode identificar as características essenciais de aplicações ubíquas.
- **H2:** A elaboração de uma arquitetura de referência para computação ubíquas pode expressar os elementos que caracterizam esse tipo de sistema, bem como a relação entre eles.
- **H3:** Uma linguagem de descrição arquitetural que considere as características essenciais de aplicações ubíquas identificadas em uma revisão sistemática, bem como a arquitetura de referência para computação ubíqua, pode ser adequada para definir a arquitetura, endereçando as características dessa aplicações.

1.3 Objetivo

O objetivo geral deste trabalho é propor e desenvolver uma Linguagem de Descrição Arquitetural baseada em uma Arquitetura de Referência para possibilitar a modelagem de aplicações ubíquas.

1.4 Metodologia

Para esse fim, a metodologia usada nesse trabalho consiste nas seguintes tarefas:

Figura 1.3: Representação Gráfica do Objetivo da Tese.

1. Realizar uma revisão sistemática, de forma investigar na literatura relacionada com sistemas ubíquos, os elementos comuns a esses sistemas;

2. Propor e validar uma arquitetura de referência, para sistemas ubíquos, com base nos elementos comuns encontrados na revisão sistemática;

3. Propor uma linguagem de descrição arquitetural, UbiACME, para permitir a especificação arquitetural de sistemas ubíquos. Essa linguagem deve basear-se nas características levantadas na revisão sistemática e na arquitetura de referência, incluindo suporte para elementos comuns de sistemas ubíquos, tais como: representação de contexto, adaptação dinâmica e representação de parâmetros de qualidade de serviços e de contexto.

4. Validar UbiACME através de estudos de caso e experimento controlado com o usuário.

Cada um dos itens da metodologia supracitados possui um produto fundamental para o conjunto de documentos e ferramentas almejado. Esses produtos complementam-se por fornecerem uma estrutura conceitual para o desenvolvimento da ADL para computação ubíqua. Conforme mencionado na metodologia e ilustrado na Figura 1.3, para realizar o objetivo, isto é, construir a Linguagem de Descrição de Arquitetura para sistemas ubíquos, foi necessário realizar uma revisão sistemática para identificar os elementos essenciais para sistemas ubíquos e elaborar uma arquitetura de referência para organizar os elementos já encontrados na revisão sistemática deixando clara a complementariedade de cada um desses passos para o desenvolvimento da ADL. Essa ADL deverá ser construída a partir das necessidades identificadas pela arquitetura de referência, que por sua vez inclui os elementos arquiteturais identificado na revisão sistemática. Dessa forma, pretende-se estabelecer uma linguagem simples, objetiva e suficiente para descrever qualquer tipo de aplicação ubíqua. Experiências adquiridas pelo grupo de pesquisa no qual este trabalho está inserido sugere a adoção da ADL ACME (GARLAN; MONROE; WILE, 2000) como base para a implementação do objetivo. Essa escolha baseia-se principalmente nas seguintes

características de ACME: (i) é uma linguagem de domínio geral, que pode ser aplicada em diversos contextos; (ii) apresenta uma ontologia comum que permite intercâmbio de informações arquiteturais entre descrições em linguagens distintas; (iii) possui uma extensão, Armani (MONROE, 1998), que permite a definição de restrições a partir de expressões em lógica de primeira ordem; (iv) apresenta mecanismos para definição de famílias (ou estilos) arquiteturais.

1.5 Estrutura do Documento

Este estudo está organizado da seguinte forma. O Capítulo 2 apresenta os conceitos básicos referentes a este trabalho, a saber, computação ubíqua, arquitetura de referência, ADLs e Acme/Armani, Revisão Sistemática. O Capítulo 3 apresenta os trabalhos relacionados aos objetivos específicos deste trabalho e está dividido em três sub-seções: (i) revisões sistemáticas sobre arquiteturas de sistemas ubíquos; (ii) arquiteturas de referência para computação ubíqua; e por fim, (iii) ADLs para computação ubíqua. O Capítulo 4 apresenta uma revisão sistemática sobre elementos arquiteturais para sistemas ubíquos. O Capítulo 5 apresenta a arquitetura de referência RA-Ubi. O Capítulo 6 apresenta UbiACME, uma linguagem de descrição arquitetural para sistemas ubíquos. O Capítulo 7 apresenta a ferramenta para modelagem de UbiACME, chamada *UbiACME Studio*. O Capítulo 8 apresenta o experimento controlado que avalia a linguagem e a ferramenta. Por fim, o Capítulo 9 apresenta as considerações finais e trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo apresentamos os conceitos básicos relacionados a este trabalho. A Seção 2.1 apresenta conceitos relacionados a computação ubíqua. A Seção 2.2 discorre sobre arquitetura de referência. A Seção 2.3 discursa sobre as linguagem de descrição arquitetural ACME e *Armani*. A Seção 2.4 apresenta conceitos relacionados com revisão sistemática.

2.1 Computação Ubíqua

Na computação ubíqua a computação move-se para fora das estações de trabalho e computadores pessoais (PCs), tornando-se pervasiva em nossa vida cotidiana. *Marc Weiser*, considerado o pai da computação ubíqua, vislumbrou há uma década que, no futuro, computadores habitariam os mais triviais objetos: etiquetas de roupas, xícaras de café, interruptores de luz, canetas, etc, de forma invisível para o usuário.

A computação ubíqua beneficia-se dos avanços da computação móvel e da computação pervasiva, cujo computador tem a capacidade de obter informação do ambiente no qual ele está embarcado e utilizá-la para dinamicamente construir modelos computacionais, ou seja, controlar, configurar e ajustar a aplicação para melhor atender as necessidades do dispositivo ou usuário. A computação ubíqua surge então da necessidade de se integrar mobilidade com a funcionalidade da computação pervasiva, ou seja, qualquer dispositivo computacional, enquanto em movimento conosco, pode construir, dinamicamente, modelos computacionais dos ambientes nos quais nos movemos e configurar seus serviços dependendo da necessidade.

O termo ubíquo tem origem no Latim. *"Ubiquu"* é um adjetivo para qualificar o que está ao mesmo tempo em toda a parte. *Weiser*, considerado um dos pesquisadores que

primeiro definiu a computação ubíqua, previu que computadores desapareceriam. Isto reflete a ideia de que computação tornar-se-á embutida e invisível. Itens do dia-a-dia que, tradicionalmente, não apresentam capacidade computacional, passarão a ter. (WEISER, 1988).

Uma das principais características da computação ubíqua é a ciência do contexto *(context-awareness)*. A proposta dessa área é, em linhas gerais, elaborar uma maneira de coletar, para dispositivos computacionais, entradas capazes de refletir as condições atuais do usuário, do ambiente no qual ele se encontra e do próprio dispositivo computacional utilizado, considerando tanto suas características de hardware, como também de software e de comunicação. Tais entradas são os chamados *contextos*. Vários pesquisadores publicaram trabalhos no início da década de 90, descrevendo definições para o significado do termo contexto: (LOUREIRO A.A.FERREIRA, 2009).

De um modo geral, os sistemas ubíquos funcionam através do sensoriamento e processamento de informações do contexto no qual o usuário está inserido, a definição que melhor representa o que vem a ser contexto foi proposta em (DEY, 2001):

"Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação."

Isso é, informações de contexto são qualquer informação que descreva as características de uma entidade (pessoa, lugar, objeto), como por exemplo, o nome de uma pessoa, a temperatura de uma sala, o nível de energia na bateria de um *handheld*, etc. Assim, as aplicações que utilizam informações de contexto para reagir proativamente em favor do usuário são conhecidas como aplicações sensíveis ao contexto.

Para que as informações de contexto possam ser utilizadas pelas aplicações, é necessário que haja algum(ns) mecanismo(s) que seja(m) capaz(es) de obter as informações de contexto do ambiente, representar essas informações de modo que as aplicações possam entender o seu significado e ainda que raciocinem acerca dessas informações de contexto para que possam ser transformadas em informações relevantes para os usuários.

2.2 Arquitetura de Referência

Uma Arquitetura de Referência contém um conjunto das melhores práticas arquiteturais que podem ser adotadas, sistematicamente, em todos os projetos de uma organização.

A missão de uma arquitetura de referência é fornecer uma base a partir da qual os projetos podem iniciar seu ciclo de vida. Com ela, o arquiteto do projeto poderá concentrar seu esforço em resolver questões arquiteturais específicas de seu projeto sem se preocupar em projetar a resolução de problemas já solucionados.

Um projeto que não possui uma arquitetura de referência como base, não necessariamente vai falhar; ele vai demandar um esforço muito maior por parte da equipe de arquitetura no sentido de pesquisar, investigar e ponderar sobre as decisões arquiteturais.

Segundo o RUP *(Rational Unified Process)*(RATIONAL, 2001), uma arquitetura de referência: é, em essência, um padrão pré-definido de arquitetura, ou um conjunto de padrões, possivelmente parcial ou totalmente instanciado, projetado e comprovado para uso em negócio particular e contextos técnicos, juntamente com o apoio de artefatos para permitir a sua utilização. Muitas vezes, esses artefatos são recolhidos a partir de projetos anteriores.

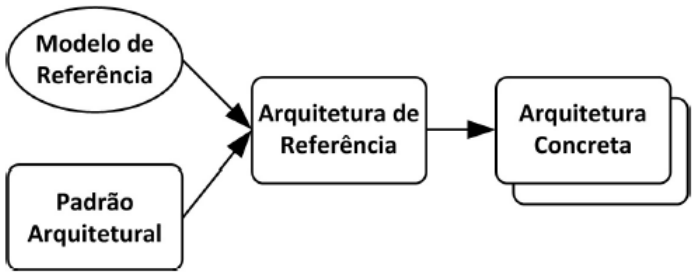

Figura 2.1: Relação entre Modelo de Referência, Padrão Arquitetural, Arquitetura de Referência e Arquitetura Concreta.(BASS; CLEMENTS; KAZMAN, 2003)

Segundo BESS et al. (2003), diferenciamos modelo de referência de arquitetura de referência, pois, um modelo de referência é uma decomposição de problemas em partes que cooperativamente resolvem o problema. Já a arquitetura de referência é um modelo de forma cooperativa que implementa a funcionalidade do modelo de referência.

Um padrão arquitetural é um descrição do problema e a essência da sua solução, responsável por documentar boas soluções para problemas recorrentes e deve ser suficientemente abstrato para ser reusado em aplicações diferentes.

A partir de um modelo de referência e de padrões arquiteturais, arquiteturas de referências podem ser derivadas e essas arquiteturas, quando instanciadas, dão origem a arquiteturas de software concretas, como ilustra a Figura 2.1.

Segundo ANGELOV e GREFEN (2008) e NAKAGAWA et al. (2012), arquiteturas de referên-

cia têm se destacado por prover direções para a especificação de arquiteturas concretas pertencentes a um domínio de aplicação.

Arquiteturas de referência podem ser utilizadas como guia para o projeto de arquiteturas concretas ou com artefato de padronização, provendo interoperabilidade entre sistemas ou componentes de sistemas (MULLER, 2008). Dessa forma, uma arquitetura de referência pode ser utilizada em diversos contextos, resultando no desenvolvimento de instâncias arquiteturais distintas, dependentes dos *stakeholders* e das metas pertencentes a cada instituição, como ilustra a Figura 2.2

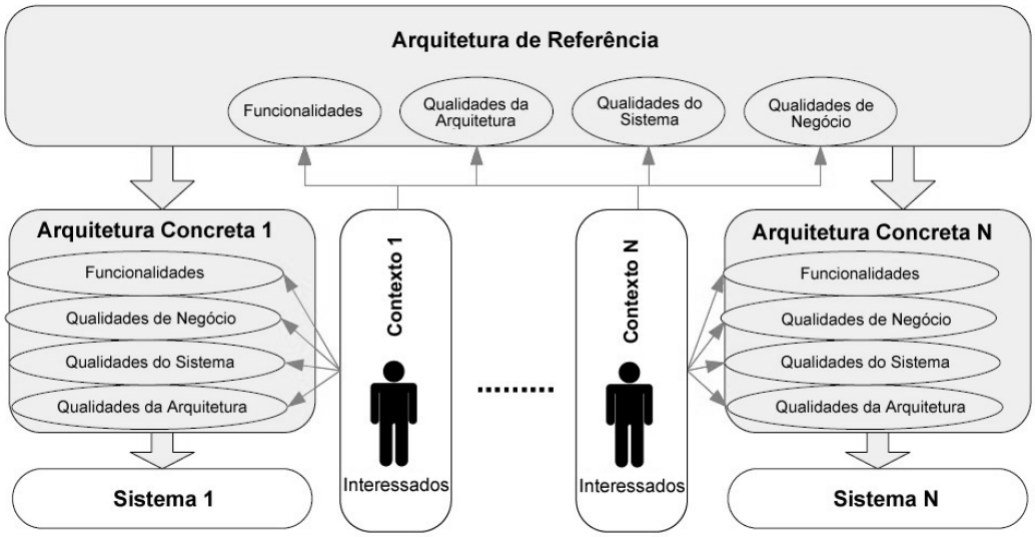

Figura 2.2: Interação de Interessados e Contextos entre Arquiteturas Concretas e Arquiteturas de Referência (ANGELOV et al., 2008).

A Figura 2.2 mostra que, dependendo das funcionalidades, da qualidade da arquitetura, da qualidade dos sistemas e da qualidade dos negócios que cada interessado tem, uma arquitetura de referência pode instanciar arquitetura(s) concreta(s) específica(s) para um domínio. Ou seja, de uma arquitetura de referência, podemos instanciar diversas arquiteturas concretas para sistemas em principio independentes.

2.2.1 Tipos de Arquiteturas de Referência

Dada a diversidade de arquiteturas de referência, ANGELOV et al. (2009) propõem um *framework* para a classificação dessas arquiteturas na qual se avalia o seu **contexto**, **objetivos** e **detalhes de projeto**.

Para determinar o **contexto** de uma arquitetura de referência, investigam-se os **aspectos de projeto**, isto é, para quantas organizações a arquitetura de referência interessa,

se para uma única organização ou múltiplas organizações; as **aplicações que podem afetar seus objetivos de negócio**, isto é, quais são as principais categorias de interessados envolvidos no projeto dessa arquitetura, organizações de software, de usuários, centros de pesquisa ou uma organização padronizadora; e a **especificação da arquitetura**, isto é, em qual momento ela é construída, antes (denominada preliminar) ou depois (denominada clássica) da existência de arquiteturas concretas do domínio.

Para determinar os **objetivos** de uma arquitetura de referência, avalia-se **por qual razão ela é proposta**, se para padronizar arquiteturas concretas do domínio (visando a interoperabilidade) ou se para servir como um mecanismo facilitador no projeto de arquiteturas concretas desse domínio (visando o fornecimento de diretrizes para o projeto de novos sistemas na forma de padrões, esquemas,etc).

Para determinar o **projeto** de uma arquitetura de referência, avalia-se (Angelov et al., 2009): o **conteúdo dessas arquiteturas**, se elas descrevem componentes, interfaces, protocolos, algoritmos e/ou políticas e diretrizes; o **nível de detalhamento** com o qual esses elementos são definidos (detalhado, semidetalhado ou superficial); o **nível de abstração adotado** (concreta, semiconcreta ou abstrata); e o **nível de formalização** da arquitetura de referência, os tipos de técnicas empregadas em sua representação (formais, semiformais ou informais). As técnicas **formais** são aquelas baseadas em sintaxes e semânticas formalmente definidas e que permitem a descrição não ambígua da arquitetura. Essas técnicas são mais fáceis de serem compreendidas por ferramentas.As técnicas **semiformais** são aquelas que combinam, em parte, o rigor das linguagens formais à compreensibilidade da linguagem natural. Ja as técnicas **informais** são aquelas mais próximas da linguagem natural. Essas técnicas são mais fáceis de serem compreendidas por seres humanos.

2.2.2 Como Projetar uma Arquiteturas de Referência

Em um dos seus trabalhos, EKLUND et al. (2005) aponta a necessidade de se formalizar o processo por meio do qual a arquitetura de referência é projetada, uma vez que processos informais ainda têm sido muito utilizados. Nesse contexto, a proposta de MATTHIAS CALSTER e PARIS AUGERIOU (2011), nomeada de arquiteturas de referência empiricamente fundamentadas, ajuda a projetar uma arquitetura de referência de forma sistemática e consiste em seis passos executados pelo arquiteto de software e especialista de domínio. Os passos que garantem a fundamentação empírica são: Passo 1: decisão sobre o tipo da arquitetura de referência; Passo 2: seleção das estratégias do projeto; Passo 3: aquisição

empírica dos dados; Passo 4: construção da arquitetura de referência; Passo 5: ativação da arquitetura de referência; e para garantir a validação empírica, o passo 6: avaliação da arquitetura de referência. Esta abordagem ajuda a projetar qualquer arquitetura de referência a partir do zero ou com base em artefatos de arquiteturas existentes (GALSTER; AVGERIOU, 2011). Um outro projeto que se destaca e que usamos em nosso trabalho é a proposição do ProSA-RA (NAKAGAWA et al., 2009), um processo que visa a construção, representação e avaliação de arquiteturas de referência orientadas a aspectos. Contudo, o processo também tem sido utilizado na construção de arquiteturas de referência de outros domínios como, por exemplo, para ferramentas de visualização (NAKAGAWA et al., 2009) e sistemas marítimos (BORG, 2011). Os passos para aplicação do ProSA-RA são ilustrados na Figura 2.3 e explicados a seguir.

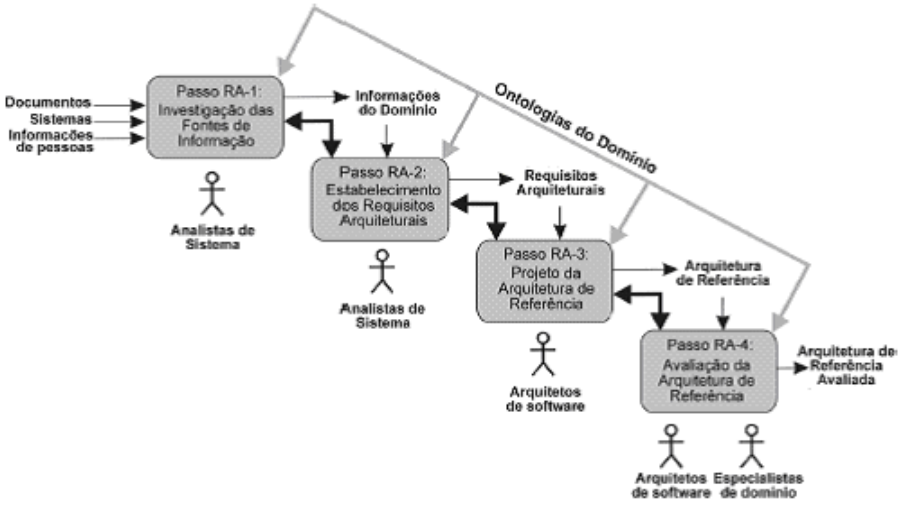

Figura 2.3: Estrutura Geral do ProSA-RA (NAKAGAWA et al., 2009).

O **Passo RA-1 - Identificação das Fontes de Informação**: refere-se a investigação e seleção de fontes de informação com o propósito de levantar informações sobre o domínio para o qual a arquitetura de referência está sendo criada;

O **Passo RA-2 - Estabelecimento dos Requisitos Arquiteturais**: refere-se a identificação dos requisitos da arquitetura de referência com base nas fontes de informações. Para isso, primeiramente identificam-se os requisitos de sistemas do domínio e, em seguida,estabelecem-se quais são os requisitos específicos da arquitetura de referência, isto é, os requisitos arquiteturais. A seguir, os requisitos arquiteturais são mapeados para os conceitos do domínio alvo. Nesse ponto, identificam-se aqueles conceitos que possuem característica transversal, ou seja, interesses transversais;

O **Passo RA-3 - Projeto da Arquitetura de Referência**: a partir dos requisitos arquiteturais e dos conceitos identificados no passo anterior, é estabelecida uma descrição arquitetural da arquitetura de referência, considerando-se também padrões e estilos arquiteturais; A seguir, constrói-se a descrição arquitetural utilizando-se visões arquiteturais. Particularmente, as visões de módulos, de tempo de execução, de implantação e conceitual têm sido propostas para descrever essas arquiteturas (NAKAGAWA E. Y.; MALDONADO, 2008); contudo, outras visões podem ser relevantes considerando-se diferentes interessados ou o uso e a natureza da arquitetura de referência em questão; e

Passo RA-4 - Avaliação da Arquitetura de Referência: a ideia nesse passo é a utilização de um *checklist* para a avaliação da arquitetura. O *checklist* define uma lista de questões que guiam um revisor para a detecção de defeitos em documentos relacionados às arquiteturas de referência.

2.3 ADLs / ACME / ARMANI

No contexto de arquitetura de software, as Linguagens de Descrição Arquitetural *(Architectual Description Language*, ADLs) (SHAW; GARLAN, 1996) (CLEMENTS, 1996) (MEDVIDOVIC; TAYLOR, 2000b) têm sido usadas na especificação da arquitetura de sistemas para expressar componentes e relacionamentos arquiteturais. Segundo (MEDVIDOVIC; TAYLOR, 2000b) ADLs propõem uma sintaxe e uma estrutura conceitual que permite caracterizar uma arquitetura de forma não ambíqua, para prover representações formais para arquitetura de software.

Inúmeras linguagens de descrição de arquitetura têm sido criadas. Em geral, cada uma oferece capacidades específicas. Por exemplo, **AESOP** (GARLAN; ALLEN; OCKERBLOOM, 1994) permite o uso de estilos arquiteturais; **WRIGHT** (ALLEN, 1997) dá suporte à especificação e análise de interações entre componentes arquiteturais; **RAPIDE** (LUCKHAM, 1996) permite a simulação e análise de diferentes soluções arquiteturais; **AADL** (FEILER; LEWIS; VESTAL, 2003) para especificar arquiteturas de sistemas, contêm construções e recursos úteis para modelar uma ampla variedade de sistemas embarcados e em tempo real; **ALI** (BASHROUSH et al., 2008) especifica descrição de arquiteturas para aplicações industriais.

ACME (GARLAN; MONROE; WILE, 2000) destaca-se por ser uma ADL de propósito geral que fornece um quadro estrutural simples para representar arquiteturas e permitir o desenvolvimento de arquitetura compatível com múltiplas ADLs. Para isso, ACME provê

uma ontologia com oito entidades usadas para representação arquitetural. A Figura 2.4 ilustra os principais elementos de ACME.

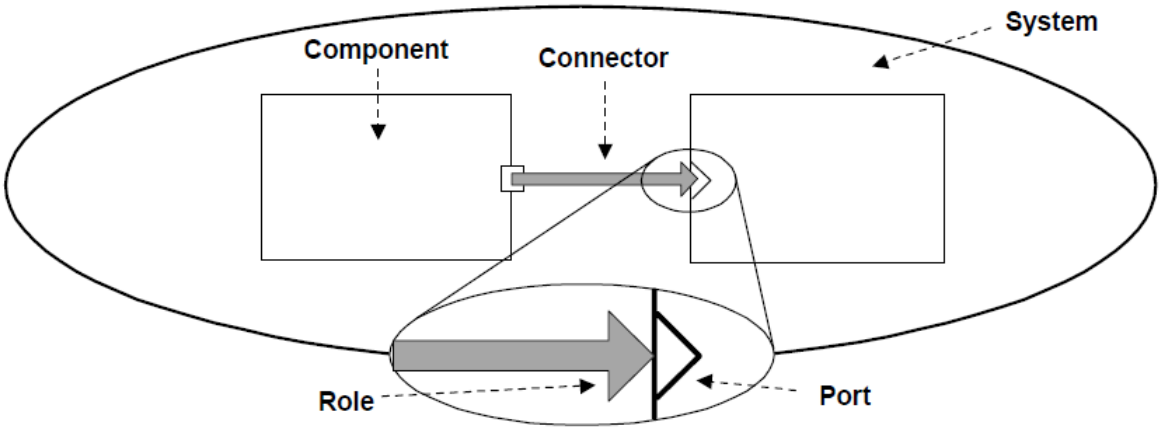

Figura 2.4: Representação Gráfica de Alguns Elementos de uma Descrição ACME. (GARLAN; MONROE; WILE, 2000).

Components (Componentes) representam os componentes arquiteturais, que são abstrações computacionais responsáveis por processamento, armazenamento ou manipulação de dados; ***Connector*** (Conector) é a entidade responsável por representar a abstração conector, que é o mecanismo de interação entre componentes; ***Systems*** (Sistemas) são abstrações para sistemas inteiros, que por sua vez são constituídos de um conjunto de componentes, um conjunto de conectores e um conjunto de *attachments* que descrevem a topologia do sistema; ***Ports*** (Portas) são as interfaces dos componentes. Conforme supracitado, cada porta identifica um ponto de interação entre o componente e seu ambiente. Um componente pode prover múltiplas interfaces usando diferentes tipos de portas. ***Roles*** (Papéis) são as interfaces dos conectores. ***Representation*** (Representação) correspondem às representações, sendo exibições arquiteturais de um elemento ou decomposições mais detalhadas de um determinado elemento (componente, conector, porta, ou papel) para descrever em maior detalhe, de modo que possa ser visto como uma representação mais refinada de um elemento e, assim, concretizar a noção de que esse elemento pode ter várias implementações alternativas. Elementos internos a uma representação podem ter suas interfaces disponibilizadas para o exterior, através do uso de elemento ***Binding***, responsável por associar interfaces de mesmo tipo. ***Attachments*** compõem a seção de configuração de uma arquitetura e definem um conjunto de associações entre as portas dos componentes e os papeis dos conectores; ***Properties*** são mecanismos de ACME para acomodar uma ampla variedade de informações auxiliares, como anotar desenhos e elementos de projetos com informações detalhadas, geralmente não estrutural. A linguagem

ACME permite anotação de estruturas arquiteturais com listas de propriedades, cada propriedade tem: nome, tipo opcional e valor.

Além dos elementos arquiteturais básicos, ACME também permite a definição de estilos arquiteturais (SHAW; GARLAN, 1996), a fim de aumentar reúso e expressividade, o que é feito através dos elementos ***Type e Family***. Em ACME, um estilo arquitetural define uma família de sistemas em termos de um padrão de organização estrutural, definindo um vocabulário de tipos de elementos que poderão ser utilizado em uma etapa posterior. O elemento ***Type*** é usado para definir um vocabulário de tipos abstratos de elementos como componentes, conectores, portas e papeis, e o elemento ***Family*** é usado para definir estilos arquiteturais em termos de um vocabulário de tipos abstratos de elementos.

A Figura 2.5 exibe a representação gráfica em ACME do sistema *Smart Car*, baseado em propostas existentes para um veículo autonômico, mostrando um conjunto de elementos arquiteturais responsáveis pelo tratamento do Piloto automático em autoestradas. Esse sistema recupera as informações da estrada, como: tráfego (fluxo de veículos), velocidade máxima, condições climáticas (temperatura, umidade, etc), condições da pista (acesso, visibilidade, etc). Nessa descrição existem três componentes fundamentais, que são os componentes que representam os elementos físicos dos pedais e do volante *(Wheel e Pedals)*, e o componente que representa o sub sistema de navegação *(NavigationSystem)*, que atua sobre o motor, freios e rodas do veículo para realizar as operações de mudança de direção ou velocidade. O conector *PilotingMechanism* é responsável pela comunicação entre os controladores do veículo e o atuador *NavigationSystem*, para que o veículo responda de acordo com os comandos.

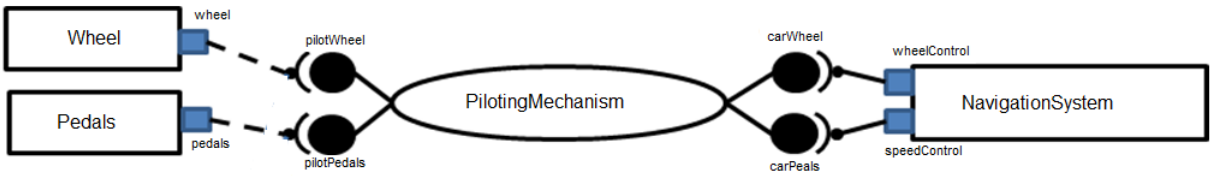

Figura 2.5: Representação Gráfica em ACME do Sistema SmartCar.

A Figura 2.6 exibe uma descrição textual simplificada em ACME do sistema *SmartCar*. Essa descrição possui três componentes: *NagivationSystem* (linha 2), *Wheel* (linha 6) e *Pedals* (linha 7) e um conector *PilotingMechanism* (linha 8) Os *attachments* realizados nas linhas 13 e 14 associam as portas dos componentes aos roles do conector, realizando a comunicação entre os componentes. O componente *NagivationSystem* (linha 2) possui em sua descrição um elemento *Properties* (linha 3) e dois elementos *Port: WheelControl*

(linha 4) e *SpeelControl* (linha 5).

```
1  System Smart_Car = {
2     Component NavigationSystem = {
3          Property locationId;
4          Port wheelControl;
5          Port speedControl; }
6     Component Wheel = { Port wheel; }
7     Component Pedals = { Port pedals; }
8     Connector PilotingMechanism = {
9          Role pilotWheel;
10         Role carWheel;
11         Role pilotPedals;
12         Role carPedals; }
13    Attachment PilotingMechanism.carWheel to NavigationSystem.wheelControl;
14    Attachment PilotingMechanism.carPedals to NavigationSystem.speedControl;
      ⋮
```

Figura 2.6: Definição dos Elementos Principais em ACME da Arquitetura Responsável pelo Sistema SmartCar.

Armani (MONROE, 1998) é uma linguagem extensão da ADL ACME que oferece uma linguagem de predicados baseada em lógica de primeira ordem usada para expressar restrições arquiteturais sobre elementos ACME. Restrições são definidas em termos de invariantes *(invariants)* ou *heurísticas (heuristics)*. *Invariantes* são restrições de projeto que devem ser seguidas e heurísticas são sugestões que podem ou não ser seguidas. As invariantes em *Armani* são potencialmente úteis para garantir que um sistema preserve as restrições impostas pela arquitetura do software apesar da inserção ou remoção dinâmica de elementos ACME. Heurísticas, por sua vez, podem também ser utilizadas como potenciais indicadores de qualidade, ou como representação de alguns requisitos não-funcionais. *Armani* é basicamente composta de um conjunto de funções primitivas, um conjunto de operadores, um conjunto de quantificadores e um mecanismo para definição de novas funções. As funções primitivas são usadas na definição de invariantes, heurísticas e criação de novas funções. As funções primitivas podem ser agrupadas em funções de tipos, funções de grafo, funções de propriedades e funções de conjunto. O conjunto de operadores é composto de operadores de comparação, operadores lógicos, operadores aritméticos, etc. O conjunto de quantificadores possui os quantificadores universal *(forall)* e existencial *(exists)*. Finalmente, o elemento *analysis* permite criar novas funções usando funções primitivas ou funções previamente definidas pelo usuário.

A Figura 2.7 contém um exemplo de uma outra versão do sistema *SmartCar* em ACME usando os elementos de *Armani*; invariantes *(invariants)* ou heurísticas *(heuristics)*.

O elemento *Invariant* da (linha 6) define que deve existir um componente c, que possui uma porta p, no qual c está conectada à porta *wheelControl*. Em outras palavras,

```
1 System smart_car = {
   ...
2     Component NavigationSystem = {
3           Property locationId : float;
4           Port wheelControl;
5           Port speedControl;
6           Invariant exists Component c in smart_car|exists port p in c |
                        isAttacged(self.wheelControl, p);
7           Invariant exists Component c in smart_car| exists Port p in c |
                        IsAttached(self.speedControl, p);
8           Heuristic locationId > 0;
9           }
10 }
```

Figura 2.7: Exemplo do Uso de Elementos Armani (Invariant,Heuristic).

a porta *wheelControl* do componente *NavigationSystem* deverá estar conectada a alguma porta de algum componente, necessariamente. O elemento *Invariant* (linha 7) define regra semelhante à invariante da (linha 6), entretanto, refere-se à porta *speedControl*. Essas duas invariantes garantem que vão existir portas conectadas às portas *wheelControl* e *speedControl*, o que implica que necessariamente haverá um canal de comunicação entre o componente*NavigationSystem* e algum outro componente ou conjunto de componentes. O elemento *Heuristic* (linha 8) define que *locationId* deve ser maior que 0. Uma vez que *locationId* é uma enumeração, valores negativos ou iguais a 0 significam que a localização não está registrada no sistema, não possuindo, portanto, um identificador. Essa regra é uma heurística, pois apesar de atribuir um estado de inconsistência para o sistema pode ser quebrada, por não possuir importância fundamental.

2.4 Revisão Sistemática

Revisão sistemática (DYBA; KITCHENHAM; JORGENSEN, 2005) é uma técnica utilizada para resumir, avaliar e interpretar as evidências relacionadas a uma questão específica, área de tópico ou fenômeno de interesse. Nesse contexto, uma evidência individual (por exemplo, um caso de estudo ou estudo experimental divulgado em publicação), que contribui para uma revisão sistemática é chamado estudos primários. A revisão sistemática é uma técnica que se originou a partir da *Evidence-Based Software Engineering* (EBSE) (DYBA; KITCHENHAM; JORGENSEN, 2005).

A revisão sistemática é importante para identificar uma variedade de trabalhos que podem envolver teorias e conceitos, relatos de desenvolvimento tecnológicos, resultados de pesquisas experimentais e outros.

Para aplicar uma revisão sistemática é necessário escolher um tópico específico de

estudo e seguir o processo proposto por (KITCHENHAM, 2004). Tal processo é composto por três fases principais: Planejamento, Execução e Avaliação. A Figura 2.8 ilustra essas etapas.

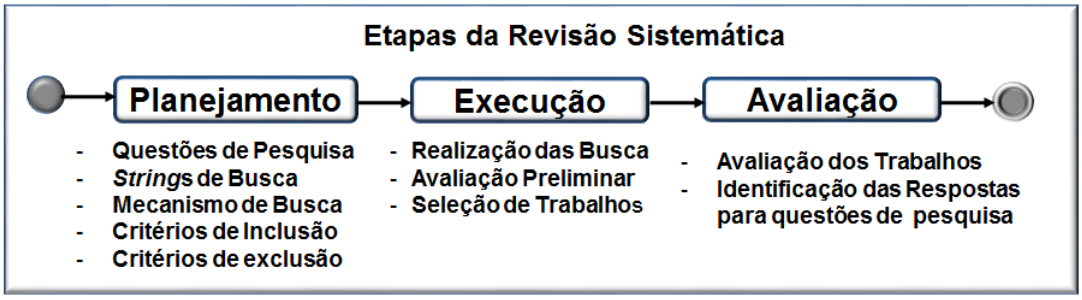

Figura 2.8: Representação Gráfica das Fases de uma Revisão Sistemática.

Na fase de **planejamento**, os objetos e o protocolo de revisão sistemática são definidos. O protocolo orienta a condução da revisão sistemática e é composto por: (a) **questões da pesquisa,** (b) **estratégia de busca**, (c) **critérios de inclusão** e **exclusão e (d) extração de dados**.

As **questões de pesquisa** são definidas para encontrar todos os estudos primários que compreendem e sintetizam evidências sobre a abordagens propostas ou utilizadas para representar o tópico de interesse da pesquisa. Por exemplo, se o tópico de interesse da pesquisa é "arquiteturas de referência", então, uma questão usada (RQ) poderia ser:

RQ1: Quais são as abordagens para representar arquiteturas de referência?

A **estratégia de busca**: No âmbito da recuperação da informação, a estratégia de busca pode ser definida como uma técnica ou conjunto de regras para tornar possível o encontro entre uma pergunta formulada e a informação armazenada em uma base de dados. Isto significa que, a partir de um arquivo, um conjunto de itens que constituem a resposta de uma determinada pergunta será selecionada (KITCHENHAM, 2004). Na estratégia de busca criamos: (i) os termos de buscas, tais como derivar as palavras-chaves da questão da pesquisa e identificar os sinônimos ou palavras relacionadas; (ii) o *string* de busca, agrupando os sinônimos e palavras relacionadas com o identificador *OR* e agrupando cada um dos termos com o identificador *AND*. Por exemplo: a fim de estabelecer a estratégia da busca, considerando os termos de busca que inicialmente identifica as principais palavras-chave *"Arquitetura de Referência"* e *"Descrição arquitetônica"*, é necessário também identificar os seus sinônimos. Assim, o *string* de busca poderia ser: (*"Reference Architecture" OR "Reference Model") AND ("Architectural View" OR "Architecture View" OR "Architectural Model" OR "Architecture Model" OR "Architectural Description" OR*

"Architecture Description"). Por fim, na estratégia de busca, estabelece-se as fontes de pesquisa (ou seja, bancos de dados de publicação) para encontrar os estudos primários, como exemplo: *ACM Digital Library, IEEEXplore, ISI Web of Knowledge, ScienceDirect, Scopus e SpringerLink.*

Critérios de Inclusão e Exclusão: Os critérios de **Inclusão** (CI) são usados para selecionar para a pesquisa. Assim, como exemplo, os critérios de inclusão (CI) para um estudo sobre arquitetura de referência podem ser: o estudo preliminar propõe ou usa uma abordagem para representar arquiteturas de referência.

Os critérios de **exclusão** (CE) são usados para excluir os estudos que não contribuem para responder às questões de pesquisa. Um exemplo de critério de exclusão para o caso de seleção de estudos sobre arquiteturas de referência pode ser: o estudo primário não propõe ou usa qualquer abordagem para representar arquiteturas de referência.

Extração de dados e Método de síntese: na extração dos dados deve-se: (i) projetar formulários de extração de dados obtidos dos estudos; (ii) registrar detalhes de referências e informações que tratam as questões de pesquisa e (iii) adotar um gerenciador de referências bibliográficas, exemplo: JabRef (KITCHENHAM, 2004).

Na síntese dos dados, deve-se: (i) fazer a sumarização dos resultados dos estudos incluídos; (ii) definir a síntese descritiva, síntese quantitativa (meta-análise) e (iii) apresentar os resultados em tabelas e gráficos (KITCHENHAM, 2004).

Execução: nesta fase, os estudos primários são identificados, selecionados e avaliados de acordo com o protocolo previamente estabelecido. Nessa etapa realiza-se as buscas nas diversas bases de dados de publicações, tais como, *IEEEXplorer,ACM DigitalLibrary, Web of Science e Science Direct.* Além disso, também se faz a avaliação e seleção dos trabalhos selecionados relevantes para tema.

Avaliação: nesta última fase apresentam-se os resultados analíticos da revisão sistemática. Se faz a avaliação dos trabalhos e a identificação das respostas para as questões da pesquisa.

Em uma revisão sistemática é aconselhável que sejam mostradas também as limitações do trabalho, tais como: outras fontes de pesquisas que poderiam ser selecionadas, outras palavras-chave que poderiam ser adicionadas à cadeia de pesquisa. Além disso, vale destacar que a condução da revisão sistemática não é uma tarefa trivial, devido à quantidade de documentos que precisam ser manipulados e o fato de que pode haver relevantes estudos primários escritos em outras línguas, que não o Inglês e, em geral, eles

são negligenciados.

Finalmente, deve-se concluir uma revisão sistemática fazendo um relato das conclusões e das principais contribuições do trabalho em estudo.

2.5 Considerações Finais

Este capítulo apresentou os conceitos básicos relacionados a esta tese, tais como: computação ubíqua e suas características; arquitetura de referência, especificando seus tipos e mostrando como projeta-los; linguagens de descrição de arquiteturas e a linguagem ACME, e Armani, bem como o conceito e processo de revisão sistemática. O próximo capítulo apresenta os trabalhos que estão relacionados a proposta desta tese.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta trabalhos que estão relacionados à proposta desta tese. Na Seção 3.1 são discutidos trabalhos relacionados com revisões sistemáticas. Na Seção 3.2 são discutidas propostas existentes de arquiteturas de referência para computação ubíqua ou pervasiva; e na Seção 3.3 são discutidas propostas existentes de linguagens para descrição arquitetural de sistemas ubíquos e pervasivos.

3.1 Revisão Sistemática

Durante o estudo de uma nova área de conhecimento, pesquisadores costumam realizar uma revisão bibliográfica (quase sempre uma revisão informal) para identificar publicações relacionados a um assunto específico. No entanto, este tipo de análise não utiliza uma abordagem sistemática e não oferece qualquer apoio para evitar os erros durante a seleção das publicações que serão analisados. Assim, o uso de mecanismos para resumir e fornecer uma visão geral sobre uma área ou tema de interesse se torna importante (PETERSEN et al., 2008). Em particular, a revisão sistemática tem sido amplamente investigada e adotada na *Evidence-Basead Software Engineering (EBSE)* (KITCHENHAM; DYBA; JORGENSEN, 2004). Uma revisão sistemática fornece uma avaliação abrangente e sistemática de pesquisa usando uma estratégia de busca predefinida para minimizar o viés (KITCHENHAM; CHARTERS, 2007). Em outras palavras, possibilita sistematicamente obter a revisão de literatura e é usada para resumir, avaliar e interpretar a relevância das evidências relacionadas a uma pergunta específica, área temática, ou fenômeno de interesse. Uma evidência individual (por exemplo, um estudo de caso ou estudo experimental relatado em uma publicação *paper*), que contribui para uma revisão sistemática é

chamado de estudo primário, enquanto que o resultado de uma revisão sistemática é um estudo secundário.

Revisões sistemáticas já foram aplicadas para diferentes temas de interesse, como: i) SPÍNOLA E TRAVASSOS (2012), que avaliou projetos de sistemas ubíquos para identificar quais características fundamentais de sistemas ubíquos estavam sendo atendidas por esses projetos; ii) No trabalho de GUESSI (2011), foi feita uma revisão sistemática para identificar todas as abordagens ou propostas para representar arquiteturas de referência. Como principal resultado, verificou-se que a representação de arquiteturas de referência não tem sido suficientemente explorada e ainda há perspectivas diferentes que poderiam ser investigadas, com o objetivo de promover a sua divulgação e utilização eficaz; iii) O trabalho de OLIVEIRA, OSÓRIO e NAKAGAWA (2012) apresenta uma visão detalhada e analítica sobre os sistemas, implementação, tecnologias e diretrizes de engenharia de software para robôs com base em SOA . Como principais resultados, observou-se um recente aumento no número de obras que relatam sistemas, tecnologias e ambientes de desenvolvimento de robôs baseados em SOA, além de direções para trabalhos futuros.

Vários outros trabalhos envolveram revisões sistemáticas em diferentes áreas da computação: Arquitetura de Software ((OLIVEIRA et al., 2010)(WILLIAMS; CARVER, 2010)(BOER; FARENHORST, 2008)); Teste de Software ((ENGSTROM E.; SKOGLUND, 2010); (ENDO; SIMAO, 2010)); Extração de requisitos (DAVIS et al., 2006).

Não identificamos na literatura trabalhos que investiguem elementos arquiteturais essenciais para sistemas ubíquos, como apresentamos nesta tese.

3.2 Arquitetura de Referência

Arquiteturas de referência têm desempenhado um importante papel, agregando informações sobre um determinado domínio de aplicação e contribuindo para o sucesso do desenvolvimento de sistemas inseridos nesse domínio. Dessa forma, a literatura apresenta vários trabalhos para diferentes arquiteturas de referências.

Existem várias arquiteturas de referência para diferentes domínios, tais como sistemas orientados a serviços (ARSANJANI et al., 2007) (OLIVEIRA; NAKAGAWA, 2011), sistemas embarcados (AUTOSAR, 2012) (BATORY et al., 1995) (EKLUND et al., 2005), e sistemas robóticos (BLACKAND B.; KNAPP, 2010). No entanto, para a computação ubíqua, há muito poucas propostas, e ela têm como alvo aplicações específicas, tais como ambientes inteligentes (FERNANDEZ-MONTES et al., 2009). Em contrapartida, nesta tese foi apresentada

uma arquitetura de referência genérica para o domínio da computação ubíqua. Isso significa que tal arquitetura pode ser usada, em princípio para derivar arquiteturas concretas para qualquer tipo de aplicações ou sistemas ubíquos. Nessa subseção, vamos discutir alguns trabalhos relacionados encontrados na literatura.

PCA (LIU; LI, 2006) propõe uma arquitetura dividida em duas partes: computação pervasiva orientada a redes e computação pervasiva orientada a personalidade. A computação pervasiva orientada a redes está focada em inteligência da computação distribuída, computação móvel, etc. Ela engloba quatro camadas: *middleware*, serviços de rede, segurança e sistema e hardware. A segunda parte, orientada à personalidade, trata de aplicações voltadas para o usuário, como casas inteligentes, carros inteligentes, navegação inteligente, etc. Essa segunda parte abrange quatro camadas: aplicação, *middleware* e interface de segurança, sistemas operacional embarcados e de hardware. PCA é muito descritiva para as questões de segurança e dá uma visão geral conceitual dos elementos arquiteturais envolvidos na computação pervasiva. No entanto, é restrito a um alto nível de abstração e, uma vez que é voltado para computação pervasiva, PCA não lida com questões de adaptação e mobilidade. Isso significa que a arquitetura de referência não cobre todo o propósito de sistemas ubíquos.

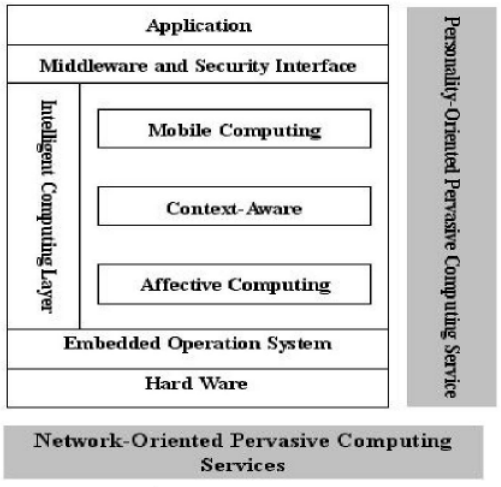

Figura 3.1: Uma Arquiteturade para Computação Pervasiva.(LIU; LI, 2006)

PSC-RM (ZHOU et al., 2009) propõe um modelo de referência para computação pervasiva, com foco em soluções baseadas em serviços *web* para computação pervasiva. Propõe-se uma estrutura em camada, enquanto que a primeira camada é responsável por descrever os serviços, interfaces homem-máquina, problemas de mobilidade, colaboração peer-to-peer, etc. Essa camada é responsável por descrever e representar os serviços básicos disponíveis que serão utilizados pela segunda camada: a camada do sistema PSC-RM. A

segunda camada engloba repositórios, gerente de contexto, multimodal HCI, uma interface (Linguagem) para descrever regras de composição de serviços, e o núcleo do modelo de referência, a composição Serviço Pervasivo. Esse elemento é responsável pela coordenação entre pares (que visa formar uma comunidade de cooperação) e colaboração de serviço (teve como objetivo reunir as funções e competências dos cooperadores), isto é, compor os serviços existentes em serviços novos. Finalmente, a terceira camada é a camada de habilitação e aperfeiçoamento. PSC-RM é um modelo de referência muito complexo, mas é focado em composição de serviços. O modelo de referência não lida com a descoberta de serviço nem de gestão de eventos, que são alguns dos principais desafios dos sistemas ubíquos. Assim, não se tornando suficiente para descrever sistemas ubíquos.

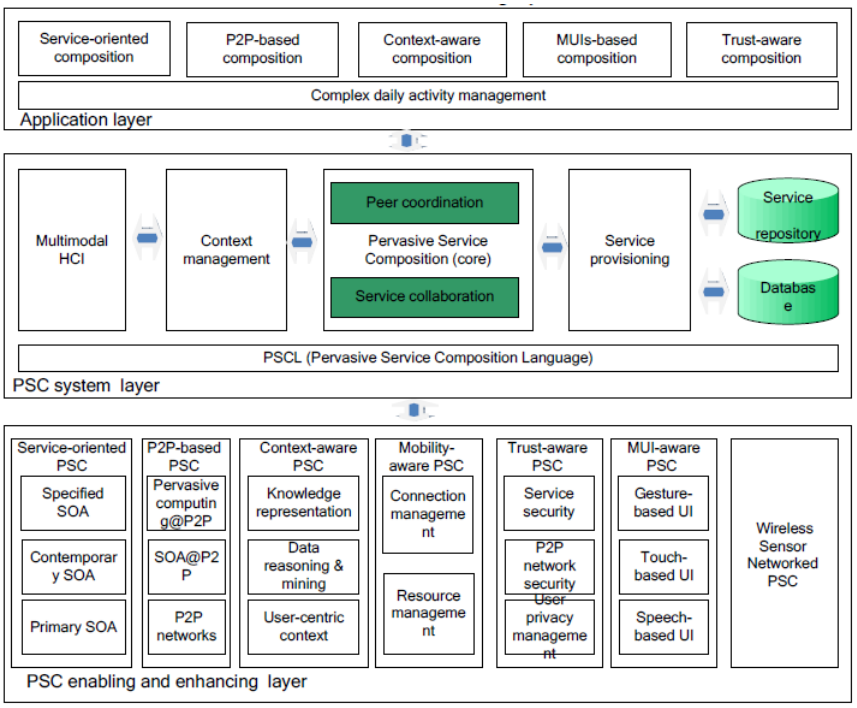

Figura 3.2: Modelo de Referência para Computação Pervasiva.(ZHOU et al., 2009)

Smart Environment Software Reference Architecture (FERNANDEZ-MONTES et al., 2009) propõe um modelo de referência para ambientes inteligentes. Esse modelo organiza ambientes inteligentes em tarefas. Essas tarefas podem ser executadas por um ou mais componentes de arquitetura e são essencialmente três: *perception*, *reasoning* e *acting*. Ver Figura 3.3.

A tarefa *perception* implementa a consciência do contexto, e envolve cinco elementos que são responsáveis pela sub-tarefas: *Collector* para coletar informações de contexto; *Verifier*, para verificar a regularidade de dados de contexto; *Repairer*, para corrigir os dados incorretos detectados pelo verificador; *Filter*, para isolar dados e *Ontologiser*, para

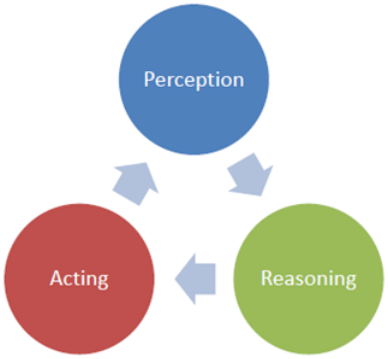

Figura 3.3: Ciclo de Automação de Processo em um *Smart Environment*.(FERNANDEZ-MONTES et al., 2009)

organizar os dados de acordo com um modelo do mundo real. Ver Figura 3.4.

Figura 3.4: Tarefa do Processo *Perception*.(FERNANDEZ-MONTES et al., 2009)

A tarefa de *Reasoning* implementa recursos de inteligência artificial e pode ser separada em várias sub-tarefas, mas estas tarefas colaboram umas com as outras com o objetivo de atingir os seguintes objetivos: aprender, raciocinar, e prever. A tarefa *Learning Agent* pode ser vista como a cola que une qualquer outra tarefa proposta para alcançar os objetivos do processo de raciocínio, tendo as seguintes sub-tarefas: *Data Mining*, para mineração de dados; *Situation Recognition*, para reconhecimento de padrões; *Predicrion*, para encontrar ações futuras a serem tomadas; *Error Detector*, para detectar decisões erradas e fazer melhorias relacionadas ao logo de todo processo. Ver Figura 3.5.

Por fim, a tarefa *Acting* é responsável por ação. Ela é composta por três sub-tarefas: *Policy Manager*, responsável pela definição e acompanhamento das políticas e define quais ações devem ser executadas e sua ordem de execução, com base nessas políticas, o *Task Scheduler*, responsável por agendar as ações definidas pelo *Policy Manager*, e *Task Runner*, responsável pela execução das ações, traduzindo as instruções para instruções de baixo

Figura 3.5: Tarefa do Processo *Reasoning*.(FERNANDEZ-MONTES et al., 2009)

nível. Ver Figura 3.6.

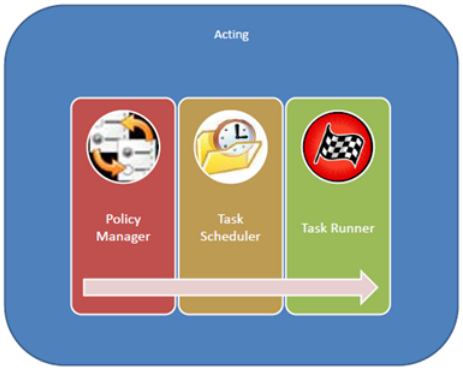

Figura 3.6: Tarefa do Processo *Acting*.(FERNANDEZ-MONTES et al., 2009)

Essas três tarefas interagem umas com as outras, em um processo cíclico, de modo a alcançar os objetivos do sistema. Embora esse trabalho seja proposto como uma arquitetura de referência para computação ubíqua, não é uma arquitetura de referência real, uma vez que não define elementos arquiteturais (componentes e interfaces) que sejam mais utilizados para obter arquitetura concreta para este domínio.

OLIVEIRA e NAKAGAWA (2011) propõem uma arquitetura de referência orientada a serviço, denominada *RefTEST-SOA (Reference Architecture for Software Testing Tools based on SOA)*, que agrega o conhecimento e a experiência de como organizar ferramentas de teste orientadas a serviço e visa a integração, a escalabilidade e o reúso providos pela SOA para essas ferramentas.

Em OLIVEIRA e NAKAGAWA (2009), o principal objetivo da identificação de métodos que abordem a avaliação arquiteturas de referência, arquiteturas orientadas a serviços e arquiteturas orientadas a aspectos. Como principal resultado, observou-se que há ainda

uma grande carência de métodos consolidados e largamente aceitos para a avaliação de tais arquiteturas. Os métodos mais conhecidos são o SAAM (KAZMAN et al., 1994) e o ATAM (KAZMAN et al., 1998), outros métodos são extensões dos supracitados: D-SAAM (GRAAF; DIJK; DEURSEN, 2005), uma adaptação do SAAM para avaliar uma arquitetura de referência para sistemas embarcados; a abordagem de GALLAGHER (2000), que estende o ATAM para avaliar uma arquitetura de referência para sistema de controle militar; a de ANGELOV (2008), que modifica o ATAM para que seja possível sua utilização em arquiteturas de referência, entre outros.

3.3 ADLs para Computação Ubíqua

No contexto de ADLs para computação ubíqua existem poucas propostas, sendo elas *ScudADL* (WU; LI, 2009) e LindaQoS (NETO; RODRIGUES; SOARES, 2004). Nesta seção, ambas serão discutidas e será feita uma breve relação entre seus elementos e os elementos de UbiACME, ADL apresentada no capítulo 6.

ScudADL (WU; LI, 2009) é uma ADL para descrição de *middlewares* adaptativos para computação ubíqua, baseada em π-*ADL* (OQUENDO, 2004) e D-ADL (CHANGYUN; GANSHENG, 2006) para representar ambos estrutura e comportamento dos elementos arquiteturais que compõem o sistema. *ScudADL* possui ênfase em reconfiguração dinâmica, e é voltada para descrição de ambientes inteligentes. A linguagem não possui abstrações de primeira ordem específicas para computação ubíqua, tais como: Elementos para representação de contexto; Representação de parâmetros de qualidade e conexões condicionais (*attachments* condicionais). Em comparação com UbiACME, a linguagem mostra-se bastante complexa por se basear em π-*calculus*. Por outro lado, *ScudADL* é capaz de representar em detalhes a estrutura e o comportamento dos elementos do sistema, enquanto UbiACME possui ênfase em descrições estruturais. Uma vez que *ScudADL* não possui abstrações para representação de contexto - por a linguagem ser baseada em expressões de reconfiguração, dificulta-se a reutilizar ou a utilização de contextos.

LindaQoS (LINguagem de Descrição de Arquitetura com QoS) (NETO; RODRIGUES; SOARES, 2004) propõe uma linguagem específica de domínio voltada para representação de parâmetros de qualidade e reconfiguração dinâmica. A linguagem dá margem à representação de contexto de forma implícita e permite a definição de estruturas de reconfiguração baseadas nessas definições. Uma vez que a ênfase da linguagem é representar parâmetros de QoS, a definição de contexto, em *LindaQoS*, mostra-se menos expressiva que em UbiACME, que possui um elemento arquitetural de primeiro nível para representar contextos.

Outro ponto negativo é a dificuldade em reusar e rastrear expressões de contextos, uma vez que essas expressões podem se encontrar espalhadas ao longo da descrição.

Ambos os trabalhos não satisfazem as características desejáveis de ADLs para computação ubíqua, que são: (i) representações para contexto como elemento de primeira ordem; (ii) mecanismo para adaptações em tempo de execução; (iii) mecanismo para reconfiguração arquitetural em tempo de execução; e (iv) representação de parâmetros de qualidade no nível arquitetural. *LindaQoS* possui as características (ii), (iii) e (iv), enquanto *ScudADL* possui as características (ii) e (iii). A caracteristica (i), cuja importância é discutida em (LOPES; FIADEIRO, 2005), não é provida por nenhuma das ADLs avaliadas.

3.4 Considerações Finais

Este capítulo apresentou vários trabalhos encontrados na literatura que tem relação com os tópicos desta tese, tais como: Revisão Sistemática; Arquitetura de Referência; PCA, uma arquitetura de referência para computação pervasiva; RM-PSC: Modelo de Referência para Composição de Serviços Pervasive; *Smart Environment Software Reference Architecture* um modelo de referência para ambientes inteligentes; *ScudADL*, uma ADL para descrição de *middlewares* adaptativos para computação ubíqua, baseada em π-*ADL*; e finalmente LindaQoS (LINguagem de Descrição de Arquitetura com QoS). No próximo capítulo será apresentado uma Revisão Sistemática sobre Elementos Arquiteturais para Sistemas Ubíquos, conduzida, desde o seu planejamento até a análise dos resultados, focando-se nos elementos arquiteturais que caracterizam os sistemas para a computação ubíqua.

Capítulo 4

Uma Revisão Sistemática sobre Elementos Arquiteturais para Sistemas Ubíquos

A revisão sistemática apresentada neste capítulo tem como objetivo identificar os principais elementos que constituem arquiteturas de sistemas ubíquos e também se há arquiteturas de referência para tal domínio. É importante destacar que arquiteturas de referência usualmente possuem elementos arquiteturais comuns de um conjunto de sistemas de software para determinado domínio. Dessa forma, no contexto desta revisão sistemática, é interessante examinar a existência de tais arquiteturas, visando identificar quais são esses elementos.

Nessa mesma linha, SPÍNOLA e TRAVASSOS (2012) apresentaram uma revisão sistemática que teve como objetivo a caracterização de projetos de softwares ubíquos. Uma das questões de pesquisa investigada refere-se às características que definem aplicações para computação ubíqua. Vale destacar que este trabalho difere-se da revisão sistemática apresentada neste capítulo porque ela visa identificar elementos arquiteturais comumente encontrados em sistemas ubíquos, além também de investigar a existência de arquiteturas de referência.

Este capítulo está organizado da seguinte forma: a Seção 4.1 apresenta a revisão sistemática conduzida, desde o seu planejamento até a análise dos resultados, focando-se nos elementos arquiteturais que caracterizam os sistemas para a computação ubíqua e que foram identificados nos diversos trabalhos analisados. A Seção 4.2 contém uma discussão sobre os dados coletados. A Seção 4.3 apresenta as ameaças à validade dessa revisão sistemática. A Seção 4.4 apresenta as conclusões do capítulo.

4.1 Aplicação da revisão sistemática

A revisão sistemática foi conduzida no contexto de arquiteturas de software para computação ubíqua, objetivando avaliar estudos relevantes para a área até março de 2013. Para conduzir essa revisão sistemática, dividiu-se o processo em três etapas, conforme ilustrado na Figura 4.1: Planejamento, Execução e Avaliação.

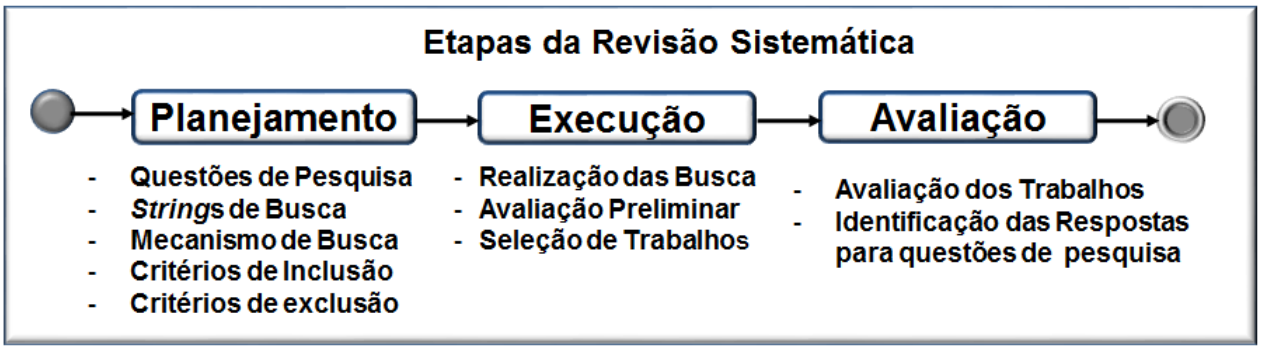

Figura 4.1: Etapas da Revisão Sistemática.

Na etapa de Planejamento foram definidos os critérios de busca e critérios de inclusão e exclusão que foram utilizados para coletar trabalhos relacionados à computação ubíqua ou pervasiva. Essa etapa também foi responsável por definir o que se deseja extrair dos trabalhos encontrados.

A etapa de Execução consistiu na execução da revisão sistemática, na qual foi realizada a busca pelos estudos primários (ou seja, publicações em conferências, periódicos, teses, entre outros), utilizando-se o planejamento realizado na etapa anterior. Nessa etapa também foram aplicados os critérios de inclusão e exclusão, a fim de filtrar os resultados que eram relevantes para essa revisão.

Por fim, na etapa de Avaliação foi realizada a avaliação dos resultados encontrados, extraindo dados para formular respostas para as questões de pesquisa.

4.1.1 Planejamento

Nessa etapa da revisão sistemática são definidas: (1) questões de pesquisa, (2) estratégias de busca e (3) critérios de inclusão e exclusão.

1. **Questões de pesquisa:** Visando encontrar todos os estudos primários que apresentam elementos arquiteturais comuns de sistemas ubíquos, as seguintes Questões de Pesquisa (QP) foram definidas:

(a) **QP1:** Quais são as arquiteturas de referência para sistemas ubíquos?
Observação: Essa questão de pesquisa foi formulada visando encontrar arquiteturas de referência de sistemas ubíquos.

(b) **QP2:** Quais são os elementos arquiteturais comuns a sistemas ubíquos?
Observação: Essa questão foi definida de modo a complementar a resposta encontradas na QP1 e identificar um conjunto de elementos arquiteturais comuns aos sistemas ubíquos.

2. **Estratégia de busca:** Visando estabelecer a estratégia de busca dos estudos primários, a partir das questões QP1 e QP2, foram inicialmente definidas as palavras-chave: *"Reference Architecture"* e *"Ubiquitous Computing"*. Também foram definidos sinônimos para essas palavras-chave ou contextos semelhantes: *"Reference Architecture"* pode ser referenciado como *"Reference Model"* e está diretamente relacionada à *"Software Architecture"* ou *"Architectural Model"*. Adicionalmente, *"Ubiquitous Computing"* está relacionada à *"Pervasive Computing"*. Foram também considerados os *middlewares* para computação ubíqua na busca, através das palavras-chave *"ubiquitous middleware architecture"* e *"pervasive middleware architecture"*. Essa inclusão teve dois objetivos: (i) obter uma visão geral dos sistemas existentes, visto que os *middlewares* são projetados para atender a uma grande variedade de aplicações ubíquas/pervasivas; e (ii) possibilitar a identificação dos elementos que compõem esses *middlewares*, que expressam componentes importantes para sistemas ubíquos. Dessa forma, foi estabelecida a seguinte *string* de busca: ((*"Reference Architecture"* OR *"Reference Model"* OR *"Software Architecture"* OR *"Architecture Model"*) AND (*"Ubiquitous Computing"* OR *"Pervasive Computing"* OR *"ubiquitous middleware"* OR *"pervasible middleware"*)). Essa *string* foi adaptada nas seguintes bases de dados de publicações: *IEEEXplorer , ACM Digital Library , Web of Science e Science Direct*, adicionalmente, a *string* foi utilizada para cada uma dessas bases de modo a realizar uma busca direcionada sobre os campos título, resumo e palavras-chaves. Optou-se por avaliar apenas os estudos publicados em inglês, uma vez que é a língua comumente utilizada para publicações de trabalhos na área de Computação.

   O processo de revisão foi planejado da seguinte forma. A busca deve ser realizada em bibliotecas digitais que incluem os principais veículos cujos trabalhos podem estar publicados. Em seguida, deve ser realizada a leitura do título, resumo e palavras-chave dos trabalhos encontrados, para então definir quais trabalhos merecem a leitura do texto completo. Após a leitura completa, as respostas das questões de pesquisa devem ser formuladas.

3. **Critérios de inclusão e exclusão:** para avaliar e selecionar cada estudo encontrado, definiu-se um conjunto de critérios de inclusão e exclusão. Esses critérios foram utilizados após a etapa de busca para definir a relevância de um determinado estudo. Os critérios de inclusão (CI) foram utilizados para incluir estudos relevantes nessa revisão sistemática, sendo eles:

   (a) **CI1:** O trabalho propõe, utiliza ou avalia uma arquitetura de referência para sistemas ubíquos.

   (b) **CI2:** O trabalho apresenta um *middleware* para computação ubíqua, apresentando explicitamente sua arquitetura.

   Critérios de exclusão (CE) foram definidos para excluir dessa revisão estudos que não contribuem para responder as QP1 e QP2, sendo eles:

   (a) **CE1:** O trabalho não está relacionado a sistemas ubíquos ou pervasivos;

   (b) **CE2:** O estudo não está em inglês;

   (c) **CE3:** O estudo não apresenta resumo ou o texto completo não pode ser acessado;

   (d) **CE4:** O estudo consiste de uma compilação de trabalhos de conferências ou *workshops*, por exemplo;

   (e) **CE5:** O estudo define uma arquitetura de baixo nível, no sentido de descrever elementos de hardware ou operacionais.

   Definiu-se, finalmente, que um trabalho relevante para essa revisão sistemática é aquele que atende a pelo menos um critério de inclusão e não satisfaz nenhum critério de exclusão.

Os critérios de inclusão e exclusão foram aplicados depois da leitura completa dos trabalhos selecionados pela leitura de títulos e resumos, cada trabalho selecionado passou pelos critérios de inclusão e exclusão e o trabalho que atendesse a pelo menos um critério de inclusão e não satisfizesse nenhum critério de exclusão seria relevante para a revisão sistemática.

4.1.2 Resultados da Execução

Após a realização das buscas, obteve-se os resultados sintetizados na Figura 4.2. Este gráfico mostra a quantidade de artigos encontrados, artigos filtrados e de artigos seleciona-

dos. Os artigos encontrados correspondem ao número de artigos retornados pela busca automática, os artigos filtrados foram avaliados, ou seja, tiveram os títulos, palavras-chaves e resumo lidos. Já os artigos selecionados são os trabalhos cujos resumos e palavras-chaves mostraram-se interessantes para essa revisão, sendo assim selecionados para a realização da leitura completa.

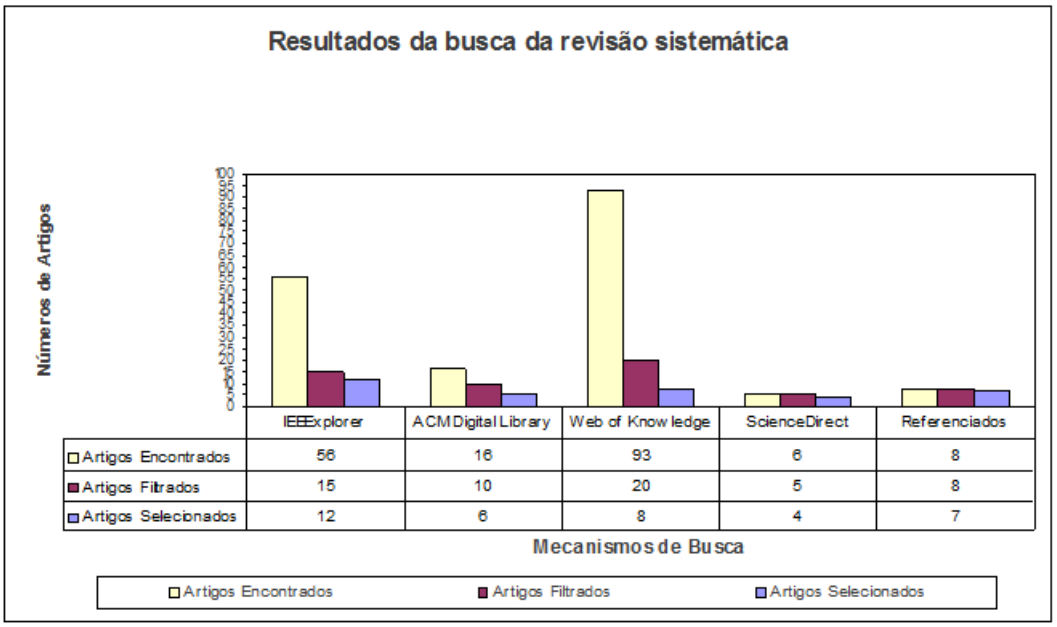

Figura 4.2: Resultados da Busca.

Conforme ilustra o Gráfico 4.2:

(i) dos 56 resultados encontrados no mecanismo de busca *IEEExplorer*, 15 foram filtrados e 12 foram selecionados para a segunda etapa.

(ii) dos 16 resultados encontrados no mecanismo de busca no *ACM Digital Library*, 10 foram filtrados e 6 foram selecionados para a segunda etapa.

(iii) dos 93 resultados encontrados no mecanismo de busca *Web of Knowledge*, 20 foram filtrados e 8 foram selecionados para a segunda etapa.

(iv) dos 6 resultados encontrados no mecanismo de busca *ScienceDirect*, 5 foram filtrados e 4 foram selecionados.

Adicionalmente, 8 novos trabalhos foram encontrados a partir da avaliação das referências dos artigos selecionados em primeira instância e 7 deles foram selecionados, totalizando-se 37 trabalhos primários selecionados. Após a leitura completa de cada trabalho e a aplicação dos critérios de inclusão e exclusão definidos no planejamento da revisão sistemática, 13 trabalhos foram considerados como relevantes para esse estudo,

conforme listados na Tabela 4.1.

Estudo	Autor	Ano
E1	Jiehan Zhou et al (Zhou 2009)	2009
E2	Yi Liu, Freng Li (Liu 2006)	2006
E3	Tao Xu, Bertrand David, René Chalon, Yun Zhou (Xu 2011)	2011
E4	Shriram. R , Vijayan Sugumaran (Shriram 2007)	2007
E5	Seung Wok Han, Yeo Bong Yoon and Hee Yong Youn (Han 2004)	2004
E6	Saeed, A. and Waheed, T. (Saeed and Waheed 2010)	2010
E7	Chang-Woo Song et al (Song 2013)	2013
E8	Eugster, Patrick Th.; Garbinato, Benoît; Holzer, Adrian (Eugster 2009)	2009
E9	Román, M. et al. (Román 2002)	2002
E10	Spínola, R. and Travassos, G. (Spínola and Travassos 2012)	2012
E11	Raychoudhury, V., Cão, J., Kumar, M., Zhaung, D. (RAYCHOUDHURY et al., 2013)	2013
E12	DA, K., Dalmau, M., Roose, P. (DA; DALMAU; ROOSE, 2012)	2012
E13	Fernandez-Montes, A., Ortega, J. A., Alvarez, J.A. (FERNANDEZ-MONTES et al., 2009).	2009

Tabela 4.1: Lista de Artigos Selecionados.

Dentre esses trabalhos, destacam-se os estudos E6, E8 e E11 que apresentam *surveys* sobre *middlewares* para computação ubíqua e citam, entre outras, arquiteturas precursoras, tais como Gaia (ROMÁN et al., 2002) e Homeros (HAN et al., 2004). Entretanto, como esses *surveys* possuem objetivos diferentes deste trabalho, foram utilizados apenas como fonte de pesquisa para novos *middlewares*. Além disso, o trabalho E10 apresenta uma revisão sistemática também sobre computação ubíqua, mas tem seu foco na caracterização de projetos de computação ubíqua. Vale ressaltar que o presente estudo também é interessante para esta revisão sistemática, no entanto, é diferente, pois o objetivo desta revisão sistemática é identificar os elementos arquiteturais comumente encontrados em sistemas ubíquos, bem como arquiteturas de referência existentes.

4.1.3 Resultados da Avaliação

Foram encontrado quatro estudos (E1, E2, E11 e E13) que apresentam duas arquiteturas de referência para sistemas ubíquos ou pervasivos: (ZHOU et al., 2009), (LIU; LI, 2006), (RAYCHOUDHURY et al., 2013) e (FERNANDEZ-MONTES et al., 2009). A arquitetura proposta por *ZHOU* (2009) é voltada para composição de serviços em sistemas pervasivos, enquanto a arquitetura apresentada por *LIU* (2006) foi definida de forma genérica. Apesar de definida para computação pervasiva, a arquitetura de *Liu* (2006) apresenta um elemento de mobilidade, típico de sistemas ubíquos. A arquitetura proposta por *RAYCHOUDHURY* (2013) foi construída para fundamentar comparações entre sistemas pervasivos existentes, não dando suporte à mobilidade e descreve uma estrutura multi-nível que mescla elementos de alto nível de abstração, como raciocinadores, com elementos de baixo nível de abstração como protocolos de rede. Por fim, a arquitetura proposta por FERNANDEZ-MONTES (2009) é voltada para construção de aplicações para ambientes inteligentes, com

Elemento	Descrição	Estudos
Sensor	Responsável por prover informação de contexto	E1, E3, E7, E8, E9, E11, E12, E13, E14, E15, E16
Atuador	Responsável por alterar o ambiente, fornecendo feedback para o usuário	E3, E8, E14, E15
Serviço de Contexto	Serviço utilizado para recuperar as informações de contexto oriundas dos sensores, eventualmente coordenando mais de um sensor	E1, E3, E4, E7, E9, E11, E12, E14, E15, E16
Serviço de Atuação	Serviço utilizado para fornecer feedback para o usuário, eventualmente coordenando mais de um atuador	E3, E8, E14, E15
Repositório de Contexto	Repositório onde informações de contexto e Qualidade de Contexto (Quality of Context - QoC) são armazenadas	E1, E2, E3, E4, E5, E7, E9, E11, E12, E13, E14, E15, E16
Módulo de Eventos	Mecanismo para dar suporte ao monitoramento assíncrono de serviços	E1, E5, E7, E9,
Módulo de Raciocínio	Mecanismo que permite produzir nova informação de contexto a partir das informações existentes	E1, E2, E3, E7, E8, E9, E13, E16
Mecanismo de Adaptação	Mecanismo para alterar o comportamento do sistema de acordo com um determinado conjuntos de regras.	E1, E5, E9, E12
Mecanismo de Acoplamento e Mobilidade	Abstrai a noção de ambiente, tornando o sistema funcional em vários ambientes. Utiliza mecanismos de rastreamento, busca de serviço e comunicação móvel.	E2, E4, E12, E13, E16
Módulo de Composição	Compõe informações de contexto, produzindo informações de maior nível de abstração.	E2, E3, E7, E8, E9, E12, E15, E16
Módulo de Segurança	Módulo responsável por implementar regras de proteção tais como mecanismos de autenticação, restrição de acessos e validação de serviços	E2, E5, E9, E11, E13

Tabela 4.2: Elementos Comuns em Sistemas Ubíquos.

um enfoque maior em definição de requisitos a elementos arquiteturais. Esses trabalhos contribuíram então para responder a QP1, sobre arquiteturas de referência para sistemas ubíquos. Utilizando essas quatro arquiteturas em conjunto com os demais trabalhos sobre *middlewares* para computação ubíqua (ou seja, os estudos E3, E4, E5, E7, E8, E9 e E12), é possível identificar os elementos comuns e essenciais para arquiteturas de sistemas ubíquos, buscando responder assim a QP2.

A Tabela 4.2 descreve os elementos comuns em sistemas ubíquos. A primeira coluna nomeia o elemento, a segunda contém uma breve descrição do elemento e a terceira lista os estudos primários que apresentaram um conceito igual ou semelhante ao elemento em questão. Portanto, pode-se afirmar que quando se trata do desenvolvimento de sistemas ubíquos, esse conjunto de 10 elementos poderiam estar presente, uma vez que são comumente encontrados nesses sistemas. Além disso, pode-se inclusive afirmar que são elementos essenciais em arquiteturas de sistemas ubíquos.

4.2 Discussão

Com relação aos trabalhos relacionados, a revisão sistemática apresentada no trabalho de Spínola e Travassos (2012) teve como objetivo caracterizar um projeto de software de sistema ubíquo e entender como esse domínio afeta o ciclo de vida de desenvolvimento de software. Além da revisão sistemática, o trabalho também identificou uma lista de 10 características de sistemas ubíquos e contrastou essa lista com 26 diferentes aplicações ubíquas. Em termos de características, o trabalho cita as características apresentadas na

Características	Elemento	Estudos
Onipresença de serviços	Mecanismo de acoplamento e mobilidade	E2, E4
Invisibilidade	Sensor	E1, E2, E7, E11, E12
	Atuador	E3, E8
	Serviço de contexto	E1, E2, E7, E11, E12
	Serviço de atuação	E3, E8
Sensibilidade ao Contexto	Sensor	E1-E3, E7-E9, E11, E12
	Serviço de contexto	E1, E2, E7, E11, E12
	Repositório de Contexto	E1-E3, E7-E9, E11, E13
	Módulo de raciocínio	E2, E3, E8, E9, E11-E13
	Módulo de Agregação ou Composição	E2, E3, E8, E9, E11-E13
Comportamento adaptativo	Serviço de contexto	E1, E2, E7, E11, E12
	Módulo de eventos	E5, E7, E9, E11
	Mecanismo de Adaptação	E1, E5, E9, E13
Captura de experiência	Módulo de raciocínio	E4, E11, E12
Descoberta de serviços	Módulo de eventos	E1, E9
Composição de função	Módulo de raciocínio	E2, E3, E8, E9, E11, E12
	Módulo de Agregação ou Composição	E8, E9, E11, E13
Interoperabilidade espontânea	Mecanismo de acoplamento e mobilidade	E2, E4
Heterogeneidade de dispositivos	Sensor	E8, E9
Tolerância a falhas	Mecanismo de acoplamento e mobilidade	E4
	Módulo de eventos	E5, E9
	Mecanismo de Adaptação	E1, E5, E9
	Módulo de Raciocínio	E12
	Serviço de contexto	E12
	Módulo de Segurança	E11

Tabela 4.3: Características de Projetos de Computação Ubíqua Associados aos Elementos Arquiteturais Comuns de Sistemas Ubíquos.

Tabela 4.3. Por meio dessa tabela, observa-se que o conjunto de elementos arquiteturais comuns encontrados pela revisão sistemática apresentada neste trabalho podem ser capazes de atender às características citadas pela revisão sistemática de SPÍNOLA AND TRAVASSOS (2012). Essa tabela lista também os estudos que citam algum elemento que agrega uma dada característica.

O estabelecimento da relação entre as características e os elementos foi pautada em uma análise cuidadosa da literatura desse domínio, focando-se nas características e no papel de cada elemento arquitetural identificado na revisão sistemática. A seguir, discute-se como cada característica associa-se aos elementos, conforme apresentado na Tabela 4.3. A característica **Onipresença de Serviço** pode ser atendida pelo Mecanismo de Acoplamento e Mobilidade, uma vez que tal mecanismo emprega protocolos de comunicação móvel que permitem acesso aos serviços em qualquer lugar e a qualquer momento. A característica **Invisibilidade** está relacionada: (i) ao elemento Sensor que capta informações de contexto do ambiente sem exigir ordem explícita do usuário; (ii) ao elemento Atuador que repassa reações do sistema ao ambiente; (iii) aos mecanismos de Serviço de Contexto e Serviço de Atuação que são os elementos arquiteturais que permitem que os sensores e atuadores possam ser acessados. **Sensibilidade ao contexto** é uma característica fundamental de qualquer sistema ubíquo. Sensor e Serviço de Contexto estão

diretamente relacionados a essa característica, permitindo a identificação do contexto e a realização de operações de acordo com o contexto corrente. O Repositório de Contexto é responsável pelo armazenamento das informações. O Módulo de Raciocínio realiza inferências sobre a informação de contexto e pode produzir novas informações. O Módulo de Agregação Composição realiza a composição de informações de contexto. A característica **Comportamento Adaptativo** define que o sistema deve se adaptar ao ambiente, oferecendo serviços de acordo com o contexto. O Serviço de Contexto é essencial para identificação do contexto enquanto o Módulo de Eventos tem a responsabilidade de gerar o evento de mudança de contexto para que a adaptação seja realizada. Essa característica pode também estar atribuída ao Repositório de Contexto, como em E5. Por fim, o Mecanismo de Adaptação realiza a adaptação necessária para o novo contexto. A característica **Captura de Experiência** consiste na captura e armazenamento de informações de aprendizado para uso futuro. Naturalmente está associada ao Módulo de Raciocínio, que emprega mecanismos de aprendizado e outros recursos da inteligência artificial. Esse módulo possui uma atribuição semelhante ao que em alguns trabalhos, como no estudo E8, aparece como Módulo de Agregação ou Composição. A diferença existente entre as definições desses módulos reside no fato de que o Módulo de Raciocínio é capaz de produzir nova informação de contexto utilizando mecanismos de inteligência, enquanto o Módulo de Agregação ou Composição apenas agrupa ou compõe a informação de contexto. Na maioria dos trabalhos, entretanto, esses módulos são integrados. A **Descoberta de Serviços** é atendida, na maioria dos trabalhos, pelo Módulo de Eventos, que é pró-ativo em relação aos serviços, realizando o monitoramento e a descoberta de serviços presentes no ambiente, disponibilizando-os por meio de um mecanismo *publish-subscribe*. Entretanto, esse comportamento também pode ser agregado ao Repositório de Contexto, como acontece no trabalho descrito em E5. A **Composição de Função** é a característica que determina a habilidade do sistema em disponibilizar novos serviços ao usuário, baseado nos serviços existentes. O Módulo de Raciocínio está relacionado a essa característica, uma vez que esse módulo deve ser capaz de identificar os serviços que serão utilizados como serviços básicos (E2, E3, E8, E9, E12) e compô-los segundo alguma regra de negócio. O Módulo de Agregação ou Composição, em alguns trabalhos (E8, E9, E11), é usado para realizar a composição. O Módulo de Raciocínio, adicionalmente, pode inferir novas informações de contexto para disponibilizar como serviço adicional. Os novos serviços que podem ser oferecidos, entretanto, variam entre as aplicações. A **Interoperabilidade Espontânea** é a capacidade do sistema de usar diversos elementos sem a necessidade de intervenção externa. Essa característica é atendida pelo Mecanismo de Acoplamento e Mobilidade,

uma vez que esse é o elemento responsável por protocolos de computação móvel e por tratar, em alto nível, trocas de ambiente (E2, E4). A característica **Heterogeneidade de Dispositivos** define que os elementos distintos devem ser acessados de forma uniforme. Os trabalhos E8 e E9 discutem o papel dos sensores na disponibilização das informações advindas de fontes heterogêneas, bem como do Módulo de Eventos para monitorar diferentes serviços de forma transparente para os usuários. Em relação à característica de **Tolerância a falhas**, o Mecanismo de Acoplamento e Mobilidade está diretamente ligado aos dispositivos móveis utilizados pelo usuário para acesso ao sistema. Por isso, esse mecanismo deve ser capaz de tratar os problemas mais comuns relacionados à computação móvel, como instabilidades de conexão e oscilações no fluxo de dados (como mostrado em E4). O Módulo de Eventos, por sua vez, pode disparar eventos diversos, tais como surgimento de novos serviços e erros ou falhas em qualquer serviço. As falhas poderão ser tratadas pelo Mecanismo de Adaptação. Em E12, a responsabilidade de verificação de falhas é difusa, em que vários elementos detectam e tratam seus próprios comportamentos inadequados. Em suma, observa-se que os elementos arquiteturais comuns identificados desta revisão sistemática atendem às características de sistemas ubíquos identificadas por Spinola e Travassos (SPÍNOLA; TRAVASSOS, 2012).

Vale ainda ressaltar que, apesar de apenas dois estudos (estudos E2 e E4) terem explicitado o Módulo de Acoplamento e Mobilidade, identificou-se que um elemento desse tipo é fundamental para sistemas ubíquos, uma vez que esses sistemas possuem essencialmente um elemento de mobilidade, que permite que o sistema seja acessível em qualquer lugar, sendo essencial para atender às características supracitadas. Os estudos E3 e E7 apresentam um mecanismo de *query* para recuperar informações de contexto armazenadas no Repositório de Contexto. Entretanto, optou-se por não inserir esse elemento explicitamente, visto que se observou que é um elemento implementado em conjunto com o próprio Repositório de Contexto, por ser altamente dependente da forma como esse repositório armazena as informações de contexto. Outros elementos de baixo nível de abstração ou bastante especializados também não foram considerados como elementos arquiteturais comuns. Por exemplo, os elementos Sistema Operacional e Protocolo de Rede não foram considerados, uma vez que apenas os trabalhos que definiam uma arquitetura de mais baixo nível de abstração citavam explicitamente tais elementos.

4.3 Ameaças à Validade

Uma importante ameaça à validade dessa revisão sistemática refere-se à completude desse estudo, ou seja, se de fato todos os artigos da literatura relacionado ao tema foram incluídos. Esse problema pode ter ocorrido pelo fato de alguns artigos relevantes não terem sido encontrados pelos mecanismos de busca, seja pela inadequação das *string* de buscas, seja pelas limitações técnicas de cada um dos mecanismos. Uma outra ameaça refere-se à validade da síntese dos resultados e conclusões apresentadas na etapa de avaliação. Buscou-se minimizar esse problema, adotando-se uma abordagem de revisão dupla em cada artigo, realizada pelos dois primeiros autores do artigo sobre Revisão sistemática publicado no ICSEA-2013. (MACHADO et al., 2013) Isso reduziu tendências específicas ou interpretações individuais de um revisor. As conclusões também foram validadas por mais de um autor. Essas estratégias garantiram que o conjunto de elementos arquiteturais encontrados cobre as necessidades essenciais de uma arquitetura para sistemas ubíquos. pelos dois primeiros autores do artigo sobre Revisão sistemática publicado no ICSEA-2013.

4.4 Considerações Finais

Este capítulo apresentou uma revisão sistemática da literatura sobre arquiteturas de referência e *middleware* para computação ubíqua, com o objetivo de sistematizar o conhecimento sobre arquiteturas de referências e dos elementos arquiteturais comuns em trabalhos da área. Para essa finalidade, foram realizadas buscas por arquiteturas de referência para sistemas ubíquos ou pervasivos e por *middlewares* para computação ubíqua, a fim de identificar os elementos comuns aos sistemas ubíquos em geral. Como resultado da revisão sistemática foram elencados os elementos essenciais de sistemas ubíquos identificados nos trabalhos investigados. A revisão sistemática também relacionou os elementos identificados com as características essenciais estabelecidas em outro trabalho (SPÍNOLA; TRAVASSOS, 2012). Esse relacionamento é importante para se verificar que os elementos que relacionamos atendem às características essenciais de sistemas ubíquos.

Uma vez que a revisão sistemática não forneceu como resultado nenhuma arquitetura de referência que possua todos os elementos identificados, no Capítulo 5 apresentamos uma Arquitetura de Referência RA-Ubi, incluindo os elementos arquiteturais resultantes deste trabalho. Com isso, pretende-se que essa arquitetura possa efetivamente contribuir para o desenvolvimento de sistemas ubíquos que têm se tornado cada vez mais importantes, senão essenciais para a sociedade atual.

Capítulo 5

Arquitetura de Referência - RA-Ubi

Como já mencionado anteriormente, há muitos desafios na computação ubíqua (KUMAR, 2009): (i) lidar com os vários tipos de eventos, como eventos de aplicativos, mudanças no ambiente, eventos de intercâmbio de dados e eventos específicos de domínio; (ii) adaptar o sistema em tempo de execução para apoiar descoberta e localização do serviço de detecção; (iii) integrar vários tipos de elementos computacionais, tais como telefones inteligentes, sensores e atuadores; e (iv) gerenciar suas comunicações, incluindo questões de mobilidade e segurança. As soluções para esses desafios não são triviais e podem envolver vários elementos de cooperação para apoiar a operação do sistema.

Essa natureza heterogênea e multifacetada, como discutido no primeiro parágrafo desta seção, torna-se muito difícil projetar e implementar sistemas ubíquos e aumenta-se o custo de construí-los. A fim de organizar sistematicamente os principais blocos de construção de um sistema ubíquo, suas responsabilidades e suas interações, fornecendo um entendimento claro e comum das arquiteturas desse domínio, uma arquitetura de referência é necessária (RA) (NAKAGAWA; ANTONINO; BECKER, 2011).

Este capítulo, apresenta RA-Ubi (MACHADO et al., 2014), uma arquitetura de referência para sistemas ubíquos. Essa arquitetura foi construída de forma sistemática, seguindo o processo de ProSA-RA (NAKAGAWA; ANTONINO; BECKER, 2011) (NAKAGAWA et al., 2009) para o estabelecimento de novas arquiteturas de referência. Esse processo define quatro etapas: (i) identificação de fontes de informação; (ii) de elicitação de requisitos; (iii) projeto de arquitetura de referência e (iv) avaliação de arquitetura de referência. As duas primeiras etapas baseiam-se na revisão sistemática anterior (MACHADO et al., 2013), que identificou na literatura existente os elementos arquiteturais essenciais de sistemas ubíquos e definidos os requisitos que são a base do projeto do RA-Ubi. No âmbito de

verificar a validade e reutilização de RA-Ubi, foram selecionados seis sistemas ubíquos relevantes existentes e seus elementos foram comparados com os elementos RA-Ubi. Embora RA-Ubi tenha sido projetada com base na literatura existente, os seis projetos usados para validá-la não foram utilizados para o projeto da arquitetura de referência.

Este capítulo está estruturado da seguinte forma: A Seção 5.1 apresenta RA-Ubi, incluindo a identificação das fontes de informação (Subseção 5.1.1), o estabelecimento dos requisitos arquiteturais (Subseção 5.1.2), o Projeto da Arquitetura de Referência (Seção 5.1.3), e finaliza com a avaliação da Arquitetura de Referência (Subseção 5.1.4).

5.1 RA-Ubi

A **RA-Ubi** é uma arquitetura de referência que agrega conhecimento do domínio de computação ubíqua, fazendo a identificação de soluções abstratas para seus problemas e promovendo a reutilização do conhecimento para o desenvolvimento de suas aplicações. A arquitetura de referência foi estabelecida com base no **ProSA-RA** (NAKAGAWA; ANTONINO; BECKER, 2011) (NAKAGAWA et al., 2009), um processo que visa a construção, representação e avaliação de arquiteturas de referência. O processo é dividido em quatros passos, conforme mostra a Figura 2.3: (1) seleção e investigação das fontes de informação **(passo RA-1)**; (2) identificação de requisitos **(passo RA-2)**; (3) descrição arquitetural da arquitetura de referência **(passo RA-3)** e (4) avaliação dessa arquitetura **(passo RA-4)**.

5.1.1 Passo RA-1 - Identificação das Fontes de Informação

Refere-se à investigação e seleção de fontes de informação com o propósito de levantar informações sobre o domínio para qual a arquitetura de referência está sendo criada. Para o RA-Ubi, as fontes de informação dividem-se em três grupos: (i) arquiteturas de referência para sistemas ubíquos; (ii) arquiteturas de *middlewares* para computação ubíqua; e (iii) arquiteturas de aplicações para computação ubíqua. Os grupos (i) e (ii) foram obtidos por meio de uma revisão sistemática no contexto de arquitetura para computação ubíqua. Já o grupo (iii) constitui-se de aplicações e ferramentas para computação ubíqua encontradas por meio de revisões bibliográficas.

Em virtude do resumido número de trabalhos encontrados na revisão sistemática, mostrados na tabela 4.1 e da falta de tempo para realizar outra revisão sistemática incluído trabalhos de arquitetura de aplicações para computação ubíqua, decidimos fazer uma

revisão bibliográfica adicional para suprir essa carência de trabalhos na área da pesquisa.

Na revisão sistemática, apresentada no Capítulo 4, executada a fim de buscar trabalhos dos grupos (i) e (ii), foram encontrados 13 trabalhos relevantes. Essa revisão sistemática encontrou duas arquiteturas de referência para sistemas pervasivos (ZHOU et al., 2009),(LIU; LI, 2006). O trabalho de LIU e LI (2006) possui elementos típicos de computação ubíqua, apesar de direcionado para computação pervasiva. Também foram encontrados arquiteturas de *middlewares* para computação ubíqua, tais como as apresentadas em (XU et al., 2011) (SUGUMARAN et al., 2007) (HAN et al., 2004) (SAEED; WAHEED, 2010). Adicionalmente, os estudos dos trabalhos relacionados aos artigos supracitados levou a mais alguns trabalhos, como (SONG et al., 2013) (EUGSTER; GARBINATO; HOLZER, 2009) (ROMÁN et al., 2002). A Tabela 4.1 lista esses trabalhos relevantes.

A revisão bibliográfica que buscou os trabalhos referentes ao grupo (iii) encontrou vários sistemas e ferramentas para computação ubíqua, que serão utilizados principalmente no passo RA-4, para a avaliação da arquitetura de referência proposta. Dentre os trabalhos encontrados, os considerados mais relevantes estão enumerados na Tabela 5.1.

Estudo	**Autor**	**Projeto**	**Ano**
E14	Want, R. et al.	Active Badge	1992
E15	Korpipää, P. et al.	CMF	2003
E16	Chen, H. L.	CoBrA	2004
E17	Hofer, T. et al.	Hydrogen	2002
E18	Bardram, J. E.	JCAF	2005
E19	Gu, T. et al.	SOCAM	2004

Tabela 5.1: Lista de Artigos que Compõem o Grupo (iii).

Os trabalhos listados pela Tabela 5.1 compõem o grupo (iii) e são trabalhos que discutem as aplicações e as ferramentas para computação ubíqua. ***Active Badge*** é um projeto de localização, desenvolvido pela *Olivetti Research Ltda*, e consiste de um sistema de localização baseado em crachás (WANT et al., 1992). O ***CMF***, é um trabalho pioneiro da área que propõe um *framework* de gerenciamento de contexto para permitir o raciocínio semântico de contexto em tempo real e até mesmo na presença de ruído, incerteza e rápida variação de contexto (KORPIPAA et al., 2003). Outro trabalho é o ***CoBrA*** (*An Intelligent Broker Architecture for Pervasive Context-Aware systems*) que utiliza uma arquitetura baseada em *BROKER* de apoio ao desenvolvimento de aplicações sensíveis ao contexto em um espaço inteligente (CHEN, 2004). Já o ***Hydrogen*** apresenta uma arquitetura de uma estrutura para sistemas sensíveis ao contexto (HOFER et al., 2002). O ***JCAF (Java Context Awareness Framework)*** para o desenvolvimento de aplicações sensíveis ao

contexto, implementado em JAVA (BARDRAM, 2005). Por fim, o ***SOCAM*** possui uma arquitetura de um serviço orientado *middleware* sensível ao contexto para a construção e prototipagem rápida de serviços móveis sensíveis ao contexto para carros inteligente (GU et al., 2004). Mais detalhes sobre as arquiteturas dos trabalhos supracitados serão descritos no passo RA-4.

5.1.2 Passo RA-2 - Estabelecimento dos Requisitos Arquiteturais

Refere-se à identificação dos requisitos da arquitetura de referência com base nas fontes de informações. Para isso, foram usadas as características apresentadas pelo trabalho de (SPÍNOLA; TRAVASSOS, 2012) como requisitos de sistemas ubíquos. A Tabela 5.2 apresenta as características que serão utilizadas como requisitos funcionais, descrevendo brevemente seu significado.

Características	**Descrição**
Onipresença de Serviço	O sistema deve permitir ao usuários mover-se com a sensação de levar os serviços de computação com ele.
Invisibilidade	O sistema deve evitar, do ponto de vista do usuário, a sensação de uso explícito de um computador
Sensibilidade ao contexto	O sistema de ser capaz de coletar informações do ambiente em que ele está sendo usado.
Comportamento adaptativo	O sistema deve ser capaz de, dinamicamente, adaptar os serviços oferecidos de acordo com o ambiente em que ele está sendo usado, respeitando suas limitações
Captura de Experiência	O sistema deve ser capaz de capturar e registrar experiências para uso posterior.
Descoberta de serviços	O sistema deve descobrir serviços de acordo com o ambiente em que ele está sendo usado.
Composição	O sistema deve possuir a capacidade de compor serviços, com base em serviços básicos, para criar um serviço solicitado pelo usuário.
Interoperabilidade espontânea	O sistema deve ter a capacidade de mudar suas funcionalidades durante o seu funcionamento de acordo com o ambiente em que se encontra.
Heterogeneidade de dispositivos	O sistema deve ser capaz de utilizar diversos dispositivos e ajustar-se a cada um deles.
Tolerância a falhas	O sistema deve ser capaz de recuperar-se quando ocorre falhas, tais como problemas de conectividade com os serviços

Tabela 5.2: Características de Sistemas Ubíquos. (SPÍNOLA; TRAVASSOS, 2012)

Adicionalmente, a revisão sistemática realizada foi capaz de identificar os elementos arquiteturais que estão presentes na maior parte dos sistemas ubíquos, sendo assim potenciais elementos essenciais para esse tipo de sistema. Esses elementos são definidos, então, como requisitos arquiteturais, que devem estar presentes na arquitetura de referência. A Tabela 4.2 enumera os requisitos arquiteturais, exibindo uma breve descrição desses elementos e os estudos nos quais o elemento (ou um elemento com função semelhante) é definido.

Por fim, define-se que a arquitetura de referência deve permitir a construção de sistemas que atendam às características da Tabela 5.2, provendo elementos que deem suporte à característica ou a implementem efetivamente. Adicionalmente, a arquitetura de refe-

rência deve possuir elementos que realizem as funcionalidades dos elementos arquiteturais listados pela Tabela 4.2, uma vez que esses elementos são comuns em sistemas ubíquos.

5.1.3 Passo RA-3 - Projeto da Arquitetura de Referência

A partir dos requisitos e dos elementos arquiteturais identificados no passo anterior é definida uma arquitetura de referência.

Os *Stakeholders* para essa arquitetura são representados por todos os interessados em arquiteturas de sistemas em computação ubíqua, tais como: Projetistas de *middleware*, arquitetos e de sistemas ubíquos, etc.

Inicialmente identificou-se que o estilo em camada era recorrente nos sistemas e *middlewares* avaliados por este trabalho, então os elementos identificados pela revisão sistemática foram agrupados de acordo com as suas funcionalidades e interações com os demais elementos.

A arquitetura de referência construída está dividida em quatro camadas: *Infrastruture Layer, Services layer, Context Aware Computing Layer* e *Application Layer*, conforme ilustrado na Figura 5.1.

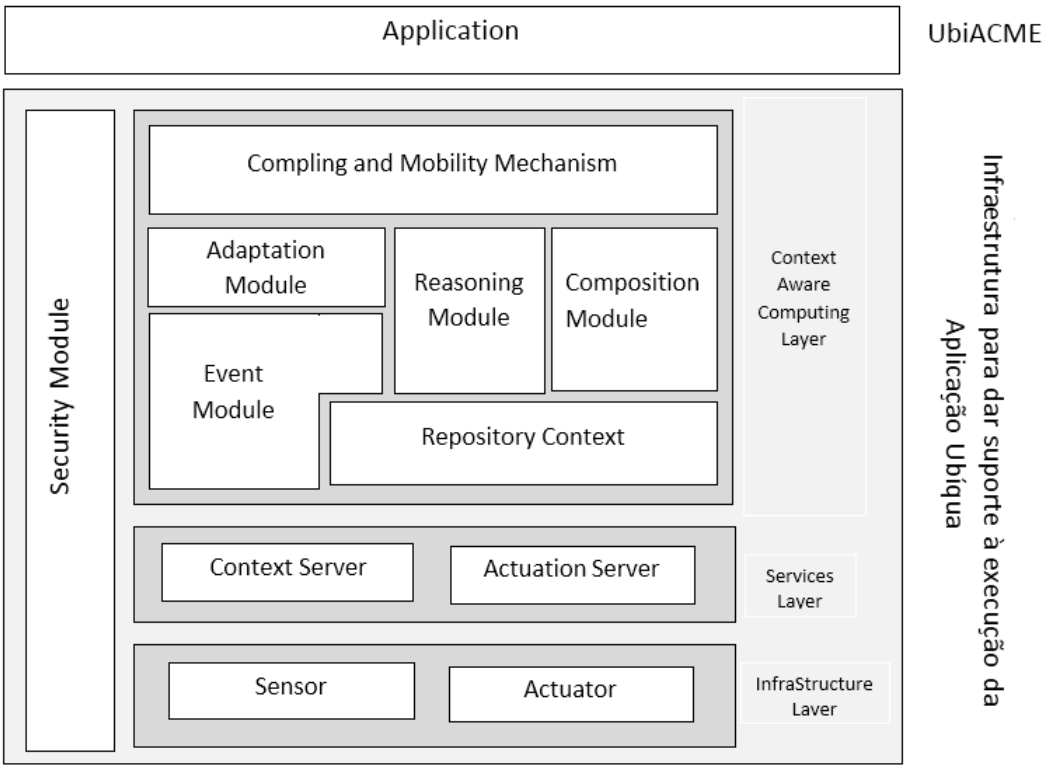

Figura 5.1: Arquitetura de Referência para Sistemas Ubíquos

A ***InfraStructure Layer*** provê o elemento ***Sensor***, responsável por fornecer informação de contexto, e o elemento ***Actuator***, responsável por alterar o ambiente, fornecendo *feedback* para o usuário. Esses elementos comunicam-se apenas com os elementos da camada de serviços, utilizando protocolos de comunicação baixo nível de abstração baseado nos protocolos de rede. Conforme ilustrado na Figura 5.1, a ***Services Layer*** usa esses elementos de infraestrutura para prover informações de contexto através de abstrações em software: ***Context Server*** e***Action Server***. Esses serviços podem utilizar mais de um sensor ou atuador para prover sua funcionalidade, permitindo assim algum tratamento preliminar sobre as informações de contexto, como correções de valor e cálculos estatísticos. A partir da ***Services Layer***, a ***Contect Aware Coputing*** pode ter acesso às informações de contexto e atuadores, fazendo requisições aos serviços correspondentes.

A **Context Aware Computing Layer** caracteriza sistemas ubíquos, sendo dividida em seis elementos:

1. ***Repository Context***, responsável por armazenar informações de contexto e prover uma interface de acesso a essas informações. Esse componente faz requisições aos serviços da ***Services Layer***, para manter as informações de contexto atualizadas, e disponibiliza uma interface para leitura dessas informações e armazenamento de outras informações que podem ser produzidas pelo sistema;

2. ***Event Module***, responsável pelo monitoramento dos serviços e das informações de contexto. Uma responsabilidade desse módulo é o monitoramento de condições de disparo de um evento pré-definido, dentre essas condições podem-se destacar critérios baseados em qualidade de serviço - QoS e qualidade de contexto- QoC, que são avaliados através dos resultados das requisições aos serviços da camada de serviços. Esse módulo possui um funcionamento semelhante a um *Observer*, também sendo responsável por notificar aos interessados sobre a ocorrência de um evento específico disparado por um serviço [E5], conforme Tabela 4.1. Para isso, o módulo provê uma interface na qual os outros módulos podem se inscrever para monitorar um determinado evento. Em caso de falhas nos serviços, também são disparados eventos, para que seja possível ao sistema recuperar-se automaticamente, substituindo o serviço defeituoso. Eventos também podem ser disparados quando uma nova informação de contexto torna-se disponível ou quando deixa de estar disponível;

3. ***Composition Module***, responsável por compor informações de contexto em informações mais abstratas. Esse módulo recupera os dados do repositório de contexto

e os utiliza para compor informações, que serão novamente armazenadas no repositório de contexto, permitindo assim que outros módulos tenham acesso a essas informações. Além disso, esse módulo deve disponibilizar uma interface para criação e edição de regras de composição que serão utilizadas, tornando o sistema mais flexível;

4. ***Reasoning Module***, responsável por produzir novas informações de contexto a partir das informações de contexto existentes, algumas vezes usando técnicas de inteligência artificial. De forma análoga ao módulo de composição, esse módulo recupera os dados do repositório de contexto e produz nova informação a partir desses dados, armazenando os dados no repositório ao final de sua execução. Adicionalmente, o módulo de raciocínio pode disponibilizar uma interface para manipulação e definição de regras de produção de informações;

5. ***Adaptation Module***, responsável por alterar o comportamento do sistema de acordo com os eventos disparados pelo Módulo de Eventos, de forma a permitir que o sistema adapte-se a diversos contextos e busque um estado eficiente de funcionamento. Para isso, esse módulo monitora eventos pré-estabelecidos e possui regras que associam ações a cada evento disparado. Essas ações podem ser reconfigurações nos módulos ou simples alterações no comportamento do sistema. O módulo de adaptação pode ser responsável por definir estados de funcionamento do sistema e, por isso, deve disponibilizar uma interface para acesso às suas regras;

6. ***Coupling and Mobility Mechanism***, responsável por abstrair a noção de ambiente, utilizando ferramentas de rastreamento, busca de serviço e computação móvel. Esse módulo acessa as informações de contexto e possui a responsabilidade de coordenar o funcionamento dos módulos de composição raciocínio e adaptação. Por abstrair a noção de ambiente, esse módulo deve ser responsável por identificar mudanças de ambiente e disponibilizar as funcionalidades do sistema com transparência de localização através de sua interface.

   Por fim, a camada da ***Application Layer*** é responsável por implementar as regras de negócio das aplicações, utilizando a estrutura provida pela ***Context Aware Computing Layer***.

Em paralelo às camadas supracitadas, ***Security Module*** é responsável por implementar as regras de segurança que serão utilizadas pelo sistema. Apesar da representação como módulo, representa um conceito transversal, podendo então não ser implementado

como um componente e sim possuir sua implementação espalhada ao longo de outros elementos do sistema. Todavia, quando implementado como módulo, esse elemento pode acessar diretamente qualquer componente ou conexão do sistema, de acordo com as regras da aplicação. Dentre as atribuições desse elemento, pode-se citar: validação de serviços, segurança em comunicações de elementos distribuídos e restrições de acessos.

A Figura 5.2 apresenta um diagrama da visão de componente com as interfaces disponibilizadas e requeridas das quatros camadas da arquitetura de referência para sistemas ubíquos.

A Figura 5.3 complementa a Figura 5.2, apresentando os elementos arquiteturais como pacotes, explicitando a interação entre tais elementos.

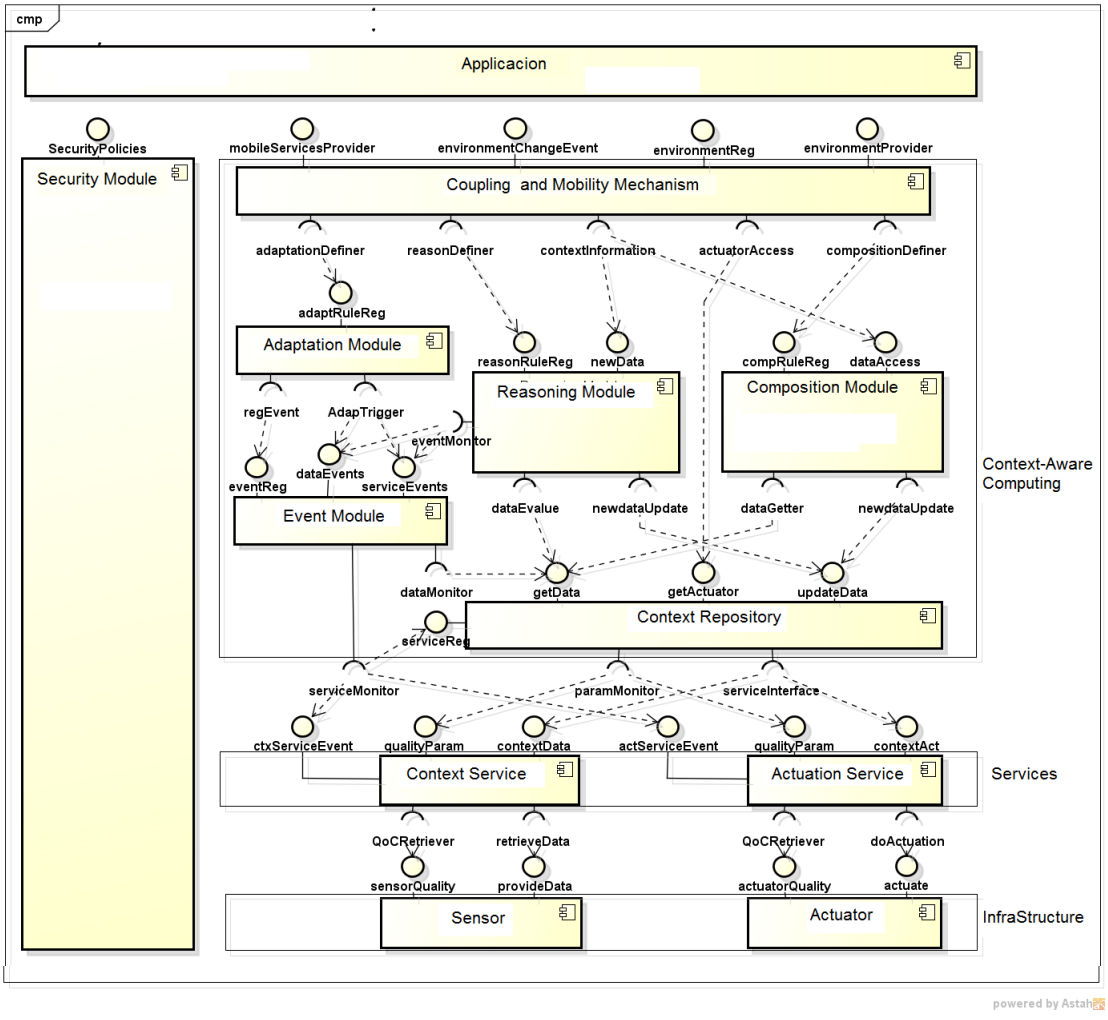

Figura 5.2: Visão de Componentes da Arquitetura de Referência para Sistemas Ubíquos.

A Figura 5.3 exibe um diagrama de pacotes como os elementos descritos pela Figura 5.2, exibindo explicitamente seus relacionamentos e sua organização interna. Alguns componentes possuem pacotes internos que não aparecem na Figura 5.2, que se referem a

elementos de grande importância para o funcionamento do componente em questão.

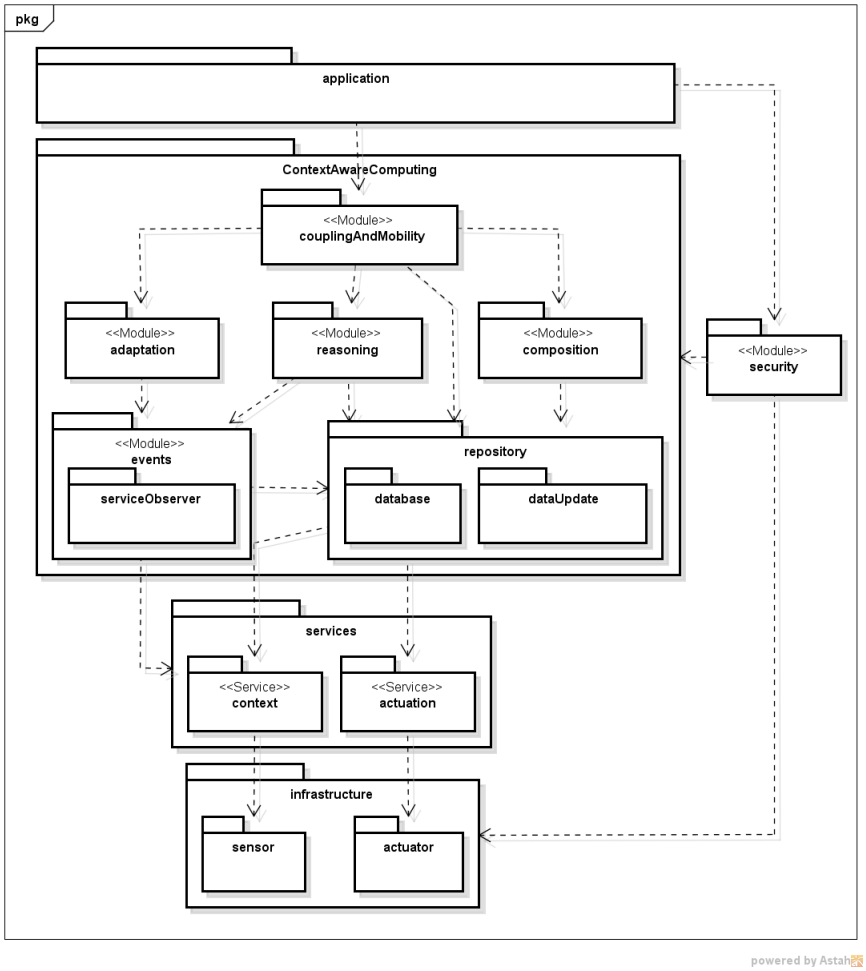

Figura 5.3: Visão de Pacotes da Arquitetura de Referência para Sistemas Ubíquos.

Na Figura 5.4, os elementos de ***Sensor*** e ***Actuator*** foram colocados no mesmo dispositivo físico que seus servidores. Os componentes que realizam muito acesso a dados, como os módulos de ***Composition e Reasoning*** foram implementados no mesmo servidor ***DataServer*** onde o ***Context Repository*** foi hospedado, visando minimizar o tempo de requisição e resposta entre esses elementos altamente dependentes. Os módulos de ***Event e Adaptation*** possuem seus próprios servidores, entretanto, o módulo ***Event*** pode ser implementado no mesmo dispositivo que o ***Context Repository***, a depender da quantidade de eventos de dados que o sistema irá monitorar. O modulo ***Coupling and Mobility Mechanism*** foi implementado em seu próprio servidor, que deve possuir mecanismos e ferramentas para otimização de comunicação com dispositivos móveis, que são fortes candidatos a dispositivos nos quais a Aplicação será implantada.

A implementação e implantação dos elementos definidos pela arquitetura de referência dependem fortemente do sistema em específico, entretanto, pode-se definir uma visão

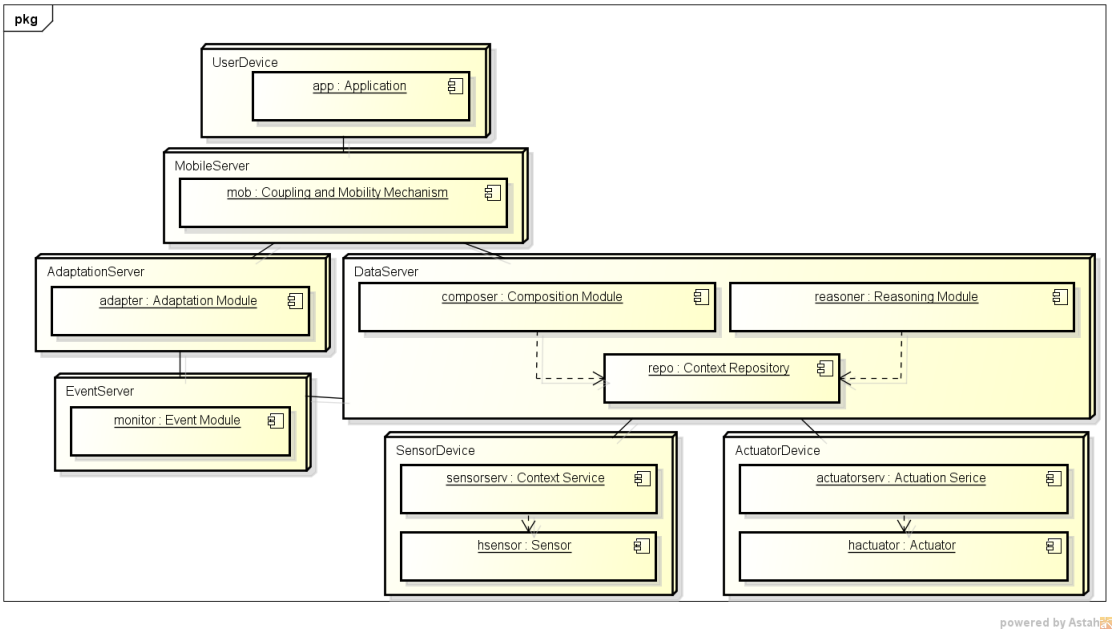

Figura 5.4: Visão de Implantação de RA-Ubi.

de implantação, visando prover ao arquiteto uma noção de como o sistema pode ser implantado. A Figura 5.4 exibe essa visão, que deverá ser refinado (ou alterado) de acordo com as necessidades da aplicação em específico.

É importante dizer que os elementos presentes na RA-Ubi podem ser implementados e utilizados de várias formas, dependendo dos objetivos do sistema. A Figura 5.4 mostra um diagrama da visão de implantação possível de um sistema baseado em RA-Ubi. O ***Sensor*** e ***Context Service***, bem como os componentes de ***Actuation Server*** e de ***Atuator*** foram implantados como um único elemento - o ***SensorDevice*** e o ***ActuatorDevice***, já que os elementos de hardware possam executar as suas próprias interfaces de software.

O módulo ***Composition***, ***Reasoning*** e o ***Context Repository***, foram implantados em um único dispositivo, o ***DataServer***, uma vez que têm uma comunicação intensa. Os módulos ***Adaptation*** e ***Event*** têm o seu próprio servidor, ***AdaptationServer*** e ***EventServer***, respectivamente, o que lhes permite ser mudados durante a execução, independentemente do outro servidor. Nesse exemplo, o ***Coupling and Mobility Mechanism*** foi implantado em um ***MobileServer***, semelhante a um ***Cloudlet***. Finalmente, ***Application*** é implantado em um dispositivo do usuário, que usará todas as informações do ambiente para alcançar seus objetivos.

Além disso, as atividades que os componentes RA-Ubi executam para alcançar algum objetivo específico foram descritas em um conjunto de diagramas de atividades. Esses diagramas foram construídos com o objetivo de prover uma visão de execução para os

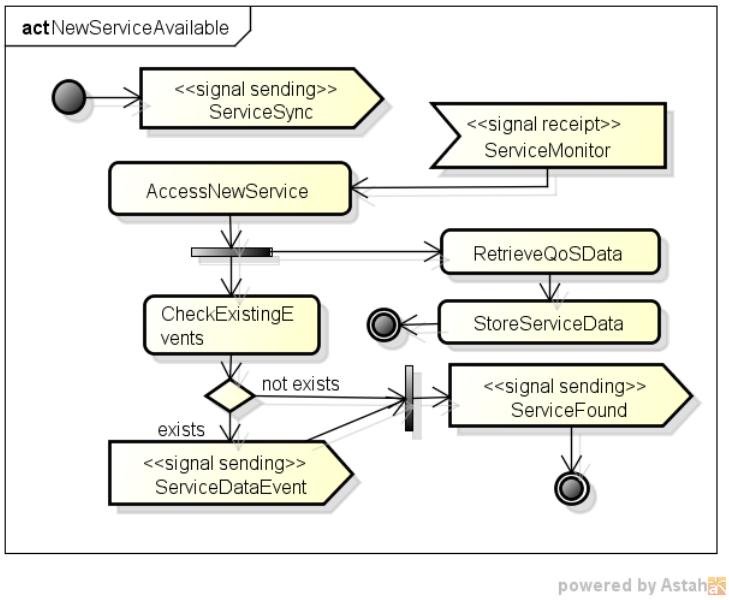

Figura 5.5: Diagrama de Ativação de Novo Serviço Disponível.

elementos que compõem a arquitetura. A Figura 5.5 mostra o diagrama de atividade ***NewServiceAvaliable***, que é executada pelo módulo de eventos quando ele detecta novos serviços no ambiente. Na Figura 5.5, a atividade começa quando o evento ***ServiceSync*** é acionado. Na próxima etapa, o Módulo de Eventos capta esse sinal e acessa o serviço usando uma interface padrão. Depois de acessar, ele recupera a qualidade de serviço (QoS) de dados e armazena no repositório. Paralelamente, ele verifica se há registro de eventos existentes, provocando um evento de dados se ele existir. Finalmente, ele aciona um evento Serviço encontrado.

A Figura 5.6 mostra o diagrama de atividades para a mudança de um fornecedor para um determinado serviço causado pelo parâmetros QoC *(Quality of Context)*. Na Figura 5.6, o ***EventModule*** é responsável por recuperar os dados QoC da operadora atual. Se esse parâmetro foi o pior de acordo com as definições das aplicações, o ***EventModule*** irá atualizar os dados QoC para todos os prestadores de serviços existentes e selecionar o melhor fornecedor. Se o melhor provedor não é o atual, uma adaptação é acionada para alterar o fornecedor, sendo o ***AdaptationModule*** responsável por essa adaptação.

A Figura 5.7 mostra o diagrama de atividades para a solicitação de serviço de atuação. Essas atividades são executadas quando um Serviço de Atuação recebe um pedido e começa com a aceitação desse pedido. Depois de aceitar, o Serviço de Atuação identifica os atuadores relacionados, ou seja, os atuadores que serão envolvidos para executar o pedido dado (o serviço frequentemente estará relacionado a um único atuador). Em seguida, ele executa uma sincronização do relógio para sincronizar todos os atuadores relacionados (al-

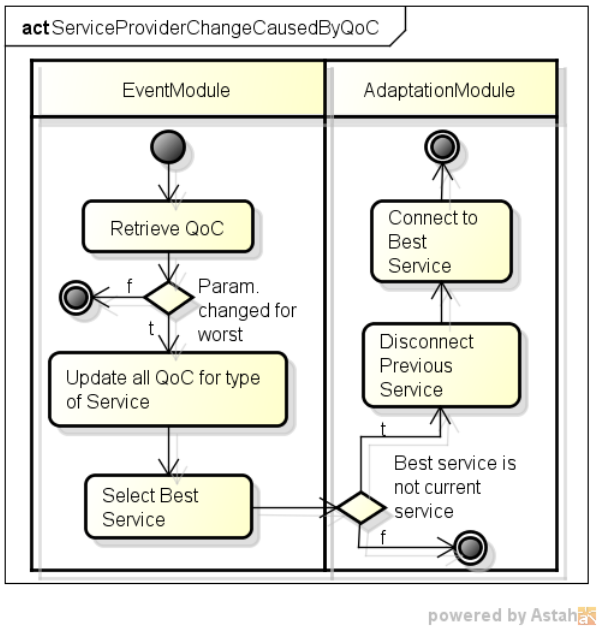

Figura 5.6: Mudança de Provedor de Serviço Causado por QoC.

gumas atuações podem precisar ser executadas em uma determinada ordem). Finalmente, o Serviço de Atuação envia sinais de instrução para todos os atuadores relacionados (XU et al., 2011), que serão responsáveis por executá-los. É importante destacar que os Serviços de Atuação podem prestar serviços de alto nível que agrega muitos atuadores, por exemplo: um serviço de acionamento pode fornecer um serviço Ilumina um quarto, e para executar a tarefa de iluminar, o serviço usará tanto iluminação e atuadores janela.

A Figura 5.8 mostra o diagrama de atividades que envolve o ***Coupling and Mobility Mechanism***, o ***Event Module*** e o ***Context Repository***. Esse diagrama mostra as atividades que são realizadas pelas partes constituintes da arquitetura de referência quando uma mudança de ambiente é detectada. O processo inicia com a própria detecção dessa mudança de ambiente, através das atividades *"Check Location"* e *"Identify Environment"*. Caso não haja uma alteração no ambiente (ou seja, o ambiente atual é igual ao que foi identificado), a execução encerra. Caso contrário, o ***Coupling and Mobility Mechanism*** vai verificar as regras existentes para o novo ambiente atual e desativar as regras (eventos e composição, basicamente) que estão ativas no momento. Após essa etapa, esse mesmo mecanismo vai verificar a existência de definições para esse ambiente (definições de regras), caso essas regras existam (parte inferior), serão recuperadas as ontologias do ambiente e então as regras de eventos e composição serão ativadas. Caso não existam essas regras, o sistema irá produzir um conjunto de "regras padrão" (por regras eu me refiro a eventos e composições) que dependem do domínio específico do sistema. Caso contrário, o

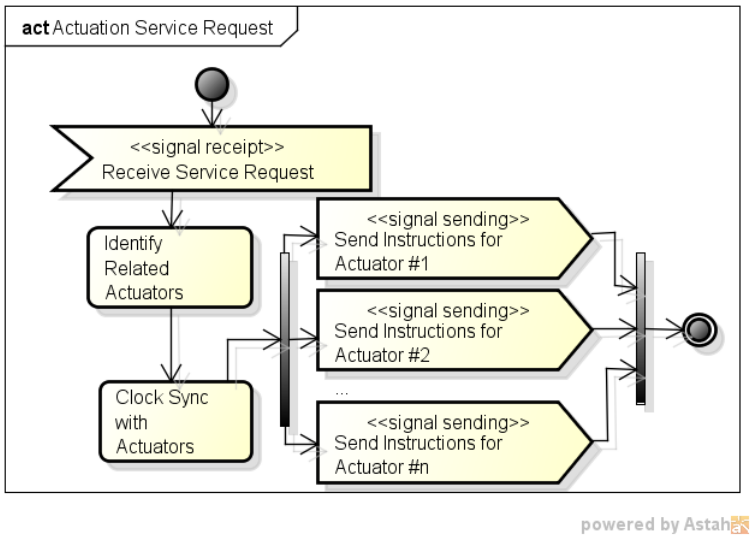

Figura 5.7: Requisição ao Serviço de Atuação.

Coupling and Mobility Mechanism vai verificar as regras existentes para o novo ambiente atual e desativar as regras (eventos e composição, basicamente) que estão ativas no momento. Após essa etapa, esse mesmo mecanismo vai verificar a existência de definições para esse ambiente (definições de regras), caso essas regras existam (parte inferior), serão recuperadas as ontologias do ambiente e então as regras de eventos e composição serão ativadas. Caso não existam essas regras, o sistema irá produzir um conjunto de "regras padrão" (por regras eu me refiro a eventos e composições) que dependem do domínio específico do sistema. Definidas as regras, o ***Event Module*** vai identificar quais são os serviços que estão relacionados a essas regras, isto é, quais serviços são utilizados pelas regras de eventos ou composição. Então será atualizado o registro de serviços, de modo a enxugar o registro de serviços, mantendo apenas esses serviços que serão utilizados (naturalmente novos serviços podem ser inseridos no registro, posteriormente). Após a atualização do registro de serviços, o ***Context Repository*** irá identificar quais dados de contexto estão disponíveis, a depender dos serviços que foram inseridos no registro, para então desativar o acesso aos dados indisponíveis. Por fim, o módulo de eventos atualiza todos os dados que estão disponíveis e o mecanismo de acoplamento e mobilidade atualiza a tabela de serviços, que contém não apenas os serviços disponíveis, como também serviços para acesso a novos dados frutos de composição e eventos.

5.1.4 Passo RA-4 - Avaliação da Arquitetura de Referência

Essa etapa divide-se em duas sub-etapas essenciais: validação e verificação, na qual a validação será feita com base no método definido em (ANGELOV et al. 2008) que propõe

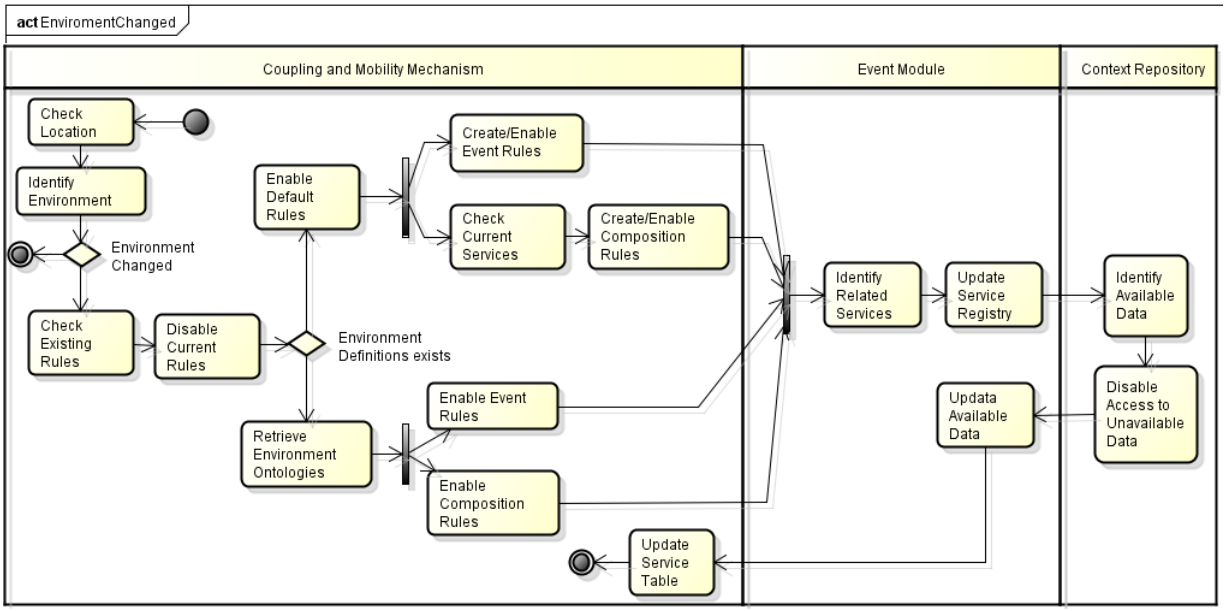

Figura 5.8: Mudança de Ambiente.

uma avaliação de arquiteturas de referência por meio de um método de propósito geral e a verificação através de um *checklist* na qual serão identificados quais elementos que identificamos como essenciais que agregam ao sistema cada característica definida no RA-2. A Tabela 5.3 sintetiza os resultados desse *checklist*, relacionando às características de sistemas ubíquos aos elementos da arquitetura de referência.

O estabelecimento da relação entre as características e os elementos foi pautada em uma análise cuidadosa da literatura desse domínio, focando-se nas características e no papel de cada elemento arquitetural identificado na revisão sistemática descrita na Capítulo 4. A seguir, discute-se como cada característica associa-se aos elementos, conforme apresentado pela Tabela 5.3.

Onipresença de Serviço pode ser atendida pelo Mecanismo de Acoplamento e Mobilidade uma vez que tal mecanismo emprega protocolos de comunicação móvel que permitem acesso aos serviços em qualquer lugar e a qualquer momento.

Invisibilidade está relacionada: (i) ao elemento Sensor que capta informações de contexto do ambiente sem exigir ordem explícita do usuário; (ii) ao elemento Atuador que repassa reações do sistema ao ambiente; (iii) aos mecanismos de Serviço de Contexto e Serviço de Atuação que são os elementos arquiteturais que permitem que os sensores e atuadores possam ser acessados.

Sensibilidade ao contexto é uma característica fundamental de qualquer sistema ubíquo. Sensor e Serviço de Contexto estão diretamente relacionados a essa caracterís-

Característica	**Elemento**
Onipresença de serviços	Mecanismo de acoplamento e Mobilidade
Invisibilidade	Sensor Atuador Serviço de contexto Serviço de atuação
Sensibilidade ao Contexto	Sensor Serviço de contexto Repositório de Contexto Módulo de raciocínio Módulo de Agregação ou Composição
Comportamento adaptativo	Serviço de contexto Módulo de eventos Mecanismo de Adaptação
Captura de experiência	Módulo de raciocínio
Descoberta de serviços	Módulo de eventos
Composição de função	Módulo de raciocínio Módulo de Agregação ou Composição
Interoperabilidade espontânea	Mecanismo de acoplamento e Mobilidade
Heterogeneidade de dispositivos	Sensor Módulo de eventos
Tolerância a falhas	Mecanismo de acoplamento e Mobilidade Módulo de eventos Mecanismo de Adaptação

Tabela 5.3: Características de Projetos de Computação Ubíqua Associados aos Elementos Arquiteturais Comuns de Sistemas Ubíquos

tica, permitindo a identificação do contexto e a realização de operações de acordo com o contexto corrente. O Repositório de Contexto é responsável pelo armazenamento das informações. O Módulo de Raciocínio realiza inferências sobre a informação de contexto e pode produzir novas informações. O Módulo de Agregação ou Composição realiza a composição de informações de contexto.

Comportamento Adaptativo define que o sistema deve se adaptar ao ambiente, oferecendo serviços de acordo com o contexto. O Serviço de Contexto é essencial para identificação do contexto. O Módulo de Eventos tem a responsabilidade de gerar o evento de mudança de contexto para que a adaptação seja realizada. Essa característica pode também estar atribuída ao Repositório de Contexto, como em E5(Um dos artigos selecionados como relevante na revisão sistemática, conforme Tabela 4.1). Por fim, o Mecanismo de Adaptação realiza a adaptação necessária para o novo contexto.

Captura de Experiência consiste na captura e armazenamento de informações de aprendizado para uso futuro. Naturalmente está associada ao Módulo de Raciocínio, que emprega mecanismos de aprendizado e outros recursos da inteligência artificial. Esse módulo possui uma atribuição semelhante ao que em alguns trabalhos, como no estudo E8, aparece como Módulo de Agregação ou Composição. A diferença existente entre as definições desses módulos reside no fato de que o Módulo de Raciocínio é capaz de produzir nova informação de contexto utilizando mecanismos de inteligência, enquanto o Módulo de Agregação ou Composição apenas agrupa ou compõe a informação de contexto. Na

maioria dos trabalhos, entretanto, esses módulos são integrados.

Descoberta de Serviços é atendida, na maioria dos trabalhos, pelo Módulo de Eventos, que é pró-ativo em relação aos serviços, realizando o monitoramento e a descoberta de serviços presentes no ambiente, disponibilizando-os por meio de um mecanismo *publish-subscribe*. Entretanto, esse comportamento também pode ser agregado ao Repositório de Contexto, como acontece no trabalho descrito em E5.

Composição de Função é a característica que determina a habilidade do sistema em disponibilizar novos serviços ao usuário, baseado nos serviços existentes. O Módulo de Raciocínio está relacionado a essa característica, uma vez que esse módulo deve ser capaz de identificar os serviços que serão utilizados como serviços básicos (E2, E3, E8, E9) e compô-los segundo alguma regra de negócio. O Módulo de Agregação ou Composição, em alguns trabalhos (E8, E9), é usado para realizar a composição. O Módulo de Raciocínio, adicionalmente, pode inferir novas informações de contexto para disponibilizar como serviço adicional. Os novos serviços que podem ser oferecidos, entretanto, variam entre as aplicações.

Interoperabilidade Espontânea é a capacidade do sistema de usar diversos elementos sem a necessidade de intervenção externa. Essa característica é atendida pelo Mecanismo de Acoplamento e Mobilidade, uma vez que esse é o elemento responsável por protocolos de computação móvel e por tratar, em alto nível, trocas de ambiente (E2, E4).

Heterogeneidade de Dispositivos define que os elementos distintos devem ser acessados de forma uniforme. Os trabalhos E8 e E9 discutem o papel dos sensores na disponibilização das informações advindas de fontes heterogêneas, bem como do Módulo de Eventos para monitorar diferentes serviços de forma transparente para os usuários.

Em relação à característica de **Tolerância a falhas**, o Mecanismo de Acoplamento e Mobilidade está diretamente ligado aos dispositivos móveis utilizados pelo usuário para acesso ao sistema. Por isso, esse mecanismo deve ser capaz de tratar os problemas mais comuns relacionados à computação móvel, como instabilidades de conexão e oscilações no fluxo de dados (como mostrado em E4). O Módulo de Eventos, por sua vez, pode disparar eventos diversos, tais como surgimento de novos serviços e erros ou falhas em qualquer serviço. As falhas poderão ser tratadas pelo Mecanismo de Adaptação.

Em suma, observa-se que os elementos arquiteturais comuns identificados neste trabalho atendem adequadamente às características de sistemas ubíquos, de forma que a arquitetura de referência produzida não tem elementos irrelevantes.

5.1.4.1 Avaliação da RA-Ubi

A avaliação de uma arquitetura ajuda a identificar os seus pontos fortes e fracos e dá uma indicação para o sucesso do desenvolvimento de sistemas e processos de implementação. A arquitetura de referência serve como uma ferramenta de orientação para projetos que ocorrem em diversos contextos. Assim, a sua avaliação antes da sua adoção pelas partes interessadas (*stakeholders*) é de grande importância. Além disso, uma forte avaliação positiva de uma arquitetura de referência é um incentivo para a sua adoção (ANGELOV et al., 2008).

A literatura fornece um conjunto de métodos que podem ser usados para avaliar as qualidades de arquitetura de software. Breves resumos e comparação dos métodos mais populares podem ser encontrados em BABAR e GORTON (2004) e IOMITA et al. (2002). A literatura não reporta métodos dedicados à avaliação holística de arquiteturas de referência (ANGELOV et al., 2008). Dessa forma, avaliamos a RA-Ubi fazendo comparações entre os seus elementos com os elementos da arquiteturas de sistemas ubíquos, já citados no Passo RA-1, na identificação das fontes de informação.

Na avaliação, foram realizadas uma série de estudos de caso a partir dos trabalhos integrantes do grupo iii "arquiteturas de aplicações para computação ubíqua". O objetivo foi descrever as arquiteturas definidas por cada um desses trabalhos utilizando como base os elementos de RA-Ubi. Dessa forma, então, avalia-se se a arquitetura de referência provê todos os elementos essenciais, conforme proposto, e se existem elementos fundamentais que não foram incluídos nessa arquitetura de referência.

5.1.4.2 Estudo 1: *ActiveBadge*

ActiveBadge (E11) (WANT et al., 1992) é um dos sistemas ubíquos precursores, desenvolvido pela *Olivetti Research Ltda* e consiste de um sistema de localização baseado em crachás. O sistema foi desenvolvido com o objetivo de funcionar como ponto eletrônico para os funcionários, agilizando o processo que é comum em várias empresas. O sistema possui uma infraestrutura muito simples, no qual os elementos baseiam-se numa arquitetura distribuída de sensores. A arquitetura em camadas de uma aplicação em execução no servidor apresentada na Figura 5.9 é composta de quatro camadas:

1. ***Network Control***, o servidor de rede que supervisiona o funcionamento da rede de sensores;

2. ***Representation***, a apresentação de informações que é responsável pela gestão de dados e controle de informações de localização;

3. ***Data Processing***, o processamento de dados que seleciona a informação interessante no momento da variação de localização; e

4. ***Display Interface***, a interface do usuário para exibir as informações textuais sobre o crachá do usuário.

O sistema *ActiveBadge* tem uma arquitetura de hardware para sistemas sensíveis a localização ao invés de uma arquitetura de sistema sensível ao contexto com vários componentes de software. Ele é específico para sistemas de localização e não pode ser facilmente utilizado para outros tipos de sistemas de detecção de contexto. Finalmente, não faz qualquer abstração de informação contextual (informações de localização, nesse caso) o que o torna muito dependente da infraestrutura de hardware.

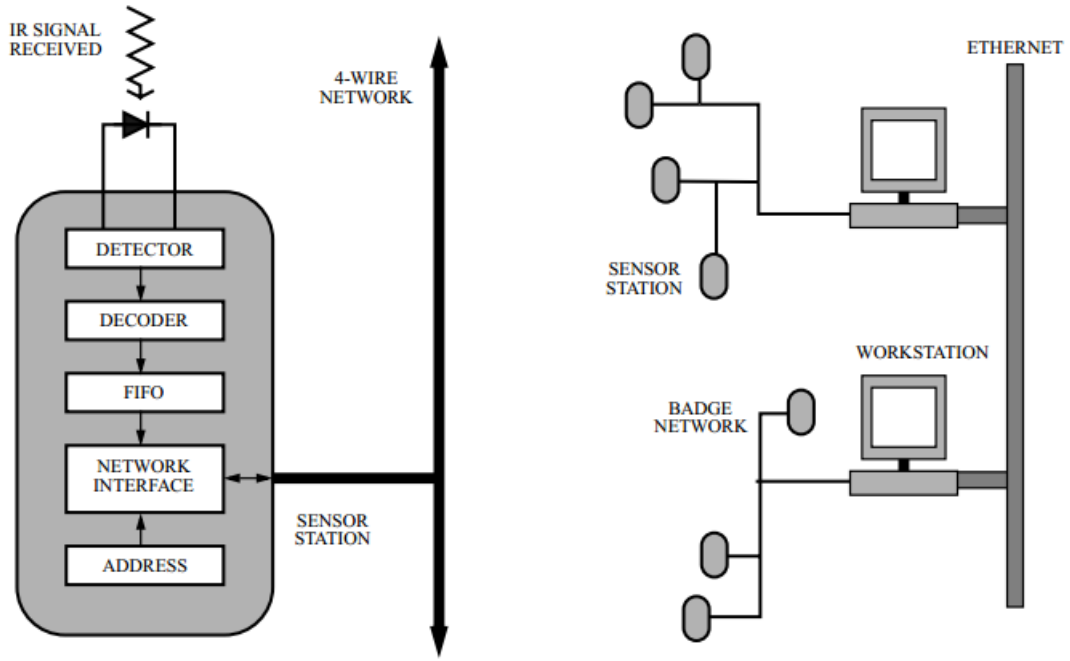

Figura 5.9: Arquitetura do *ActiveBadge*. (WANT et al., 1992)

Conforme observado na Tabela 5.4, para modelar a arquitetura do *ActiveBadge* utilizando a arquitetura de referência RA-Ubi, foram usados apenas os elementos sensores, módulo de composição e repositório de contexto, que se mostraram suficientes para as funcionalidades de coletar informações de contexto e armazenar tais informações, dando margem para composição dos dados em informações de mais alto nível.

Elemento ActiveBadge	**Elemento Arq. Ref.**
Aplicação	Aplicação
gestão dos dados	Repositório de Contexto
Sensores	Sensores

Tabela 5.4: Relação entre Elementos da Arquitetura *ActiveBadge* com Elementos da RA-Ubi

5.1.4.3 Estudo 2: *CMF*

O CMF *(Context Management Framework)* (E12) (KORPIPAA et al., 2003) é um *framework* de gerenciamento de contexto que permite o raciocínio semântico do contexto em tempo real e até mesmo na presença de ruídos, incertezas e rápida variação do contexto. Essa *framework* oferece informações contextuais para aplicativos usando um modelo de comunicação baseado em eventos. O CMF possui uma arquitetura cliente/servidor composta por cinco componentes básicos, conforme ilustrado pela Figura 5.10: (i) ***Context manager***, responsável pelo armazenamento de informações contextuais no servidor e a entrega de contexto para os clientes usando diferentes tipos de mecanismos (solicitação/resposta, *publish-subscribe*, etc); (ii) ***Resource server*** de recursos, responsável pela aquisição de informação contextual a partir de sensores físicos e sua interpretação de acordo com um formato específico antes de enviá-los para o gerente de contexto; (iii) ***Context recognition service***: responsável pela conversão do fluxo de dados para uma apresentação definida na ontologia do contexto; (iv) ***Change detection service***: responsável pela detecção de mudança de serviço e, portanto, na mudança de contexto; e (v) ***Security***: responsável pela verificação e controle da informação contextual.

Vários dos componentes da arquitetura do CMF possuem um correspondente na arquitetura de referência proposta. A Tabela 5.5 relaciona os elementos da arquitetura do CMF aos da arquitetura de referência RA-Ubi.

Elemento CMF	**Elemento RA-Ubi**
Aplicação	Aplicação
Gerente contexto	Repositório de Contexto
Servidor de recursos	Serviço de Contexto Módulo de Raciocínio
Serviço de reconhecimento de contexto	Módulo de Composição
Detecção de mudança no serviço	Módulo de Adaptação Módulo de Eventos
Segurança	Módulo de Segurança

Tabela 5.5: Relação entre Elementos da Arquitetura CMF com Elementos da RA-Ubi

Conforme observado na Tabela 5.5, o elemento servidor de recursos que é responsável

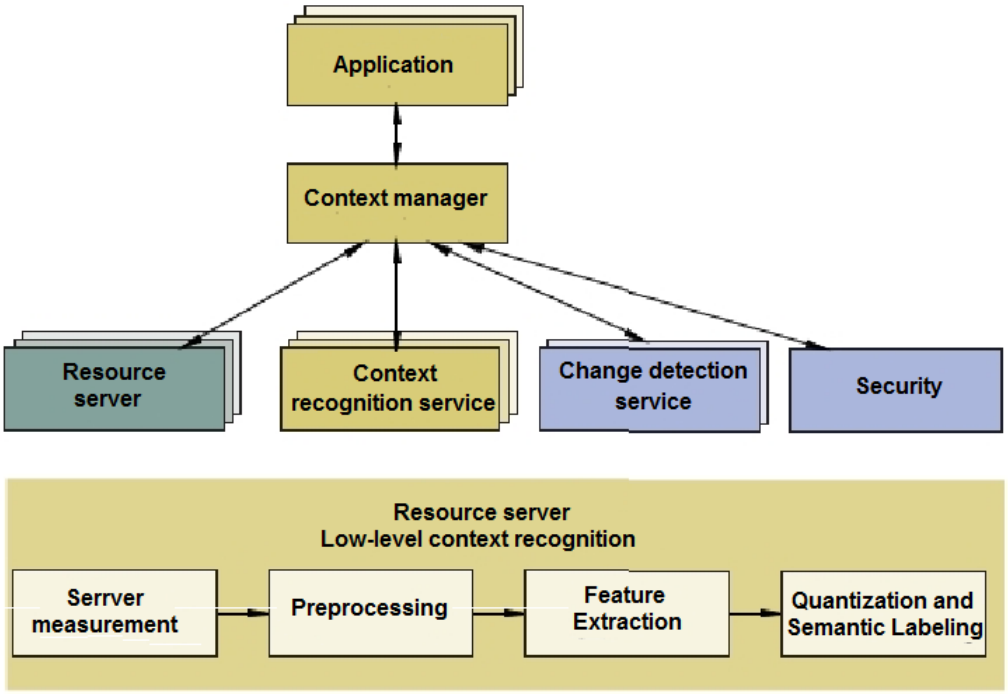

Figura 5.10: Arquitetura do CMF. (KORPIPAA et al., 2003)

pela aquisição de informação contextual a partir de sensores físicos e sua interpretação de acordo com um formato específico antes de enviá-los para o gerente de contexto, agrega a responsabilidade de dois elementos da RA-Ubi, o serviço de contexto e módulo de raciocínio, já o elemento detecção de mudança no serviço agrega a responsabilidade dos elementos módulo de adaptação e módulo de eventos da arquitetura de referência e RA-Ubi.

5.1.4.4 Estudo 3: *CoBrA*

CoBrA - The Context Broker Architecture (E13) (CHEN, 2004) é uma arquitetura baseada em ***Broker*** um agente corretor de apoio ao desenvolvimento de aplicações sensíveis ao contexto em um espaço inteligente. O ***Context broker*** é um agente autônomo que gerencia e controla o modelo de contexto de um domínio específico. O agente corretor tem uma arquitetura em camadas, conforme ilustrado na Figura 5.11, contendo quatro componentes: (i) ***Context knowledge base***; (ii) ***Context reasoning mechanism***; (iii) ***Context acquisition module***; e (iv)***Privacy management module***.

Na Figura 5.11, o broker possui uma ***Context knowledge base***, que é o componente que gerencia o armazenamento de informações, o que inclui as ontologias para descrever vários tipos de contextos e as informações oriundas do mecanismo de raciocínio de contexto.

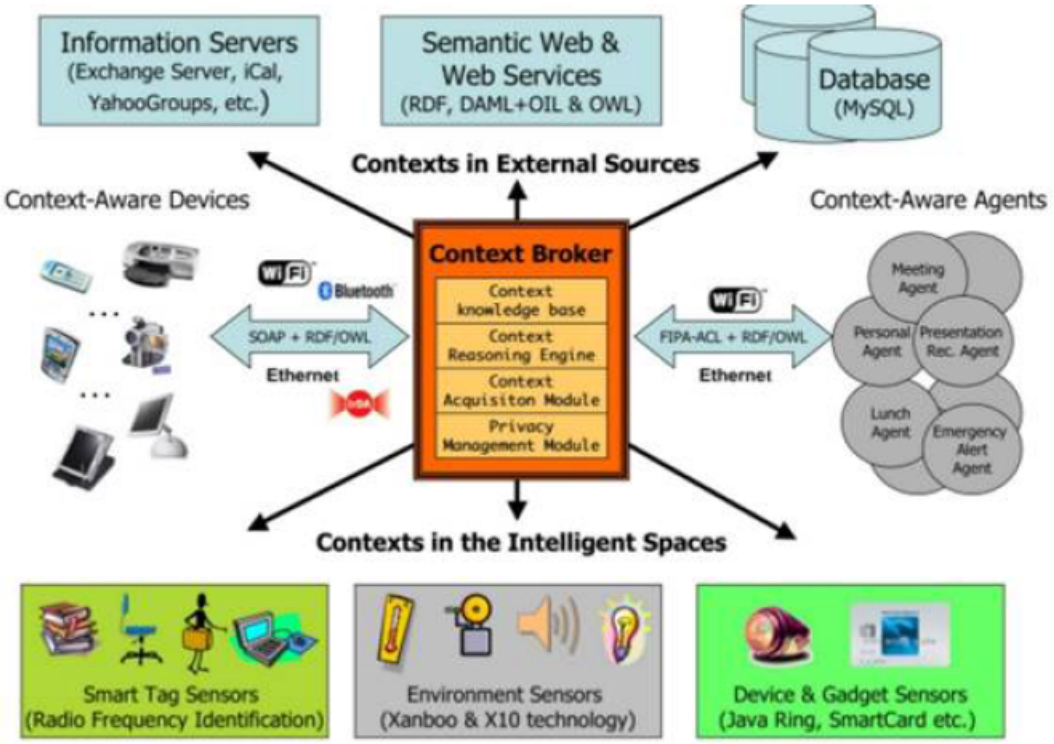

Figura 5.11: Arquitetura do CoBrA. (CHEN, 2004)

O módulo de ***Context reasoning mechanism*** interpreta os dados sensoriais providos pelo módulo de aquisição de contexto, agregando e inferindo informações de contexto. O módulo também é responsável por resolver problemas de inconsistência. O módulo de ***Context acquisition*** é um conjunto de procedimentos para aquisição de informação contextual a partir de sensores, agentes e da web. Esse módulo fornece uma camada de abstração para os componentes de baixo nível, apresentando uma interface simples para o que pode ser adquirido através de protocolos de comunicação de baixo nível. Por fim, o módulo de ***Privacy management*** é responsável por assegurar a política de privacidade adequada quando o broker tenta compartilhar informações de um determinado usuário. Esse módulo controla os acessos às informações disponíveis ao sistema.

A Tabela 5.6 associa os elementos da arquitetura CoBrA aos elementos da arquitetura de referência proposta por este trabalho.

Elemento CoBrA	**Elemento RA-Ubi**
Base de conhecimento de contexto	Repositório de Contexto
Módulo de raciocínio de contexto	Módulo de Raciocinio Módulo de Composição
Módulo de Aquisição	Servidor de Contexto
Módulo de gerenciamento de privacidade	Módulo de Segurança

Tabela 5.6: Relação entre Elementos da Arquitetura CoBrA com Elementos da RA-Ubi

Elemento Hydrogen	**Elemento RA-Ubi**
Adaptadores	Serviço de Contexto Serviço de Atuação
Servidor de Contexto	Repositório de Contexto
Aplicação	Aplicação

Tabela 5.7: Relação Entre Elementos da Arquitetura *Hydrogen* com Elementos da RA-Ubi

Conforme observado na Tabela 5.6, o módulo de raciocínio e contexto agrega as responsabilidades de dois módulos da RA-Ubi, os módulos de raciocínio e composição. O módulo de aquisição é responsável por acessar os sensores para recuperar as informações de contexto, o que corresponde ao servidor de contexto.

5.1.4.5 Estudo 4: *Hydrogen*

O *framework Hydrogen* (E14) (HOFER et al., 2002) é voltado para o desenvolvimento de aplicações sensíveis ao contexto para cenários de computação móvel. Sua arquitetura segue um estilo em camadas, composta de três camadas: (i) ***Adaptor***, responsável por coletar as informações de contexto; (ii) ***Management***, capaz de armazenar e disponibilizar uma interface simples para acesso às informações de contexto; e (iii) ***Application***, que implementa as regras de negócio usando as funcionalidades providas pela camada de gerenciamento. Conforme ilustrado pela Figura 5.12, a camada de ***Adaptor*** é composta de vários adaptadores, que recuperam informações de contexto e armazenam tais informações na camada de gerenciamento, evitando problemas de *deadlocks* que poderiam ser ocasionados por acessos diretos. A camada ***Management*** possui um único componente: ***Context server***, responsável por armazenar todas as informações de contexto recuperadas pela camada de adaptação e disponibilizá-las para a camada ***Application***, através de comunicação *peer-to-peer*. Por fim, a camada ***Application*** varia de acordo com o sistema que está sendo construído sobre o *framework*. Essa camada acessa a camada ***Management*** para obter informações de contexto que poderão ser combinadas com as informações de sistema.

A estrutura simplificada do *framework Hydrogen* possui elementos cujas funcionalidades se encaixam exatamente em alguns elementos da arquitetura RA-Ubi, conforme ilustra a Tabela 5.7. A arquitetura do *framework* agrupa as funcionalidades do serviço de contexto e de atuação, fazendo dos elementos sensoriais e atuadores componentes externos ao sistema, que podem ser implementados de qualquer forma, desde que se comuniquem com os adaptadores através de comunicação direta usando o protocolo TCP/IP.

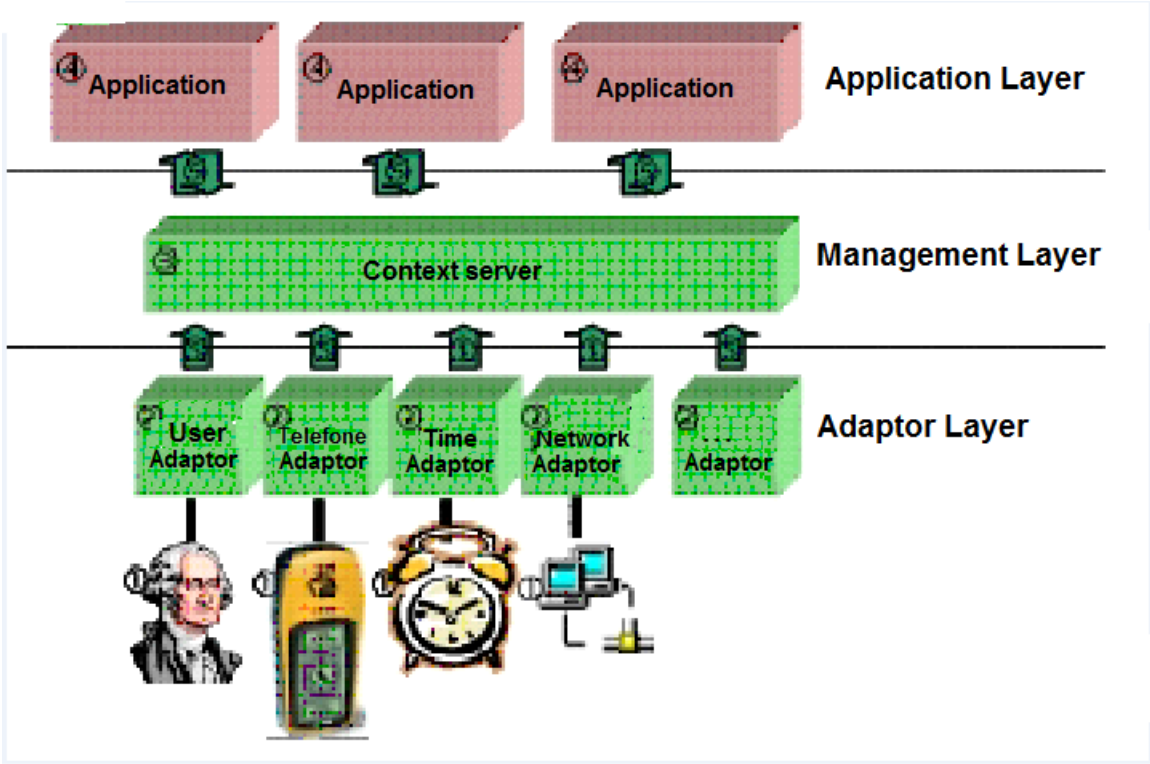

Figura 5.12: Arquitetura do *Hydrogen*.(HOFER et al., 2002)

5.1.4.6 Estudo 5: *JCAF*

JCAF (E15) (BARDRAM, 2005) é um *framework* para desenvolvimento de aplicações cientes de contexto implementado em Java. O *framework* provê um conjunto de entidades responsáveis por abstrair informações de baixo nível, como a comunicação com os sensores, de forma simples e intuitiva. O *framework* utiliza uma arquitetura cliente-servidor dividida em três camadas: (i) ***Context Semsor and Actuator Layer***, que englobam os elementos responsáveis por captar informações de ambiente e fornecer *feedback* para o usuario; (ii) ***Context Service Layer***, responsável por fornecer uma camada de abstração adicional para as informações de contexto, permitindo o uso de dados de contexto com maior nível de abstração; e (iii) ***Context client Layer***, responsáveis por implementar as regras de negócio, acessando as informações de contexto providas por um ou mais servidores de contexto.

A Figura 5.13 exibe a arquitetura do *framework JCAF*, na qual na ***Context Semsor and Actuator Layer*** encontram-se elementos ***Semsor and Actuator***, responsáveis pela interação com o usuário e elementos de monitoramento, responsáveis por interagir com esses sensores e atuadores associando-os a uma entidade. Adicionalmente, os elementos de monitoramento são responsáveis por definir políticas de cooperação entre sensores

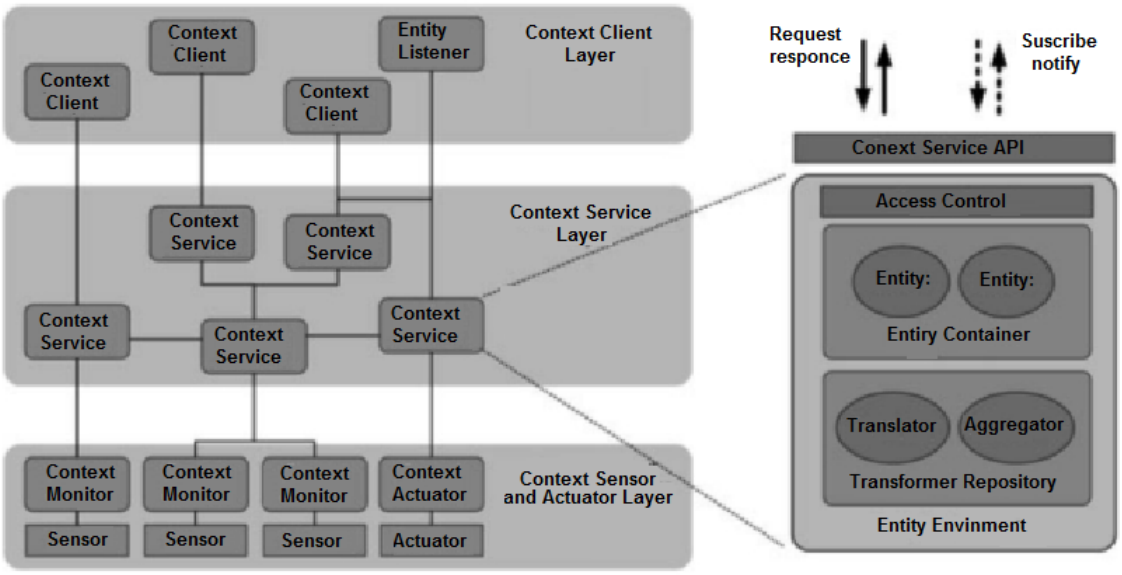

Figura 5.13: Arquitetura do *JCAF*. (BARDRAM, 2005)

Elemento JCAF	**Elemento RA-Ubi**
Atuador	Atuadoro
Sensor	Sensor
Monitor de Contexto	Serviço de Contexto
Atuador de Contexto	Serviço de Atuação
Serviço de Contexto	Repositório de Contexto Módulo de Composição
Listener de Entidades	Módulo de Eventoso
Cliente de Contexto	Aplicação

Tabela 5.8: Relação entre Elementos da Arquitetura *JCAF* com Elementos da RA-Ubi

e/ou atuadores. Na ***Context Service***, o elemento análogo possui uma organização interna dividida em três elementos: (i) ***Access Control***, que limita os acessos aos serviços de acordo com as permissões dos clientes; (ii) ***container*** de entidade, que armazena as entidades de contexto e suas informações de contexto associadas; e (iii) ***Transformer Repository***, que armazena transformadores que podem ser implementados pelo usuário para realizar transformações em informações de contexto em tempo de execução. Por fim, a camada ***Context Client*** possui implementações de clientes de contexto, que acessam entidades e suas respectivas informações de contexto, através de um mecanismo *publish-subscribe* implementado pelo elemento *Listener* de Entidade ou por acesso direto aos servidores de contexto.

Quando relacionando os elementos da arquitetura do JCAF com os elementos da arquitetura RA-Ubi, os componentes do JCAF apresentam comportamento semelhante aos elementos da arquitetura de referência proposta conforme exibido pela Tabela 5.8.

Na Tabela 5.8, destaca-se o elemento Serviço de Contexto, que apesar de possuir um nome idêntico a um elemento da arquitetura RA-Ubi, possui uma funcionalidade bem diferente. Enquanto o serviço de contexto no JCAF armazena os dados de contexto e realiza operações de composição sobre essas informações, essas funcionalidade na arquitetura RA-Ubi estão divididas em dois elementos: Repositório de Contexto e Módulo de Composição.

5.1.4.7 Estudo 6: *SOCAM*

SOCAM (E16) (GU et al., 2004) possui uma arquitetura estilo cliente-servidor na implementação de um *middleware* orientado a serviços para construção e prototipagem de sistemas móveis sensíveis ao contexto direcionados aos carros inteligentes. Sua arquitetura (Figura 5.14) é composta basicamente por seis elementos: (i) ***Context Providers***, responsável por recuperar informação de contexto dos sensores e disponibilizá-la através de uma interface de software; (ii) ***Context Interpreter***, responsável por manipular informações de contexto e inferir novas informações de contexto, esses provedores de contexto dividem-se em dois tipos: internos e externos, nos quais provedores internos fazem parte do sistema, e os provedores externos são acessados através de interfaces web; (iii) ***Context Database***, que consiste de um simples banco de dados para as informações de contexto; (iv) ***Service Location Service***, que consiste de um elemento para localização dinâmica de serviços de contexto; (v) ***Vehicle Satety Service***, que monitora aspectos de segurança do veículo, baseado nas informações de contexto; por fim (vi) ***Context-ware Comum Service***, que variam de acordo com as regras de negócio da aplicação.

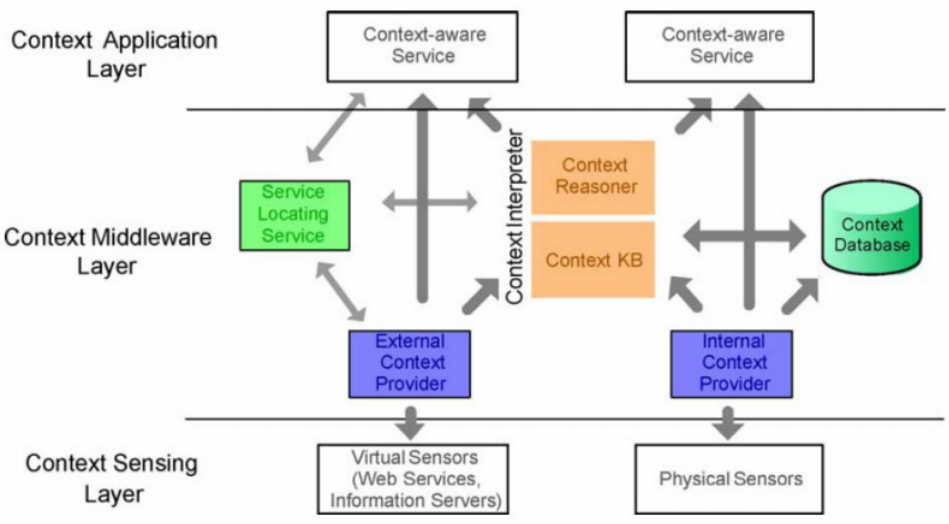

Figura 5.14: Arquitetura do *SOCAM*. (GU et al., 2004)

Elemento SOCAM	**Elemento RA-Ubi**
Provedor de Contexto	Servidor de Contexto
Intérprete de Contexto	Módulo de Raciocínio Módulo de Composição
Repositório de Contexto	Repositório de Contexto
Serviços Sensíveis ao Contexto	Módulo de Eventos
Serviço de Segurança	Módulo de Segurança
Serviços Sensíveis ao Contexto	Módulo de Acoplamento e Mobilidade

Tabela 5.9: Relação entre Elementos da Arquitetura SOCAM com Elementos da RA-Ubi

O ***Context Interpreter*** recolhe informação contextual a partir do ***Context Providers*** e ***Context Database***, fornece-a aos serviços sensíveis ao contexto móvel e o serviço de localização de serviço. O maior diferencial do SOCAM é seu ***Context Interpreter***, que possui um módulo de raciocínio que utiliza ontologias para a descrição do contexto e permite um raciocínio robusto do contexto. Ele usa dois tipos de ontologias: domínio ontologias específicas e generalizadas. Vários sistemas de raciocínio podem ser incorporados no interpretador de contexto para da suporte a uma grande variedade de regras de raciocínio.

Como fica evidente na arquitetura do *SOCAM*, vários de seus componentes possuem um correspondente na arquitetura de referência proposta. A Tabela 5.9 relaciona os elementos da arquitetura do SOCAM aos da arquitetura de referência RA-Ubi, fornecendo indícios de que os elementos da arquitetura de referência são suficientes para a descrição completa desse *middleware*.

Na Tabela 5.9, descata-se o Intérprete de Contexto, que se mostrou como elemento mais notório da arquitetura SOCAM e possui funcionalidades já previstas pela arquitetura RA-Ubi, embora essas funcionalidades estejam separadas em dois dos módulos propostos pela arquitetura de referência: os módulos de raciocínio e composição.

5.2 Considerações Finais

Este capítulo apresentou RA-Ubi, uma arquitetura de referência para sistemas ubíquos, seguindo o processo ProSa-RA, no qual foi feito o estabelecimento dos requisitos arquiteturais, o projeto da arquitetura de referência e sua validação. No próximo capítulo será apresentada a linguagem de descrição arquitetural, UbiAcme, para permitir a especificação arquitetural de aplicações ubíquas.

Capítulo 6

UbiACME - Uma Linguagem de Descrição Arquitetural para Sistemas Ubíquos

6.1 Introdução

Linguagens de Descrição Arquitetural (*Architecture Description Languages - ADLs*) (NADA,) são usadas para especificação de documentos de arquitetura de software e permitem a descrição de sistemas através de representação de componentes, conectores e configuração, responsável por estabelecer como os componentes comunicam-se através dos conectores. Dentre as ADLs existentes, ACME (GARLAN; MONROE; WILE, 1997) destaca-se por ser uma linguagem genérica, com uma extensão que permite definição de restrições a partir de expressões em lógica de primeira ordem (MONROE, 1998), permitindo também a definição de restrições no nível de estilo (ou família) arquitetural. Embora os elementos originalmente descritos na ontologia de ACME sejam suficientes para definir a estrutura de uma arquitetura de software, a natureza das características arquiteturais de aplicações ubíquas vai além do que é atualmente suportado nessa ADL e em outras da literatura. Nesse contexto, faltam modelos que capturem totalmente informações específicas de Computação Ubíqua, tais como: a representação das informações de contexto e a qualidade das informações de contexto relacionadas aos serviços sensíveis ao contexto usados pela aplicação.

O objetivo deste Capítulo é apresentar uma linguagem de descrição arquitetural *(Architecture Description Language - ADL)*, UbiACME, para permitir a especificação arquitetural de aplicações ubíquas. Este Capítulo está estruturado da seguinte forma. A Seção 6.2 apresenta identificação dos elementos a serem providos pela ADL. A Seção 6.3 des-

creve os elementos de UbiACME. A Seção 6.4 apresenta a especificação de uma aplicação Ubíqua em UbiACME: *SmartCar*.

6.2 Identificação dos Elementos a Serem Providos pela ADL

O Capítulo 4 apresentou uma revisão sistemática que identificou um conjunto de 10 elementos comuns aos sistemas ubíquos em geral. Tais elementos foram comparados com as características essenciais estabelecidas no trabalho de SPÍNOLA e TRAVASSOS (2012), ilustrados na Tabela 5.2, e identificou-se que os elementos relacionados atendem às características essenciais de sistemas ubíquos.

As características listadas pela Tabela 5.2 foram utilizadas como base para este trabalho, no sentido de fornecer uma fundamentação acerca de quais recursos deveriam ser providos pela ADL. Em termos de representação no nível arquitetural é necessário expressar os aspectos relacionados com a aplicação, que deve representar e utilizar informações relacionadas aos contextos. Nesse sentido, será discutido a seguir quais as características listadas na Tabela 5.2 que são necessárias, no nível arquitetural, para descrever aplicações ubíquas.

A característica de **Onipresença de Serviço** está diretamente associada à plataforma de implementação usada pelo sistema, pois depende de como os serviços estão instalados nos ambientes e da interface que esses serviços provêm. Portanto, não é representada na descrição arquitetural da aplicação. Da mesma forma, **Invisibilidade** também não é representada, por estar diretamente associada aos elementos físicos e plataforma de execução que compõem o sistema, uma vez que essa característica está relacionada a forma como é feita a interação com o usuário, podendo ser explícita (através do uso de teclado, mouse ou painéis) ou implícita (através de sensores). **Sensibilidade ao Contexto** deve ser representada na arquitetura como elemento de primeira ordem, uma vez que o contexto influencia o comportamento do sistema, conforme discutido em LOPES e FIADERO (2005) e, portanto, é necessário a sua representação explícita. **Comportamento Adaptativo** também deve ser representado no nível arquitetural, dado que a especificação deve usar mecanismos de adaptação arquitetural para dar suporte a adaptação. A representação de contextos e associação explícita do efeito da mudança de contexto sobre a arquitetura do sistema é a forma mais natural de representar essa característica em nível de arquitetura de software. **Captura de Experiência** pode ser representada, entretanto, não será foco no presente trabalho por envolver conceitos relacionados com inteligência artificial.

A **Descoberta de Serviço** deve ser representada no nível arquitetural, pois envolve mecanismos de conexões dinâmicas que relacionam contextos (que devem ser representados como elementos de primeira ordem) a mecanismos de comunicação em si. Não existem, entretanto, abordagens consolidadas para representação de descoberta de serviço em nível arquitetural. **Composição de Função** é uma característica que pode ser representada no nível de ADL através de composição de componentes. **Heterogeneidade de Dispositivos** é naturalmente representada no nível de ADL, uma vez que os elementos de processamento, serviços, sensores, atuadores e todos os demais elementos relacionados a computação ubíqua são representados como componentes e conectores. Por fim, **Tolerância a Falhas** pode ser representada a nível de ADL como um conceito transversal, entretanto, encontra se fora do escopo deste trabalho.

UbiACME	**Descrição**	**RA-Ubi**
Context	Representar contextos no nível arquitetural	Repository Context Context Server
ContextExpression	Descrever conjunto de condições que caracterizam as circunstâncias que ativam um contexto (Context).	Context Server
OnActivate	Descrever um conjunto de adaptações arquiteturais que serão realizadas no instante em que o Context for ativado.	Context Server
OnDeactivate	Descrever um conjunto de aptações arquiteturais que serão realizadas no instante em que o Context for desativado. Esse elemento não é obrigatório no corpo de um Context.	Context Server
Undo	Desfazer todas as adaptações realizadas por um OnActivate, com exceção das adaptações persistentes.	Context Server
Persistent	Indica que a adaptação arquitetural realizada ao ativar um Context persistirá mesmo após a desativação desse Context.	Context Server
QoSParameter	Representar parâmetros de qualidade de serviço que estão associados a um provedor de serviço (componente ou conector).	Context Server
QoCParameter	Representar parâmetros de qualidade de contexto que estão associados a uma interface (port ou role).	Context Server Repository Context
Attachment Condicional	Attachment que só está ativo "i.e. permite o fluxo de informações" em determinadas situações que são definidas a partir dos contextos ativos e inativos.	Adaptation Module Event Module
On .. do ..	Especificar uma condição de adaptação para o sistema e descreve as alterações (adaptações) arquiteturais que serão realizadas quando a condição de adaptação acontecer.	Adaptation Module Event Module
detach	Destrói conexões existentes entre componentes do sistema.	Adaptation Module
remove	Destrói componentes, conectores ou propriedades do sistema.	Adaptation Module

Tabela 6.1: Elementos de UbiACME, sua Funcionalidade e a Relação com ele. de RA-Ubi.

A Tabela 6.1 lista todas as novas abstrações introduzidas na linguagem, que serão discutidas ao longo desta seção, apresentando um breve resumo de sua funcionalidade e a relação com os elementos de RA-Ubi. Por exemplo, o elemento *Context* de UbiACME está relacionado com os elementos *Repository Context* e *Context Server* de RA-Ubi, pois servem para prover informações de contexto e implementá-las no repositório de contexto. Já os elementos de UbiACME *ContextExpression, OnActivate, OnDeactivate, Undo, Persistent* estão relacionados com o elemento *Context Server* de RA-Ubi, pois servem para prover informações de contexto. O elemento *QoSParameter* de UbiACME está relacionado com elemento *Context Server*, pois servem para prover a qualidade de serviço de

contexto. O elemento *QoCParameter* de UbiACME está relacionado com os elementos *Context Server e Repository Context* de RA-Ubi, pois representa a qualidade da informação de contexto armazenada no repositório de contexto. O elemento *Attachment* Condicional de UbiACME está relacionado com os elementos *Adaptation Module* e *Event Module* de RA-Ubi, pois verifica o evento que é verdadeiro para fazer o *attachment* de acordo com essa informação. O elemento *On .. do ..* de UbiACME está relacionado com os elementos *Adaptation Module* e *Event Module* de RA-Ubi, pois verifica o evento que é verdadeiro e especifica uma condição de adaptação para o sistema e descreve as adaptações arquiteturais que serão realizadas. Os elementos *detach e remove* de UbiAcme estão relacionado com o elemento *Adaptation Module* de RA-Ubi, pois este modulo é responsável por alterar o comportamento do sistema.

6.3 Elementos de UbiACME

A linguagem UbiACME foi definida de modo a possibilitar a descrição de aplicações ubíquas. Dentre as abstrações apresentadas por UbiACME, destacam-se os elementos para: (i) representação de contexto, (ii) representação de meta-dados, dados sobre serviços e qualidade de informação de contexto, (iii) reconfiguração dinâmica. Esta seção discute as alterações realizadas por UbiACME, dividindo-as em quatro tópicos principais: **Contexto**, **Qualidade de Serviço e de Contexto**, ***Attachment* Condicional** e **Reconfiguração Dinâmica**.

6.3.1 Contexto

Aplicações ubíquas são sensíveis ao contexto, uma vez que consideram o contexto em que executam e adaptam-se às mudanças que ocorrem no contexto. LOPES e FIADEIRO (2005) discutem sobre a importância da representação do contexto como elemento de primeira ordem na arquitetura, sob a ótica de que a arquitetura do sistema depende do contexto no qual ele se encontra. Por isso, se fazem necessários mecanismos para representação de contexto que permitam o suporte à dinamicidade ao nível arquitetural, essencial em sistemas cientes ao contexto. Uma vez que sistemas ubíquos enquadram-se nessa classificação de sistemas, é fundamental representar dados de contexto a nível arquitetural. Para tal, UbiACME define elementos que permitem a descrição de contextos e o seu uso para especificação de reconfiguração e adaptações dinâmicas.

A Figura 6.1 mostra a definição sintática para o elemento *Context*, definido essencial-

```
1 <ContextDefinition> ::= Context ID = "{"
        <ContextExpressionDef> ";"
        [<OnActivateDef>";"] [<OnDeactivateDef> ";"]
        (<PropertyDefinition>";")* "}" "[;]"
2 <ContextExpressionDef> ::= ContextExpression <Expression> ";"
3 <OnActivateDef>        ::= OnActivete "{"
        ( [Persistent] <ElementDef>
        |(<PlastickExpression>)* "}"
4 <OnDeactivateDef>      :: = OnDeactivate "{"
        [Undo";"]
        (<PlastickExpressionDef>)* "}"
```

Figura 6.1: Definição Sintática para o Element Context

mente a partir de uma *ContextExpression*. Contextos também podem possuir propriedades (como qualquer elemento ACME) e permitem definições de *OnActivate* e *OnDeactivate*, que serão explicados a seguir.

6.3.1.1 Elemento Context

Contextos são representados em UbiACME através do elemento *Context* o qual possui essencialmente uma expressão de ativação de contexto, que é uma expressão booleana envolvendo propriedades, parâmetros de qualidade e estados de outros contextos (ativo ou inativo) que, quando satisfeita, representa a ativação de um contexto, ou seja, o sistema encontra-se em um conjunto de circunstâncias pré-definidas. Assume-se que todo e qualquer contexto está desativado quando sua expressão de ativação não é satisfeita. O elemento *Context*, que representa contextos no nível arquitetural, possui um *ContextExpression*.

6.3.1.2 Elemento *ContextExpression*

ContextExpression,é responsável por descrever um conjunto de condições através de expressões booleanas que caracterizam as condições que ativam um *Context*, utilizando propriedades, parâmetros de qualidade ou até o estado (ativo/inativo) de outros *Contexts*.

6.3.1.3 Elemento *OnActivate*

O elemento *OnActivate* descreve um conjunto de adaptações arquiteturais que serão realizadas no momento em que o *Context* for ativado. Essas adaptações incluem definição de novos elementos, criação ou eliminação de conexões e alterações no comportamento de elementos existentes.

6.3.1.4 Elemento *OnDeactivate*

O elemento *OnDeactivate* descreve um conjunto de adaptações arquiteturais que serão realizadas no instante em que o *Context* for desativado. Esse elemento não é obrigatório no corpo de um *Context*, e por padrão todas as adaptações realizadas por um *OnActivate* são desfeitas no momento em que o *Context* é desativado. Na definição de um *OnDeactivate*, é possível incluir o comando *Undo*.

6.3.1.5 Elemento *Undo*

O elemento *Undo* serve para desfazer todas as alterações feitas por um *OnActivate* com exceção das adaptações persistentes.

6.3.1.6 Elemento *Persistent*

Elementos *Persistent* são uma alternativa mais simples para o *OnDeactivate* em situações nas quais o arquiteto não quer que todas as adaptações arquiteturais sejam perdidas ao desativar o *Context*. Durante a definição do elemento *OnActivate* é possível utilizar a palavra-chave *Persistent* para discriminar as adaptações que irão persistir mesmo após a desativação do *Context*.

```
17  Context Highway_Entering = {
18    ContextExpression not(Highway_Driving) AND NavigationSystem.locationId=HIGHWAY;
19    OnActivate {
20      Persistent Component DrivingSystem = {
23        Property isReady : boolean;
24        Port wheelController;
25        Port speedController;
26        Port drivingEvents;    }
27      Attachment DrivingSystem.drivingEvents to EventConnector.eventProvider;
28      EventMonitor.synchronizeNav = true; }}
```

Figura 6.2: Exemplo de Elemento Context

A Figura 6.2 mostra um exemplo de definição de Contexto para a aplicação *Smart Car*, na qual é definido o contexto *Highway_ Entering*, que representa a situação na qual

o *Smart Car* encontra-se entrando em uma autoestrada, o que significa que os módulos de piloto automático não estão prontos e precisam ser carregados. Esse contexto estará ativo quando o contexto *Highway_Driving* não estiver ativo e o sistema de navegação indicar que o veículo está em uma autoestrada. Todos os demais elementos do exemplo serão definidos com mais detalhes no cenário 1.

6.3.2 Qualidade de Serviço e de Contexto

A Figura 6.3 exibe a definição sintática para os elementos *QoSParameter* e *QoCParameter*, que consistem apenas da palavra-chave *QoSParameter* ou *QoCParameter* seguido de um nome identificador. É possível definir um tipo e o valor padrão para cada parâmetro. Qualquer tipo de *Property* pode ser atribuído aos construtores *QoSParameter* e *QoCParameter*, uma vez que esse tipo será responsável por definir o tipo de valor que esse parâmetro armazena. Quando especificado, o valor padrão de um *QoSParameter* ou *QoCParameter* deve estar de acordo com o tipo definido.

```
<QoSParameterDef>::=
   QoSParameter ID [":" <PropertyTypeID>] ["=" <value>]
";"
<QoCParameterDef>::=
   QoCParameter ID [':" <PropertyTypeID>] ["=" <value>]
";"
```

Figura 6.3: Definição Sintática dos Elementos QoSParameter e QoCParameter

Ambos, *QoSParameter* e *QoCParameter*, estendem a abstração *Property*, adicionando as características de dinamicidade e monitoramento contínuo, além de explicitar que os valores armazenados em *QoSParameters* e *QoCParameters* serão calculados pela aplicação em tempo de execução. Quando não for possível calcular o valor do parâmetro, o valor padrão é atribuído. Elementos *QoSParameter* podem ser instanciados em elementos arquiteturais que podem ser usados para representar provedores de serviço, que tipicamente são elementos *Component* ou *Connector*. Elementos *QoCParameter*, por sua vez, só podem ser instanciados em interfaces (*Port* ou *Role*), devido a sua natureza de estarem diretamente associados aos dados de serviços, que são necessariamente providos através de interfaces.

6.3.2.1 Elemento *QoSParameter*

O elemento *QoSParameter* permite estabelecer quais são os medatados de qualidade de serviços das informações de contexto de um serviço.

6.3.2.2 Elemento *QoCParameter*

O elemento *QoCParameter* permite que, para um dado parâmetro de contexto essencial à aplicação, estabeleçam-se os critérios de qualidade relacionados a ele.

A Figura 6.4 mostra um exemplo de utilização de um *QoSParameter* e *QoCParameter*.

Como exemplo, na Figura 6.4, define-se o *Component Type LocationService*, que é o tipo básico dos componentes provedores de serviço de localização. Esse tipo de componente possui: (i) um *QoSParameter availability*, responsável por monitorar a disponibilidade do servidor; (ii) uma interface para o serviço *provideLocation*, essa interface (*Port*) possui um *QoCParameter precision*, que se refere à precisão do dado que é fornecido pelo serviço; e (iii) uma propriedade *syncInterval*, que define o intervalo de atualização dos parametros de qualidade.

```
1 Component Type LocationService = {
2    QoSParameter availability;
3    Port provideLocation = {
4          QoCParameter precision;   }
5    Property syncInterval = 30;
6 }
```

Figura 6.4: Exemplo de uso de QoSParameter e QoCParameter

6.3.3 *Attachment* Condicional

O *Attachment* Condicional tem comportamento semelhante ao *Attachment* nativo de ACME. Entretanto, só está ativo, ou seja, permite o fluxo de informações em situações que são definidas a partir de uma expressão de contexto, a qual consiste em uma combinação de estados (ativo ou inativo) de contextos pré-definidos.

6.3.3.1 Elemento *Attachment* Condicional

A Figura 6.5 exemplifica um *Attachment* condicional, definindo três *attachments* que se encontram ativos apenas quando em contextos específicos estiver ativo. Nessa Figura, os

contextos *UsingGPSLocation*, *UsingGSMLocation* e *UsingWifiLocation* são ativados a depender principalmente do parâmetro de QoC *(QoCParameter) precision*. Da forma como foram definidos, no máximo um desses contextos pode estar ativo por vez. A partir do estado desses contextos (ou seja, ativo ou inativo), são realizados *attachments* condicionais que relacionam o provedor do serviço de localização (que pode ser GPS, GSM ou Wifi) ao conector *Location*, que conecta o serviço de localização aos elementos que utilizam esse serviço.

```
01 Context UsingGPSLocation = {
02   ContextExpression LocationGPS.error != 0  AND
03         LocationGPS.provideLocation.precision >= LocationGSM.provideLocation.precision AND
04         LocationGPS.provideLocation.precision >= LocationWifi.provideLocation.precision;
05 }

06 Context UsingGSMLocation = {
07    ContextExpression LocationGSM.error !=0 AND
08          LocationGSM.provideLocation.precision >  LocationGPS.provideLocation.precision AND
09          LocationGSM.provideLocation.precision >= LocationWifi.provideLocation.precision;
10 }

11 Context UsingWifiLocation = {
12     ContextExpression LocationWifi.error != 0  AND
13           LocationWifi.provideLocation.precision >  LocationGPS.provideLocation.precision AND
14           LocationWifi.provideLocation.precision >  LocationGSM.provideLocation.precision;
15 }

16 Attachment LocationGPS.provideLocation to Location.provider  [UsingGPSLocation];
17 Attachment LocationGSM.provideLocation to Location.provider  [UsingGSMLocation];
18 Attachment LocationWifi.provideLocation  to Location.provider  [UsingWifiLocation];
```

Figura 6.5: Exemplo de *Attachment* Condicional

6.3.4 Reconfiguração Dinâmica

Devido à natureza dinâmica de aplicações ubíquas, faz-se necessária a inclusão de mecanismos arquiteturais de planejamento de reconfiguração dinâmica programada. Originalmente, ACME/Armani não oferecem recursos para se descrever o planejamento de reconfigurações programadas, por isso, foram adicionadas à UbiACME as extensões propostas por *Plastik* (BATISTA; JOOLIA; COULSON, 2005): *On-Do; Detach; Remove.*

6.3.4.1 Elemento *On-Do*

Especifica uma condição de adaptação para o sistema e descreve as alterações (adaptações) arquiteturais que serão realizadas quando a condição de adaptação acontecer, permitindo ao arquiteto expressar em qual situação a reconfiguração programada deve ocorrer e o que deve ser mudado para efetivar essa reconfiguração.

Na Figura 6.6, o comando condicional *On-Do* (linha 1) verifica se os três contextos

relacionados ao serviço de localização não estão ativos. Se essa condição for verdadeira, significa que nenhum serviço de localização está sendo utilizado, então deve ser realizada uma reconfiguração que consiste em incluir um provedor de localização padrão e associar seu serviço de localização a *Role Provider* do *Connector Location*.

```
01 On (not(UsingGPSLocation) and not(UsingGSMLocation) and not(UsingWifiLocation)) do {
02         Component DefaultLocationProvider : LocationService;
03         Attachment DefaultLocationProvider.providerLocation to Location.provider;
04 }
```

Figura 6.6: Exemplo da Utilização do Comando *On-do*

6.3.4.2 Elemento *Detach*

O elemento *Detach* destrói conexões existentes entre componentes do sistema, construtor que permite a alteração da arquitetura, utilizado para remover uma ligação entre uma *port* e uma *role*.

6.3.4.3 Elemento *Remove*

O elemento *Remove* é usado para destruir do sistema, um componente, um conector, uma representação, uma propriedade ou os elementos QoSParameter e QoCParameter.

A BNF completa da ADL UbiACME está disponível no endereço:

http://consiste.dimap.ufrn.br/projects/ubiacme/

6.4 Aplicação Ubíqua em UbiACME: *Smart Car*

De modo a verificar as caracteristicas da linguagem para especificar um sistema ubíquo, foi usado a aplicação de *Smart Car*, baseada em propostas existentes para um veículo autonômico. No setor automotivo, os carros autônomos são dotados de sistemas embarcados que caracterizam um sistema computacional com propósitos específicos (HORN, 2001). O carro pode enviar sinais a outros automóveis automaticamente para indicar as condições da estrada à frente ou a necessidade de frear, por exemplo.

A arquitetura do sistema *Smart Car*, nesse estudo de caso, possui sete componentes: (i) **subsistema de condução**, responsável pelo piloto automático; (ii) **subsistema de navegação**, responsável por controlar os elementos físicos do carro, como por exemplo

alterando sua velocidade e direção. Esse subsistema monitora dados do veículo, do clima e da pista que serão exibidos para o usuário e utilizados pelo sistema na tomada de decisão; (iii) **mecanismo de pilotagem**, responsável por captar as instruções do usuário, através de sensores no volante e pedais; (iv) **componente de comunicação via rádio**, responsável pela interação com outros sistemas semelhantes; (v) **subsistema de localização**, responsável por monitorar a localização do veículo; (vi) **serviços de localização**, responsáveis por prover a localização, esses serviços são implementados nos próprios elementos de hardware do sistema; e (vii) **módulo de eventos**, responsável por tratar requisições assíncronas, disparadas segundo alguma condição pré-estabelecida. Para essa aplicação, foram definidos inicialmente dois cenários principais usados como fundamento para a descrição arquitetural. Esses cenários definem situações nas quais o *Smart Car* poderá estar inserido (contexto), como também descreve ações que devem ser tomadas pelo sistema computacional do veículo. No primeiro cenário, o *Smart Car* deve ser capaz de ativar o piloto automático quando o veículo entra em uma autoestrada. É importante ressaltar que algumas propostas definem o *Smart Car* como um carro totalmente automático (KARVONEN; KUJALA, 2006). Entretanto, optou-se por descrever uma proposta mais realista, que é um veículo semi-autônomo. O segundo cenário descreve o uso e sincronização do Serviço de Localização utilizado pelo veículo, que se mostrou a situação mais comum nesse tipo de aplicação ciente de contexto.

6.4.1 Cenário 1. Piloto Automático em Auto-Estradas

Definição: O veículo entra em uma autoestrada, o sistema do *Smart Car* recupera as informações da estrada, como: tráfego (fluxo de veículos), velocidade máxima, condições climáticas (temperatura, umidade, etc), condições da pista (acesso, visibilidade, etc), entre outros. Enquanto isso, o subsistema de condução, que irá assumir o controle da velocidade e direção do carro, é preparado e sincronizado com o subsistema de navegação, responsável por controlar os elementos físicos do carro e monitorar os elementos de navegação, como posicionamento e condições dos equipamentos. Quando ambas as etapas estão concluídas, o sistema desabilita as conexões do mecanismo de pilotagem, responsável por captar os controles do usuário através da direção e pedais, ativando assim o piloto automático.

No piloto automático, o sistema usa baixas frequências de rádio para se comunicar com os carros próximos.

Descrição: Para esse cenário, o sistema reage a dois contextos: ***Highway_Entering***, que inicializa os módulos do piloto automático e é ativado quando o carro entra em

uma auto-estrada e o componente responsável pelo piloto automático não está pronto para uso; e ***Highway_ Driving***, que ao ser ativado inicializa o monitor de eventos com a comunicação pelo rádio e ativa os módulos do piloto automático para controlar toda navegação do carro. *Highway_ Driving* é ativado quando o carro está em uma autoestrada e o componente responsável pelo piloto automático já está inicializado. Enquanto o contexto *Highway_ Driving* estiver ativo, o carro está sendo controlado automaticamente.

A Figura 6.7 mostra a representação gráfica da descrição arquitetural, em UbiACME, do conjunto de elementos arquiteturais responsáveis pelo tratamento do cenário 1. Nessa representação existem três componentes fundamentais, que são os componentes que representam os elementos físicos dos pedais e do volante (*Wheel e Pedals*), e o componente que representa o subsistema de navegação (*NavigationSystem*), que atua sobre o motor, freios e rodas do veículo para realizar as operações de mudança de direção ou velocidade. O conector *PilotingMechanism* é responsável pela comunicação entre os controladores do veículo e o atuador *NavigationSystem*, para que o veículo responda de acordo com os comandos.

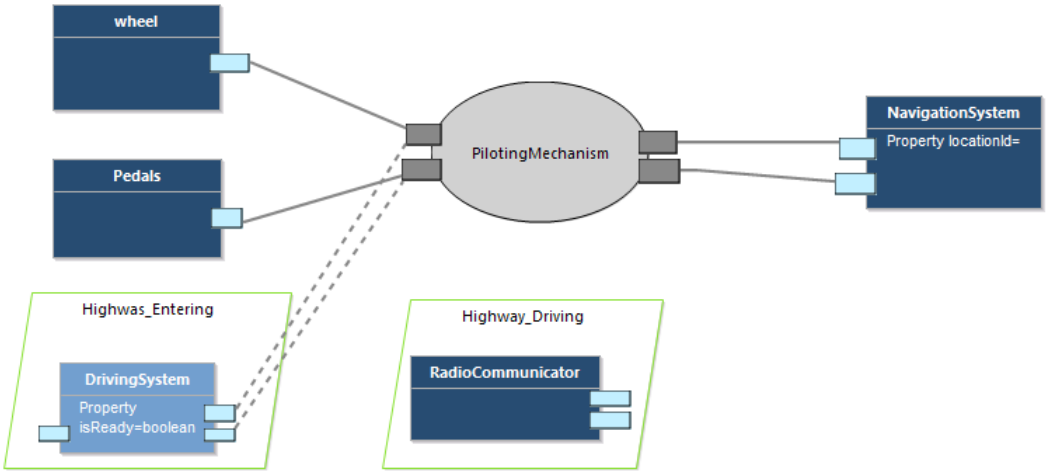

Figura 6.7: Representação Gráfica da Descrição do Cenário 1

A Figura 6.7 usa a linguagem de comunicação gráfica de UbiACME e foi modelada usando *UbiACME studio* uma ferramenta de modelagem gráfica para projetos de sistemas Ubíquos, apresentada no capítulo 7. Na representação gráfica, os contextos *Highway_ Entering* e *Highway_ Driving* são descritos. O contexto *Highway_ Entering* define um componente persistente, *DrivingSystem*, que será conectado ao *PilotingMechanism* e assumirá o controle do carro durante o contexto *Highway_ Driving*. O contexto *Highway_ Driving*, por sua vez, define um componente *radioCommunicator*, que será uti-

lizado para comunicação com outros sistemas semelhantes que se encontram próximos.

```
1  System Smart_Car = {
2     Component NavigationSystem = {
3          Property locationId;
4          Port wheelControl;
5          Port speedControl; }
6     Component Wheel = { Port wheel; }
7     Component Pedals = { Port pedals; }
8     Connector PilotingMechanism = {
9          Role pilotWheel;
10         Role carWheel;
11         Role pilotPedals;
12         Role carPedals; }
13    Attachment PilotingMechanism.carWheel to NavigationSystem.wheelControl;
14    Attachment PilotingMechanism.carPedals to NavigationSystem.speedControl;

15    Attachment Wheel.wheel to PilotingMechanism.pilotWheel [not(Highway_Driving)];
16    Attachment Pedals.pedals to PilotingMechanism.pilotPedals [not(Highway_Driving)];
```

Figura 6.8: Definição dos Elementos Principais da Arquitetura Responsável por Implementar o Cenário 1

A Figura 6.8 mostra a descrição arquitetural, em UbiACME, do conjunto de elementos arquiteturais responsáveis pelo tratamento do cenário 1. A Linha 1 descreve o *System Smart_ Car*, mostrando o nome da aplicação. A Linha 2 descreve o *Component NavigationSystem*, no corpo do componente. A Linha 3 descreve o elemento *Property locationId* e as linhas (4 e 5) descrevem as portas *wheelControl* e *speedControl*. Este componente atua sobre o motor, freios e rodas do veículo para realizar as operações de mudança de direção ou velocidade. A Linha 6 define o *Component Wheel*, que representa os elementos físicos do volante, no corpo do componente, também na Linha 6 é definida a porta *wheel*. A Linha 7 define o componente *Pedals*, que representa os elementos físicos dos pedais. O corpo do componente, também na Linha 7 descreve a porta *pedals*. A Linha 8 descreve o conector *PilotingMechanism*, no corpo do conector, linha (9 a 12), descreve os papéis, com os elementos *Role pilotWheel*; *Role carWheel*; *Role pilotPedais*; *Role carWheel*, respectivamente para permitir a ligação do conector aos componentes. Nas Linhas (13 a 14) são descritos os elementos *Attachment ...*, ligando os papeis do conector *PilotingMechanism* as portas do componente *NavigationSystem*, estes *Attachment ...* é o elemento nativo do ACME. As Linhas (15 a 16) descrevem os elementos *Attachment ...*, ligando os papéis do conector *PilotingMechanism* as portas dos componentes *Wheel* e *Pedals*, respectivamente. Esse *Attachment ...* é um *Attachment* condicional de UbiACME, pois a ligação só será feita se o contexto *Highway_ driving* não estiver ativo.

A Figura 6.9 mostra um exemplo de definição de Contexto para a aplicação *Smart Car*, na qual é definido o contexto *Highway_ Entering*, que representa a situação na qual o *Smart Car* encontra-se entrando em uma autoestrada, o que significa que os módulos

```
17 Context Highway_Entering = {
18   ContextExpression not(Highway_Driving) AND NavigationSystem.locationId=HIGHWAY;
19   OnActivate {
20     Persistent Component DrivingSystem = {
23       Property isReady : boolean;
24       Port wheelController;
25       Port speedController;
26       Port drivingEvents;    }
27     Attachment DrivingSystem.drivingEvents to EventConnector.eventProvider;
28     EventMonitor.synchronizeNav = true; }}
```

Figura 6.9: Descrição do Contexto Highway_Entering

de piloto automático não estão prontos e precisam ser carregados. Esse contexto estará ativo quando o contexto *Highway_Driving* não estiver ativo e o sistema de navegação indicar que o veículo está em uma auto-estrada. O contexto *Highway_Driving* ativo, por sua vez, representa a circunstância na qual o *Smart Car* está em modo de piloto automático, ativado em autoestradas. Quando o contexto *Highway_Entering* for ativado, será definido um componente persistente, *DrivingSystem*, que possui uma propriedade booleana *isReady* e três portas: *wheelController*, responsável por enviar sinais de controle de volante; *speedController*, responsável por enviar sinais de controle de velocidade; e *drivingEvents*, responsável por receber eventos e dados de elementos externos. *DrivingSystem* foi definido como um componente persistente pois implementa um complexo subsistema de Inteligência Artificial e, por isso, demanda um tempo maior para ser totalmente inicializado. Esse componente é responsável pelos controles automáticos do carro (o piloto automático). Implementando-o como componente persistente, descreve-se explicitamente que esse componente só será inicializado quando o contexto *Highway_Driving* se tornar ativo. Entretanto, após a inialização, *DrivingSystem* irá permanecer no sistema, auxiliando o usuário na navegação e controles do veículo e possibilitando a ativação do piloto automático mais rápido em próximas entradas em autoestradas. Uma alternativa seria inicializar o *DrivingSystem* durante as atividades normais do carro. Entretanto, essa solução iria alocar recursos computacionais desnecessariamente, na maioria das situações, remover o *DrivingSystem*, por outro lado, iria requerer uma nova inicialização sempre que o contexto *Highway_Entering* se tornar ativo.

A Figura 6.10 mostra um exemplo de definição de Contexto para a aplicação *Smart Car*, na qual é definido o contexto *Highway_Driving*, que representa a circunstância na qual o *Smart Car* está em modo de piloto automático, ativado em autoestradas. Esse contexto estará ativo quando as variáveis *DrivingSysten.isReady* e *EventMonitor.synsReady* for *True* e o sistema de navegação indicar que o veículo está em uma autoestrada. Na Linha 31, o elemento *OnActivate* descreve as adaptações arquiteturais que serão realizadas

```
29  Context Highway_Driving = {
30    ContextExpression DrivingSystem.isReady AND EventMonitor.syncReady AND
                  NavigationSystem.locationId=HIGHWAY;
31    OnActivate {
32        Attachment DrivingSystem.wheelController to PilotingMechanism.pilotWheel;
33        Attachment DrivingSystem.SpeedCotroler to PilotingMechanism.pilotPedals;
34        Component radioCommunicator = {
35            Port radioEvent;
36            Port radioData; }
37        Attachment radioCommunicator.radioEvent to  EventConnector.eventProvider; }
38  OnDeactivate {
39    Undo;
40    Remove DrivingSystem;
41    Remove AutoPilot; } }
```

Figura 6.10: Descrição do Contexto Highway_Driving

no instante em que o *Context* for ativado. Da Linha (31 a 37), Os *Attachment ...* para fazer a ligação das portas do componente *DrivingSystem* os papéis do conector *Poling-Mechanism*; A definição do componente *radioCommunicator* e suas portas *radioEvent* e *radioData* e o *Attachment ...* para fazer a ligação de suas portas aos papeis dos conectores dos sistemas semelhantes que se encontram próximos. Na linha 38, o elemento *OnDeactivate* descreve as adaptações arquiteturais que serão realizadas já que o *Context* está desativado que são o elemento *Undo*, para desfazer todas as adaptações realizadas pelo *OnActivate* e os *Remove DrivingSystem* e *Remove AutoPilot* para destruir os respectivos componentes do sistema.

6.4.2 Cenário 2. Serviço de Localização

Definição: Em intervalos fixos (30 segundos), o subsistema de localização do veículo sincroniza todos os metadados relativos aos serviços de localização, que podem ser por *GPS, GSM ou Wifi*. A partir desses dados, o sistema identifica o melhor serviço de localização baseado em sua precisão, adaptando-se para utilizar apenas esse serviço. Se nenhum dos serviços estiver disponível, o sistema realiza uma adaptação para inserir um componente que é capaz de fornecer um dado de localização estimado, podendo basear-se nas posições prévias e velocidade do veículo.

Descrição: Para esse cenário foram definidos seis contextos: (i) ***ChoosingLocationProvider***, que será ativado a cada 30 segundos para atualizar os metadados dos serviços de localização (que serão providos pelos próprios serviços) para escolher o serviço a ser utilizado; (ii) ***UsingGPSLocation***, que é ativado quando o sistema escolhe utilizar o mecanismo de localização via *GPS*; (iii) ***UsingGSMLocation***, que é ativado quando o sistema escolhe utilizar o mecanismo de localização via *GSM*; (iv) ***UsingWifiLoca-***

tion, ativado quando o sistema escolhe utilizar o mecanismo de localização via *Wifi*; (v) ***NoLocationProvider***, ativado quando o sistema identifica que nenhum dos servidores de localização está disponível; e finalmente, (vi) ***LocationServiceError***, ativado sempre que o serviço atual de localização entra em um estado de erro.

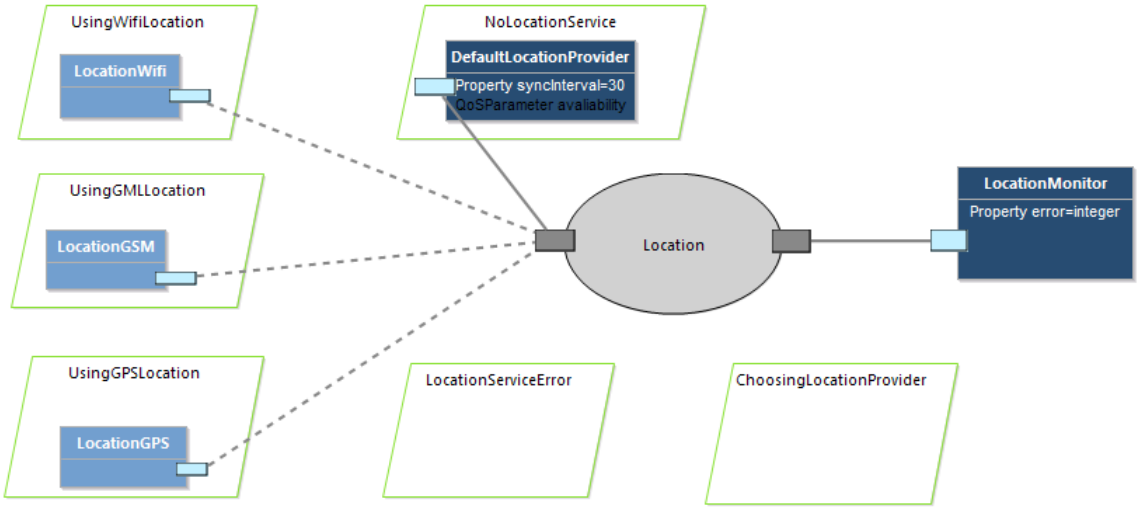

Figura 6.11: Representação Gráfica do Cenário 2

Essencialmente, a arquitetura possui um componente ***LocationMonitor***, responsável por monitorar os dados de localização e repassar as informações para as demais partes do sistema; e um conector *Location*, que realiza a comunicação entre o serviço de localização atual e o *LocationMonitor*. A Figura 6.11 mostra uma representação gráfica para essa descrição e a Figura 6.12 os trechos da descrição UbiACME relativos a esse cenário. Nessa figura é possível observar os componentes ***LocationGPS***, ***LocationGSM*** e ***LocationWifi***, responsáveis por prover o serviço de localização, que estão conectados ao conector ***Location*** por um *attachment* condicional, que é ativo quando o serviço respectivo é escolhido para ser utilizado pelo sistema.

Na Figura 6.13 o contexto *ChoosingLocationProvider* que se refere ao contexto no qual o sistema não escolheu ainda o mecanismo de localização e está atualizando seus dados.

Na Figura 6.14, o contexto *UsingGPSLocation* refere-se à situações na qual o sistema está utilizando o mecanismo de localização via GPS. Este contexto é ativado quando o sistema escolhe utilizar o mecanismo de localização via GPS. Este contexto é ativado pelo parâmetro *QoC (QoCParameter) precision*, isto é, quando a precisão de GPS for maior ou igual as precisões de GMS e *Wi-fi*. Esta precisão é atualizada pelo contexto *ChossingLocatonProvider* que será ativado a cada 30 segundos para atualizar os metadados dos serviços de localização.

```
1  System Smart_car
2  Component Type LocationService = {
3     QoSParameter availability;
4     Port provideLocation = {
5           QoCParameter precision;   }
6     Property syncInterval = 30;
7  }
8  Component LocationMonitor = {
9    Port locationPort;
10     Property error : integer;
11 }
12 Connector Location = {
13     Role provider;
14     Role monitor;
15 }
16 Component LocationGPS : LocationService;
17 Component LocationGSM : LocationService;
18 Component LocationWifi : LocationService ;
19 Attachment LocationMonitor.locationPort to Location.monitor;
20 Attachment LocationGPS.provideLocation to Location.provider
                                        [ChoosingLocationProvider OR UsingGPSLocation];
21 Attachment LocationGSM.provideLocation to Location.provider
                                        [ChoosingLocationProvider OR UsingGSMLocation];
22 Attachment LocationWifi.provideLocation to Location.provider
                                        [ChoosingLocationProvider OR  UsingWifiLocation];
```

Figura 6.12: Descrição dos Principais Elementos Arquiteturais Responsáveis por Implementar o Cenário 2

```
23 Context ChoosingLocationProvider = {
24     ContextExpression syncInterval == 0;
25     OnActivate {   }
26 }
```

Figura 6.13: Descrição do Contexto ChoosingLocationProvider

Na Figura 6.15, o contexto *UsingGSMLocation* refere-se à situações na qual o sistema está utilizando o mecanismo de localização via GSM. Este contexto é ativado quando o sistema escolhe utilizar o mecanismo de localização via GSM. Este contexto é ativado pelo parâmetro *QoC (QoCParameter) precision*, isto é, quando a precisão de GSM for maior que precisões de GPS e maior ou igual a precisão de *Wi-fi*. Esta precisão é atualizada pelo contexto *ChossingLocatonProvider* que será ativado a cada 30 segundos para atualizar os metadados de dos serviços de localização.

Na Figura 6.16, o contexto *UsingWifiLocation* refere-se à situações na qual o sistema está utilizando o mecanismo de localização via *Wi-fi*. Este contexto é ativado quando o sistema escolhe utilizar o mecanismo de localização via *Wi-fi*, este contexto é ativado pelo parâmetro *QoC (QoCParameter) precision*, isto é, quando a precisão de *Wi-fi* for maior que precisões de GPS e maior que a precisão de GSM. Esta precisão é atualizada pelo contexto *ChossingLocatonProvider* que será ativado a cada 30 segundos para atualizar os metadados de dos serviços de localização.

Na figura 6.17, o contexto *NoLocationService*, representa quando nenhum dos me-

```
27  Context UsingGPSLocation = {
28    ContextExpression LocationGPS.error != 0  AND
29              LocationGPS.provideLocation.precision >= LocationGSM.provideLocation.precision AND
30              LocationGPS.provideLocation.precision >= LocationWifi.provideLocation.precision;
31  }
```

Figura 6.14: Descrição do Contexto UsingGPSLocation

```
32  Context UsingGSMLocation = {
33     ContextExpression LocationGSM.error !=0 AND
34              LocationGSM.provideLocation.precision > LocationGPS.provideLocation.precision AND
35              LocationGSM.provideLocation.precision >= LocationWifi.provideLocation.precision;
36  }
```

Figura 6.15: Descrição do Contexto UsingGSMLocation

canismos de localização está disponível. Nesse caso o sistema vai usar um componente *DefaultLocationProvider* que calcula uma aproximação para a posição do veículo a partir de dados passados. No contexto *NoLocationService* é instanciado um novo componente, que é conectado ao conector Location. Esse componente é o *DefaultLocationProvider*, que é responsável por implementar um mecanismo padrão de localização.

Na figura 6.18, o contexto *LocationServiceError*, que representa o momento em que o contexto que está sendo utilizado atualmente entra em um estado de erros. Nesse caso o sistema deve refazer todo o processo de escolha de provedor, resetando as variáveis de controle.

6.5 Considerações Finais

Este capítulo apresentou uma linguagem de descrição arquitetural, UbiACME e a identificação e descrição dos seus elementos, exemplificados com trechos da aplicação *Smart Car*. Também foi apresentado um sistema para aplicação ubíquo em UbiACME, usando dois cenários: Piloto automático em autoestradas e Serviço de localização. Em ambos cenários foram apresentadas as representações gráfica e descrição textual em UbiACME.

Todos os seis elementos de UbiACME que se referem a contexto, *Context, ContextExpression, OnActivate, OnDeactivate, Undo, Persistent*, sofrem influências dos elementos da Revisão Sistemática e da Arquitetura de Referência, tais como:

1. **Sensor**, responsável por prover informações de contexto;
2. **Atuador**, responsável por alterar o ambiente, que é provedor de contexto;
3. **Serviço de Contexto**, responsável por recuperar informações de contexto;

```
37 Context UsingWifiLocation = {
38     ContextExpression LocationWifi.error != 0  AND
39             LocationWifi.provideLocation.precision > LocationGPS.provideLocation.precision AND
40             LocationWifi.provideLocation.precision > LocationGSM.provideLocation.precision;
41 }
```

Figura 6.16: Descrição do Contexto UsingWifiLocation

```
42 Context NoLocationService = {
43     ContextExpression not(ChoosingLocationProvider) AND not(UsingGPSLocation)
                          AND not(UsingGSMLocation)  AND not(UsingWifiLocation);
44     OnActivate {
45     Component DefaultLocationProvider : LocationService;
46     Attachment DefaultLocationProvider.providerLocation to Location.provider;}
47 }
```

Figura 6.17: Descrição do Contexto NoLocationService

4. **Serviço de Atuação**, capaz de prover atuação para o usuário;
5. **Repositório de Contexto**, capaz de armazenar informações de contexto;
6. **Modulo de Evento**, responsável por monitorar todos eventos, tanto na captação como na alteração de contexto;
7. **Modulo de Raciocínio**; capaz de produzir novas informações de contexto;
8. **Modulo de Composição**, capaz de compor informações de contexto;
9. **Modulo de Adaptação**, capaz de alterar o comportamento do sistema de acordo com os eventos disparados;
10. **Mecanismo de Acoplamento e Mobilidade**, capaz de abstrair a noção de ambiente, utilizando ferramentas de rastreamento, busca de serviço e computação móvel.

Já os dois elementos de UbiACME que se referem a qualidade do serviço e do contexto, *QoSParameter e QoCParameter*, também sofrem influências desses elementos citados. O *Attachment* Condicional, por tratar situações que são definidas a partir de expressões de contexto, também sofre influência destes comandos citados. Já os comandos de UbiACME, que se referem a reconfiguração dinâmica, *On-Do, Detach e Remove*, são utilizados devido à natureza dinâmica de computação ubíqua.

O próximo Capítulo apresenta *UbiACME Studio*,uma ferramenta de modelagem gráfica para permitir a descrição de aplicações ubíquas, disponibilizada em forma de *plug-in* para o *Eclipse*.

```
48  Context LocationServiceError = {
49    ContextExpression ((UsingGPSLocation AND LocationGPS.error == 0) OR
                         (UsingGSMLocation AND LocationGSM.error == 0) OR
                         (UsingWifiLocation AND LocationWifi.error==0)) AND not(ChoosingLocationProvider);
50    OnActivate {
51      Property syncInterval=0; }
52  }
```

Figura 6.18: Descrição do Contexto LocationServiceError

Capítulo 7

Ferramenta para Modelagem de UbiACME

Este Capítulo descreve a ferramenta *UbiCME Studio* que permite a descrição de aplicações ubíquas usando a linguagem UbiACME. A ferramenta envolve uma implementação do modelo abstrato *eCore*, como também uma implementação da linguagem, através do parser da ferramenta. Para definição de diagramas gráficos é usado o *sirius*. A seção 7.1 cita os requisitos de *UbiACME Studio*. A seção 7.2 descreve *UbiACME Studio*. A subseção 7.2.1 descreve o metamodelo *eCore* utilizado pela ferramenta *UbiACME Studio*. A subseção 7.2.2 discute a implementação do parser da ferramenta *UbiACME Studio*. A subseção 7.2.3 apresenta Editor *UbiACME Studio* em Execução. A subseção 7.2.4 apresenta o editore gráfico de UbiACME. Por fim, a seção 7.3 apresenta as considerações sobre o Capítulo.

7.1 Requisitos de UbiACME Studio

UbiACME possui elementos diagramáticos e textuais (MACHADO et al., 2014). Contudo, independentemente de ser textual ou diagramático, o suporte ferramental para auxiliar na utilização de determinada linguagem sempre foi visto como um importante diferencial para seus usuários. Nesta seção começaremos citandos seus requisitos:

1. **Permitir a edição de diagramas:** A ferramenta deve permitir ao designer a construção de modelos de maneira intuitiva, muito semelhante aos editores gráficos mais comuns, empregando uma área de edição e uma paleta de elementos da linguagem. Para isso, basta arrastar os componentes do diagrama da paleta para área de edição.

2. **Suportar uma notação processável:** A ferramenta deve possibilitar que modelos UbiACME sejam empregados em diversos processos.

3. **Validar sintaticamente o diagrama:** As vantagens esperadas pelo suporte de um editor gráfico estarão comprometidas se o designer tiver um esforço muito grande para checar visual e mentalmente se seu diagrama está correto. Assim, é imprescindível que a ferramenta garanta ao designer que o diagrama que ele está construindo esteja sintaticamente correto.

7.2 UbiACME Studio

UbiACME Studio é uma ferramenta disponibilizada em forma de plug-in para o *Eclipse* (STEINBERG et al., 2008), para especificações arquiteturais de aplicações ubíquas em UbiACME. A ferramenta foi desenvolvida pelo grupo do laboratório de Consepção de Sistemas (*ConSiste*) da UFRN. *UbiACME Studio* foi construída a partir de um conjunto de *frameworks* para desenvolvimento de ferramentas para auxilio a modelagem. Esses *frameworks* são o *Eclipse Modeling Framework* **(EMF)** (BUDINSKY FRANCK; STEINBERG, 2003), ***Xtext***, um *framework* de código aberto para o desenvolvimento de linguagens de programação e linguagens específicas de domínio (DSL) e ***Sirius***, um *framework* para desenvolvimento de ferramentas gráficas.

7.2.1 UbiACME eCore

Para o desenvolvimento da ferramenta *UbiACME Studio* foi necessária a definição de um metamodelo EMF, conhecido como *eCore*. Este metamodelo é responsável por definir a sintaxe abstrata da linguagem, que será então implementada pelas sintaxes textual e gráfica. O *eCore* de UbiACME foi construído como extensão do metamodelo da ferramenta AspectualACME Studio (BATISTA et al., 2006), desenvolvido por um grupo do laboratório ConSiste. A extensão deu-se através da inclusão dos elementos de UbiACME no modelo já existente de ACME utilizado pela ferramenta *AspectualACME Studio*. Essas inclusões visaram permitir a descrição dos elementos criados pela linguagem UbiACME, tais como: (i) *Context*, (ii) *ContextExpression*, (iii) *OnActivate*, (iv) *OnDeactivate*, (v) *Undo*, (vi) *Persistent*, (vii) *QoSParameter*, (viii) *QoCParamente*, (ix) *Attachment Condicional*, (x) *On - do ...*, (xi) *detach*, e (xii) *remove* (MACHADO et al., 2014).

- ***Context*** é um elemento que representa contexto em UbiACME, o qual possui essencialmente uma expressão de ativação de contexto, que é uma expressão booleana envolvendo propriedades, parâmetros de qualidades e estado de outros contextos (ativo ou inativo) que, quando satisfeita, representa a ativação de um contexto, ou

seja, o sistema encontra-se em um conjunto de circunstâncias pré-definidas. Assume-se que todo e qualquer contexto está desativado quando sua expressão de ativação não é satisfeita. O elemento *Context* possui um *ContextExpression* em sua representação sintática.

- ***ContextExpression*** é responsável por descrever um conjunto de condições através de expressões booleanas que caracterizam as condições que ativam um *Context*, utilizando propriedades, parâmetros de qualidades ou até o estado (ativo/inativo) de outros *Contexts*.

- ***OnActivate*** descreve um conjunto de adaptações arquiteturais que serão realizadas no momento em que o contexto for ativado. Essas adaptações incluem definição de novos elementos, criação ou eliminação de conexões e alterações no comportamento de elementos existentes.

- ***OnDeactivate*** descreve um conjunto de adaptações arquiteturais que serão realizadas no instante em que o *Context* for desativado. O *OnDeactivate* possui um *Undo* em sua definição sintática.

- ***Undo*** serve para desfazer todas as alterações feitas por um **OnActivate** com exceção das adaptações persistentes.

- ***Persistent*** é a alternativa mais simples usada no *OnDeactivate* em situações nas quais o arquiteto não quer que todas as adaptações arquiteturais ser percam ao desativar o *Context*. Pois, durante a definição do elemento *OnActivate*, é possível utilizar a palavra-chave *Persistent* para discriminar as adaptações que irão persistir mesmo após a desativação do *Context*.

- ***QoSParameter*** permite estabelecer quais são os medatados de qualidade de serviços das informações de contexto de um serviço.

- ***QoCParameter*** permite que, para um dado parâmetro de contexto essencial a aplicação, estabeleçam-se os critérios de qualidade relacionados a ele.

- ***Attachment Condicional*** tem comportamento semelhante ao *attachment* nativo de ACME, entretanto só está ativo, ou seja permite o fluxo de informações em situações que são definidas a partir de uma expressão de contexto, a qual consiste em uma combinação de estados (ativo/inativo) de contextos pré-definidos.

- ***On ... Do*** especifica uma condição de adaptação para o sistema e descreve as alterações (adaptações) arquiteturais que serão realizadas quando a condição de adaptação acontecer, permitindo ao arquiteto expressar em qual situação a reconfiguração programada deve ocorre e o que deve ser mudado para efetivar essa reconfiguração.

- ***Detach*** destrói conexões existentes entre componentes do sistema, construtor que permite a alteração da arquitetura, utilizado para remover uma ligação entre uma *port* e uma *role*.

- ***Remove*** usado para destruir um componente, conector, representação, propriedade ou *QoSParamenter* e *QoCParamente* do sistema.

A partir dessas definições no metamodelo para a linguagem ACME, define-se o metamodelo da linguagem UbiACME. A próxima etapa do desenvolvimento é feita automaticamente pelo *framework EMF*, e consiste na geração das classes Java que implementam esse meta-modelo.

7.2.2 Parser UbiACME

Como funcionalidade básica da ferramenta *UbiACME Studio*, fez-se necessário implementar um parser e validadores para a linguagem. O parser é responsável pela análise sintática e semântica da linguagem, e os validadores são um conjunto de ferramentas adicionais que trabalham em conjunto com o parser para complementar as verificações semânticas. Adicionalmente, foi incrementando todo o conjunto de ferramentas de edição textual existente no *UbiACME Studio*, tais como autocompletar e quickfix.

Essa seção será subdividida em três subseções: a sub-seção 7.2.2.1 apresenta a sintaxe UbiACME, em que será descrita a gramática que será reconhecida pela ferramenta e semântica UbiACME; a subseção 4.2.2.2 descreve os escopos definidos para as variáveis; por fim, a subseção 4.2.2.3 apresenta as ferramentas adicionais, que incluem as ferramentas de *quickfix* e auto-completar supracitadas.

7.2.2.1 Sintaxe UbiACME

A partir do metamodelo descrito na seção 7.2.1, descreveu-se a gramática da linguagem UbiACME. Para isso, foi necessário estender o parser já existente na ferramenta *AspectualACME Studio*. A extensão dessa gramática incluiu as formas sintáticas dos novos

elementos inseridos pela linguagem: *Context, ContextExpression, OnActivate, OnDeactivate, Undo, Persistent, QoSParameter, QoCParamente, Attachment Condicional, On ... do , Detach, e Remove.* A gramática da linguagem foi descrita utilizando o framework Xtext, através de instrumentos providos pelo próprio framework para definição de gramáticas, a partir dessa gramática, o Xtext é responsável por gerar automaticamente o parser da ferramenta.

7.2.2.2 Escopo UbiACME

O *framework Xtext* provê um mecanismo simples para a implementação de funções de definição e cálculo de escopo. Esse mecanismo se baseia em um conjunto de classes que estendem a classe *AbstractDeclarativeScopeProvider*, nativo do *framework*. Uma dessas classes é criada por padrão, e deve ser incrementada para atender as necessidades da linguagem que se implementa. Na ferramenta *UbiACME Studio*, essa classe é a *UbiACMEScopeProvider*, e todas as alterações dessa subseção foram feitas sobre essa classe.

7.2.2.3 Ferramentas Adicionais

Para completar as funcionalidades providas pela ferramenta, estendeu-se os mecanismos de *quickfix*, autocompletar e auto-organizar da ferramenta *UbiACME Studio*. Para o *Xtext*, as ferramentas adicionais são implementadas em um outro projeto, o projeto dimap.ufrn.br.ubiacmedsl.ui, que é criado pelo *framework* e possui a implementação das ferramentas adicionais para o editor textual. As extensões descritas nesta subseção foram realizadas nesse projeto.

Quickfix é o nome atribuído ao mecanismo do Eclipse responsável por sugerir ao usuário e realizar correções na descrição textual, quando algum erro é detectado. Foram implementadas funções de *quickfix* para cada um dos erros ou *warnings* disparados pelos validadores para UbiACME. Que são os seguintes:

- ***WarningContextUnactivable:*** *Context* possui *ContextExpression* que nunca assume valor *true*, por isso, nunca será ativado.
- ***ErrorOnActivateInvalidPersistentAttachment:*** *Persistent Attachment* entre dois elementos que não são persistentes.
- ***WarningDetachUndetach:*** *Detach* tentando destruir uma conexão que não existe.
- ***ErrorRemoveUndeletable:*** *Delete* tentando deletar um componente que não existe.

- ***ErrorDetachUndetach:*** *Detach* tentando destruir uma conexão entre um componente removido e um conector.

- ***ErrorDetachUndetach:*** *Detach* tentando destruir uma conexão entre um componente e um conector removido.

Para implementar o *quickfix* foi necessário criar uma função para cada um desses erros/*warnings*. O *Xtext* cria uma classe padrão para a implementação dessas funções, que é a classe *UbiACMEQuickfixProvider*, essa classe, porém, não possui as funções necessárias para tratar os problemas supracitados.

Cada função de *quickfix* deve criar um conjunto de modificações que podem ser realizadas, e cada modificação deve ter um label identificador.

7.2.3 Editor UbiACME em Execução

Como resultado do que foi descrito nas seções anteriores, foi produzido o Editor UbiACME que pode ser visualizado, em execução sob a forma de um *plugin* já completamente integrado ao Eclipse, na figura 5.3, na qual os componentes principais do editor, distribuídos sob a forma de janelas ou painéis do ambiente Eclipse, são descritos a seguir.

A primeira janela, mais à esquerda, é do explorador de pacotes, em que ficam os projetos que empacotam os diagramas. Cada projeto é armazenado no sistema de arquivo sob a forma de uma pasta ou diretório. Já os diagramas são armazenados sob a forma de arquivo, sendo dois para cada diagrama. Um arquivo *(.UbiACME)* armazena o modelo, isto é, as instâncias dos elementos do metamodelo representados através da gramática definida no Xtext, que são traduzidos para implementação no EMF.

Na parte inferior ficam várias janelas, entre elas a de edição de propriedades para elementos selecionados no modelo e a de problemas encontrados no modelo que decorrem do processo de sua validação.

7.2.4 Editor gráfico de UbiACME

A ferramenta Editor Gráfico de UbiACME executa sobre o **Sirius** um *plug-in Eclipse*. O Sirius trabalha com Viewpoints e perspectivas.

O Sirius é um projeto da *Eclipse* que permite que se crie facilmente seu próprio ambiente *(workbench)* de modelagem gráfica, aproveitando as tecnologias de modelagem

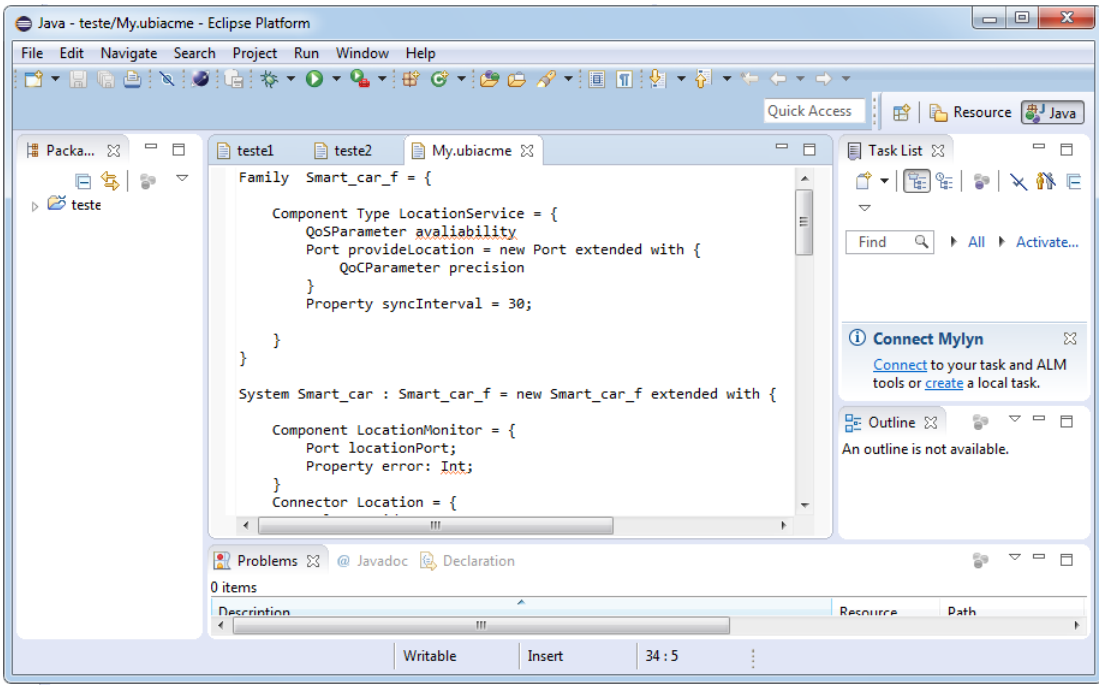

Figura 7.1: Editor UbiACME em Execução Integrado ao Eclipse

do *Eclipse*, incluindo EMF e GMF.

O Sirius foi criado para fornecer um *workbench* genérico para sistemas baseados em modelo de engenharia arquitetural que poderiam ser facilmente adaptados para atender necessidades específicas.

7.2.4.1 Princípios do Sirius

O ambiente *(workbench)* de modelagem criado com Sirius é composto por um conjunto de editores *Eclipse* (diagramas, tabelas e árvores), que permitem aos usuários criar, editar e visualizar modelos EMF. Os editores são definidos por um modelo que define a estrutura completa da *(workbench)* de modelagem, o seu comportamento, a edição e ferramentas de navegação. Essa descrição (workbench) de modelagem Sirius é interpretada dinamicamente em tempo de execução dentro do Eclipse IDE. Para apoiar a necessidade específica de personalização, Sirius é extensível, em muitos aspectos, em especial através do fornecimento de novos tipos de representações, novas linguagens de consulta e por ser capaz de chamar código Java para interagir com Eclipse ou qualquer outro sistema.

7.2.4.2 Editores de diagramas

Sirius oferece suporte para representações diagramáticas, que representam informações de forma gráfica. Diagramas são os tipos principais de representações apoiados por Sirius.

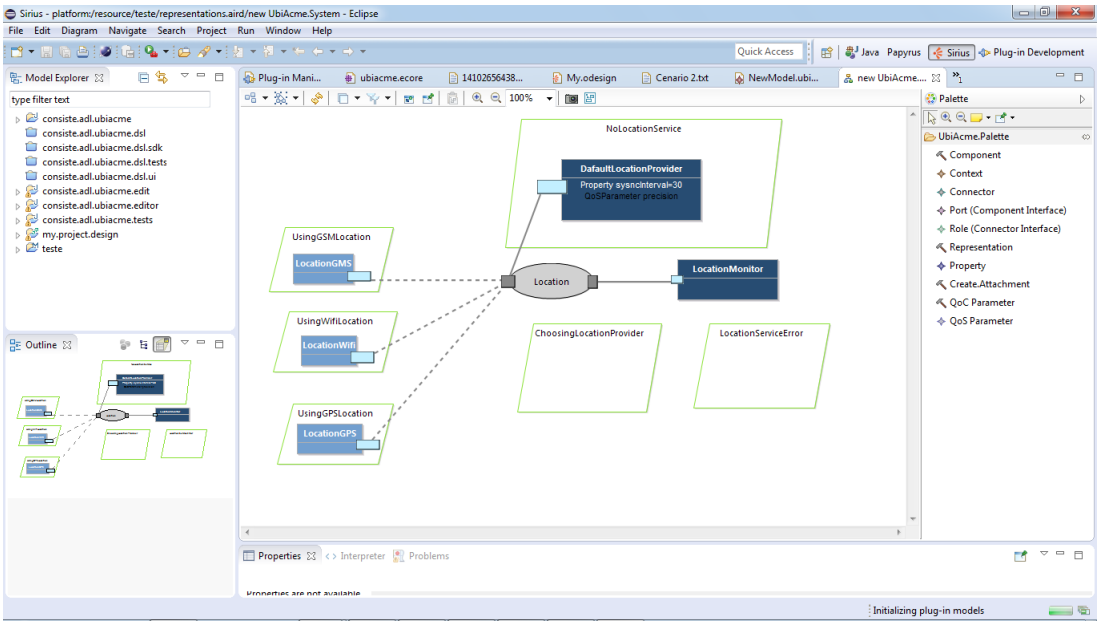

Figura 7.2: Tela Principal da Ferramenta de Edição Gráfica de UbiACME

Um editor de diagrama é dividido em três áreas:

- A área gráfica, que mostra os elementos e suporta interação direta com eles;
- A paleta, que dá acesso a ferramentas adicionais (por exemplo, ferramentas de criação para adicionar novos elementos para o diagrama);
- A barra de páginas na parte superior da área gráfica que fornece operações globais adicionais.

Além dessas áreas, que são imediatamente visíveis no interior do editor de diagramas, também é possível interagir com o esquema e seus elementos através das propriedades da vista e através de menus contextuais disponíveis sobre os elementos gráficos próprios e no fundo do diagrama. Quando o editor atual é um diagrama de Sirius, a vista de destaques Eclipse mostra uma visão geral gráfica do diagrama que pode ser usada para navegar nele se todo o diagrama não é visível na área principal.

7.3 Considerações Finais

Este Capítulo apresentou o Editor *UbiACME Studio*, uma ferramenta desenvolvida para permitir a construção de diagramas representando modelos nessa linguagem. Seu objetivo é auxiliar no processo de comunicação do designer com os demais membros da equipe de desenvolvimento, especialmente os desenvolvedores.

Capítulo 8

Experimento Controlado que Avalia a Linguagem

Este capítulo apresenta os resultados da realização de uma análise qualitativa da Linguagem de Descrição de Arquitetura UbiACME, de acordo com Dimensões Cognitivas de Notações, (CDN do inglês *Cognitive Dimensions of Notations*) (GREEN; PETRE, 1996), que é um *framework* utilizado na avaliação de interação humano-computador (IHC), proposto para auxiliar na análise de usabilidade dos ambientes de programação visual e, de um modo geral, qualquer tipo de sistema notacional.

Visando auxiliar os designers de interface, BLACKWELL e GREEN (2003) destacam que, no geral, no uso do *framework* CDN toda técnica de avaliação busca responder uma questão bem fundamental:

> *"As atividades pretendidas pelos usuários são adequadamente suportadas pelo sistema notacional?"*

Um sistema notacional é constituído de quatro componentes (BLACKWELL; GREEN, 2003):

- Uma **notação**, que engloba os sinais ou símbolos percebidos que são combinados para construir uma estrutura de informação ou um produto da notação;
- Um **ambiente para editar a notação**, que contém as operações e ferramentas para manipulação dos sinais e símbolos;
- Um **meio de interação**, cujos os sinais e símbolos são manipulados, que pode ser persistente como um papel ou transiente como o som;

- E, possivelmente, **subsistemas**, que são partes do sistema principal, que podem ser tratados separadamente porque possuem notação, ambiente e meio próprios, como os visualizadores de esboço, gerenciadores de referências cruzadas, gravadores de macros, gerenciadores de anotações, histórico de ações, etc.

O objetivo desse experimento controlado é avaliar a complexidade, a expressividade, a escalabilidade, a compreensão e o esforço de realização de dado conjunto de alterações em uma aplicação ubíqua modelada em UbiACME, em comparação com a mesma aplicação modelada em *SysML*. A escolha de SysML para ser comparada com UbiACME justifica-se pela ausência de ferramentas para modelagem de aplicações ubíqua. *SysML* é uma linguagem de modelagem de sistemas, de propósito geral com ferramenta visual para aplicações de engenharia de sistemas. *Papyrus* permite realizar modelagem com SysML e também com UML (PAPYRUSUML, 2014).

A linguagem de definição de arquitetura que está sendo validada é UbiACME, faz-se necessário caracterizá-la de acordo com esses componentes.

- O sistema notacional analisado é a Arquitetura de Software;
- A notação é a linguagem UbiACME e também *SysML*;
- O ambiente é o *UbiACME Studio* e também o *Papyrus*;
- O meio é persistente e consiste de arquivos com o código fonte UbiACME /*SysML*;
- Os subsistemas são algumas facilidades oferecidas pelo editor UbiACME /*Papyrus*.

8.1 Atividades Realizadas Usando o Sistema Notacional

Segundo BLACKWELL e GREEN (2003) nenhuma análise pode ser realizada sem que se tenha alguma ideia sobre qual será a utilidade de determinado artefato. Sem pretensão de se realizar uma análise de tarefas altamente detalhada, GREEN e BLACKWELL (1998) sugerem quatro tipos de atividades comumente realizadas sobre os artefatos construídos usando notações. O sistema notacional avaliado pode oferecer suporte diferenciado (bom ou ruim) para essas atividades. A própria análise das dimensões irá demostrar isso de maneira que as dimensões irão impactar, também de maneira diferente sobre cada tipo de atividade. Estes tipos de atividades são descritos a seguir (GREEN; BLACKWELL, 1998):

Atividades	Descrição
Adição	Acrescentar mais informação sem alterar, de nenhuma forma, a estrutura;
Modificação	Alterar uma estrutura existente, sem adicionar novo conteúdo;
Criação	Copiar conteúdo de uma estrutura para outra, comum em processo de exportação e importação;
Exploração	Também chamado de design exploratório combina adição e modificação, atrelado ao fato de não se conhecer antecipadamente o estado final desejado.

Tabela 8.1: Atividades

8.2 Dimensões Cognitivas de Notações

BLACKWELL e GREEN (2003) apresenta as quatorze dimensões cognitivas, demostrando a própria evolução quantitativa e histórica das dimensões (BLACKWELL; GREEN, 2003). Essas dimensões, com as respectivas descrições, estão apresentadas na tabela 8.2:

Dimensões	Descrição
Viscosidade	Resistência às mudanças
Visibilidade	Habilidade de facilmente visualizar os componentes
Compromisso prematuro	Restrições na ordem de fazer alguma coisas
Dependências ocultas	Importantes ligações entre as entidades que não estão visíveis
Expressividade de papéis	O propósito de um componente é facilmente inferido
Propensão a erros	A notação convida a equívocos e o sistema dá pouca proteção
Abstração	Tipos de disponibilidade do mecanismo de abstração
Notação Secundária	Informações extras em outro meio que não a sintaxe formal
Proximidade do Mapeamento	Quão próximo está a representação do domínio
Consistência	Semânticas semelhantes são expressas de maneira sintaticamente similares
Prolixidade	Verbosidade da linguagem
Operação mentais difíceis ou operações complicadas	Alta demanda de recursos cognitivos
Provisoriedade	Grau de comprometimento com ações ou marcações
Avaliação Progressiva	O trabalho como um todo pode ser checado a qualquer momento

Tabela 8.2: Dimensões do Framework CDN

A definição de notação do CDN é bastante genérica, pois considera qualquer sistema com que um ser humano pode interagir, fornecendo e obtendo informações relativas a um artefato sendo manipulado. Além disso, a notação pode incluir um conjunto de sub dispositivos que auxiliam as atividades do usuário, tanto produzindo informações como modificando a própria notação. Em geral, a avaliação do software usando o CDN considera a interface do software com seu usuário como parte da notação sendo avaliada e o resultado da utilização do software como artefato sendo produzido.

O objetivo principal do CDN é definir um vocabulário ou um conjunto básico de questões de facilidades de uso que possam ser utilizados por desenvolvedores de software em geral, sem a necessidade de treinamento prolongado ou conhecimento multidisciplinar (BLACKWELL, 2001). Desde então, o CDN já foi utilizado na avaliação de linguagem de programação (CLARKE, 2001), ambientes de programação virtual (GREEN; PETRE, 1996), APIs (CLARKE, 2006), etc. As facilidades apresentadas do CDN e suas aplicações no domínio de desenvolvimento de software foram decisivas para sua adição neste trabalho como critério de avaliação de UbiACME.

BLACKWELL e GREEN (2003) mostram que uma das principais virtudes do *framework* é clarear as táticas de design, principalmente pela necessidade de se fazer um *trade-off* das dimensões, isso porque elas impactam positiva e/ou negativamente umas sobre as outras. Dessa forma, os autores apontam alguns desses relacionamentos, que são resumidos na Figura 8.1.

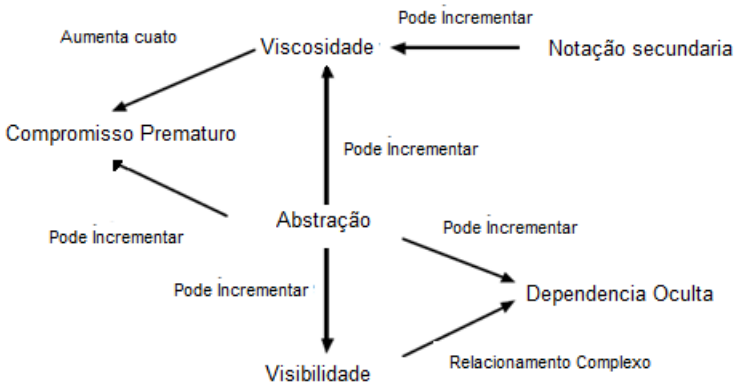

Figura 8.1: Alguns Tipos de *Trade-off*

Como exemplos, eles destacam que uma das formas de se reduzir a **viscosidade** é a introdução de **abstração**, mas isso irá sempre requerer um gerente de abstração, no qual serão definidas as abstrações e algum comprometimento prematuro. Além disso, a própria abstração pode se tornar viscosa, introduzir dependências ocultas, etc.

Além do *trade-off*, em uma análise das dimensões deve-se considerar o impacto de cada dimensão sobre os tipos de atividades que podem ser realizadas sobre o objeto da análise, pois algumas dimensões podem ser **úteis**, **aceitáveis** e até **prejudiciais** para diferentes tipos de atividades. Por exemplo, a viscosidade pode ser **aceitável** para atividades como adição e transcrição, mas **prejudicial** para modificação e design exploratório.

8.3 Análise das Dimensões

O conjunto de dimensões do CDN não é definitivo ou completo, uma vez que não há garantia que as dimensões cubram todas os aspectos relevantes da facilidade de uso. Atualmente 14 dimensões são consideradas estabelecidas e algumas outras, chamadas Dimensões candidatas.

Nesta validação serão analisadas cada uma das dimensões sobre UbiACME, considerando seus respectivos impactos sobre as atividades suportadas. Baseado nos trabalhos

sobre CDN, o impacto ou relevância de cada dimensão será classificado em: Aceitável, prejudicial, benéfico, relevante e irrelevante. Conforme mostra a Tabela 8.3.

8.3.1 Viscosidade

A viscosidade caracteriza-se pela resistência a mudanças. Quanto mais custoso é introduzir uma alteração, mais viscosa é a notação.

GREEN e PETRE (1996) enfatizam que linguagens diagramáticas que adotam o estilo caixa-e-linha *(box-and-line)*, apesar do estilo favorecer essas linguagens, podem facilmente tornar o sistema notacional viscoso, a menos que um bom editor seja fornecido junto com a notação. Como exemplo, os autores citam a viscosidade de *layout*, cujo qual, ao se mover um elemento (caixa) do diagrama, a necessidade de mover as linhas associadas a ele, torna o sistema extremamente viscoso.

Como foi dito por GREEN e PETRE (1996) acima, UbiACME é uma linguagem que adota o estilo caixa-e-linha e pode facilmente tornar o sistema notacional viscoso, mas com a ajuda do *UbiACME Studio,* o editor fornecido junto com anotação, a viscosidade é reduzida.

UbiACME, por ser uma linguagem dinâmica, reduz a viscosidade do software durante a sua execução.

8.3.2 Visibilidade

Essa dimensão refere-se à facilidade de visualizar as diferentes partes do artefato; em particular, de poder comparar visualmente partes distintas do artefato (também chamado de justaposição).

A visibilidade facilita o entendimento do sistema, pois permite analisar novas partes comparando-as com outras já conhecidas. Isso é particularmente importante em modificações exploratórias, cujo o foco da atividade está no aprendizado. A visibilidade também pode facilitar modificações de uma forma geral, pois facilita a reutilização de porções do sistema. Por exemplo, o programador pode visualizar uma parte do sistema sobre o qual ele quer se basear para desenvolver uma nova parte ou realizar uma modificação. Note que essa reutilização não se restringe a código, no qual geralmente é preferível evitar a duplicação de código, mas pode se referir a padrões de codificação, escolha de nomes de variáveis e funções, etc.

Um exemplo de visibilidade pobre é a separação de declaração de funções e suas respectivas implementações em arquivos diferentes, como é comumente feito em programas C e C++. A assinatura das funções geralmente fica em um arquivo de cabeçalho (e.g. .h) e a implementação fica em um arquivo a parte (e.g. .cpp). No caso de linguagem de programação, a visibilidade pode ser amplamente influenciada pelos recursos de editor ou ambiente de desenvolvimento (IDE, do inglês *Integrated Development Environment*) utilizado. Por exemplo, muito editores de texto permitem visualizar diferentes arquivos simultaneamente (justaposição) ou mesmo diferentes porções de um mesmo arquivo.

No editor gráfico UbiACME, existe diferentes janelas para visualização dos artefatos sendo construídos. Além das janelas de edição do diagrama, também existe a paleta que dá acesso a ferramentas adicionais e a barra da páginas que fornece operações globais adicionais na parte superior da área gráfica. Como o editor atual de diagrama de UbiACME executa sobre o *sirius*, um *plug-in Eclipse* e a vista de destaque do *Eclipse* mostra uma visão geral do diagrama, permitindo navegar nele se todo o diagrama não é visível na área principal, conforme mostrado na Figura 7.2.

8.3.3 Compromisso Prematuro

Essa dimensão caracteriza-se por restrições que o usuário deve impor sobre suas futuras necessidades em um momento em que elas não podem ser determinadas apropriadamente. Um exemplo são os *arrays* estáticos de C, cujo tamanho deve ser definido antes da execução da aplicação, porém em muitas situações o tamanho realmente necessário só pode ser definido precisamente quando a aplicação estiver executando.

Independente do custo da correção de compromissos prematuros inadequados (**viscosidade**), a necessidade de defini-los dificulta o uso da notação, pois faz com que o usuário se preocupe em minimizar a possibilidade de definir um compromisso de forma inadequada. Portanto, apesar de um **compromisso prematuro** não implicar necessariamente em um custo mecânico da modificação, eles implicam em um custo mental para o usuário.

As características das linguagens dinâmicas podem efetivamente evitar a necessidade de compromissos prematuros, especialmente aqueles que só podem ser assumidos corretamente com base em informações de tempo de execução.

Devido a natureza dinâmica de aplicações ubíquas, UbiACME incluiu um mecanismo arquitetural para reconfiguração dinâmica programada. Para isso, UbiACME incluiu as extensões propostas por *Plastik: On-DO; Detach; Remove.*

8.3.4 Dependências Ocultas

Essa dimensão caracteriza-se pela quantidade de dependências importantes entre as partes do sistema que não estejam diretamente aparentes para o usuário. Isto é, situações em que um elemento A depende de B e B depende de C, geralmente, são explicitadas. Porém, a ligação estendida A depende de C, não é explicitada, nem as ligações inversas são explicitadas, como C causa dependência em B e B causa dependência em A.

Dependências ocultas impedem um bom entendimento do sistema, como por exemplo identificar a importância ou a necessidade de cada parte individual do sistema. Um exemplo é a revisão de um programa extenso por todas as chamadas de uma função disponível globalmente cujo comportamento deve ser alterado.

Dependência ocultas dificultam identificar e compreender as implicações de modificações, independentemente da dificuldade de introduzi-la **(viscosidade)** ou da sua corretude ou adequação **(propensão a erros)**. Excetuando as modificações que claramente causam pouco impacto na implementação existente, as dependências ocultas geralmente dificultam o entendimento e planejamento da alteração. Em modificações exploratórias, cujo o foco está no aprendizado, as dependências ocultas são particularmente problemáticas.

Por exemplo, em Java, não é fácil identificar as implicações da modificação de uma interface que é herdada por uma variedade de classes ou outras interfaces. Já em UbiACME, o elemento *Type* é usado para definir um vocabulário de tipos abstratos de elementos como componentes, conectores, portas, papéis e contextos que poderão ser utilizados em etapas posteriores. O elemento *Family* é usado para definir estilos arquiteturais em termos de um vocabulário de tipos abstratos de elementos, pois, o instanciamento dos elementos herdados de outros sistemas, caracterizam dependências ocultas.

8.3.5 Expressividade de Papéis

Essa dimensão refere-se à clareza do propósito de cada entidade ou parte do artefato, ou seja, se é possível identificar facilmente a semântica de cada elemento da notação. A expressividade de papéis tem influência direta na facilidade de entendimento e aprendizado do sistema, sendo uma dimensão importante em modificações exploratórias. De forma geral, uma boa expressividade é importante para facilitar a identificação do papel de cada elemento examinado, alterado ou incluído durante as modificações.

GREEN (1989) destaca que um exemplo bem familiar de notação com esta dimensão prejudicada é o fluxograma, em que cada configuração de ações e decisões precisam ser cuidadosamente estudadas para definir se é uma estrutura condicional ou de repetição. Ainda segundo GREEN e PETRE (1996), essa dimensão é presumidamente melhorada pelo uso de identificadores significativos, modularidade bem estruturada e uso de notação secundária.

De forma geral, uma boa expressividade é importante para facilitar a identificação do papel de cada elemento sendo examinado, alterado ou incluído durante as modificações.

Em UbiACME, na sua representação gráfica, as linhas que representam as conexões entre as portas dos componentes e os papéis dos conectores não resta dúvida sobre seu significado no produto sendo criado usando a notação. Já as linhas que indicam a conexão dos *attachment* condicionais das portas dos componentes que fazem parte de um contexto, precisam do uso de identificadores significativos, pois nos diagramas elas são identificadas por serem pontilhadas.

8.3.6 Propensão a Erros

Essa dimensão caracteriza-se pela quantidade de enganos que um usuário é levado a cometer por conta da notação. Em particular, aqueles que não são facilmente detectados ou são detectados inadequadamente, por exemplo, em um momento muito tardio, quando a correção é muito custosa. Um exemplo disso é o uso de aritmética de ponteiros em C, que permite acessos a áreas de memória inválidas que não podem ser detectadas facilmente pelo compilador. Esse é um tipo de erro comum dos programadores.

A propensão a erros é extremamente danosa para a flexibilidade, pois torna as modificações mais difíceis, uma vez que o programador deve ter mais cuidado como planejamento da modificação. Essa dimensão é mais problemática quando combinada com a dificuldade de corrigir os erros cometidos (viscosidade).

Os autores ainda argumentam que notações puramente textuais são mais propícias a ocorrências de equívocos ou a dificultar a localização dos mesmo, quando estes ocorrem. Enquanto que o uso de delimitadores em pares, como chaves e colchetes, convidam os usuários ao deslize, pois mesmo sabendo que precisa fechar um parêntese aberto, não é incomum seu esquecimento (GREEN; PETRE, 1996).

Em UbiACME, na sua representação textual, os blocos com as abstrações arquiteturais são delimitados por "{" e "}". Já as expressões lógicas de ativação e desativação dos

contextos que fazem parte do elementos *ContextExpression* são delimitadas por "[" e "]", caracterizando uma situação de propensão a erros.

8.3.7 Abstração

Uma abstração é uma classe de entidades ou um agrupamento de elementos a ser tratado como uma entidade única, tanto para baixar a viscosidade ou para tornar a notação mais como estrutura conceitual do utilizador.

Esta dimensão está relacionada aos tipos e disponibilidade de mecanismos de abstração.

- ***Abstraction-hating***, que é aquele que não permite o usuário definir novas abstrações;
- ***Abstraction-tolerant***, que é aquele que permite o usuário criar novas abstrações, mas não lhe exige isso para usá-lo;
- ***Abstraction-hungry***, que é aquele que exige que o usuário crie novas abstrações para usá-lo, demandando maior esforço cognitivo para se ter uma compreensão satisfatória dos mecanismos de abstração.

Se o sistema permite que um usuário adicione novas abstrações que necessite, então, serão compreendidos por usuários subsequentes, que irão aumentar ainda mais a barreira de abstração.

8.3.8 Notação Secundária

Essa dimensão refere-se ao suporte à uma notação auxiliar que permita que o usuário possa descrever algo que não é considerado pela notação.

GREEN e PETRE (1996) apontam que em linguagem de programação, exemplos comuns sobre notação secundária são os comentários, endentação e parágrafos "rimados". Essa é uma dimensão importante por permitir que o usuário realize alguma descrição sobre a própria estrutura do produto que pode se tornar tão grande e complexa quanto possível, sem interferir sobre este.

Na representação textual de UbiACME os blocos são endentados para melhorar a visibilidade das abstrações arquiteturais, dado que esta notação auxiliar permite que arquiteto

possa descrever suas abstrações arquiteturais sem que seja considerada pela notação. Na notação gráfica, UbiACME não suporta nenhum tipo de notação secundária.

8.3.9 Proximidade do Mapeamento

Essa dimensão refere-se à adequação de representação do que está sendo modelado ao seu domínio específico. Por exemplo, uma linguagem lógica como *Prolog* apresenta maior proximidade com máquina de inferência baseadas em proposições lógicas do que uma linguagem como *PostScript*, que apresenta maior proximidade com a geração de documentos. Basicamente, linguagens de domínio específico são mais próximas ao seu domínio de aplicação do que linguagens de propósito geral.

Nas descrições das abstrações arquiteturais, UbiACME por ser uma linguagem de um domínio especifico (Computação Ubíqua) apresenta maior proximidade com informações especificas de computação ubíqua, tais como: representação de informações de contexto e a qualidade das informações de contexto relacionadas aos serviços sensíveis ao contexto usados pela aplicação.

8.3.10 Consistência

A consistência refere-se à similaridade sintática de formas com semânticas similares. Um exemplo disso, é a atribuição em Visual Basic, que é feita através da forma var = valor, entretanto, se o valor sendo atribuído for um objeto, é necessário utilizar a forma Set var = valor, caso contrário apenas uma propriedade (denominada propriedade default) é copiada (SCHENTAG, 2009). Um exemplo inverso é o uso da palavra static em C++, que é utilizada com diferentes significados dependendo do contexto (HAMMER, 2009).

Esse tipo de consistência é bastante comum em linguagem diagramática especialmente no estilo caixa-e-linha, como demostrado por GREE e PETRE 1996, as linguagens diagramáticas têm mostrado, quando comparadas com linguagens puramente textuais, um aprimoramento dessa dimensão.

A linguagem UbiACME, por ser uma linguagem diagramática, possui consistência de operações, sobre elementos diagramáticos com mesma semântica, como todos os operadores e interações básicas.

8.3.11 Prolixidade

A prolixidade caracteriza-se pela redundância ou excesso de informações ou representações que podem ofuscar outros itens mais relevantes. Um exemplo disso é o uso de palavras reservadas longas ou que precisem ser repetidas muitas vezes.

Por exemplo, em C++ o escopo de acesso dos membros de uma classe é feito em seções iniciadas por uma palavra reservada, enquanto em Java, cada membro deve ter uma palavra indicando seu escopo, que é repetido muitas vezes.

Declarações explícitas de tipo também podem ser vistas como um exemplo de prolixidade em linguagens e programação, pois em algumas situações elas acrescentam pouca informação ao desenvolvedor, sendo principalmente em recurso que facilita tarefa do compilador.

A declaração explícita de tipo em UbiACME é um exemplo e prolixidade, pois na hora de ser instanciado um determinado componente que foi descrito em um *Family*, sintaticamente passa pouca informação ao arquiteto.

8.3.12 Operação Mentais Difíceis ou Operações Complicadas

Essa dimensão refere-se à exigência que o usuário realize atividades mentais difíceis, ou seja, que demandam muitos recursos cognitivos como lembrança e raciocínio. Exemplo de operações complicadas são operações que manipulam múltiplos níveis de indireção como operações de inserção e remoção de elementos em uma estrutura em árvore implementada usando estruturas com ponteiros em C.

Segundo GREEN e PETRE (1996), esta dimensão é definida por duas propriedades. A primeira diz respeito ao fato de que as operações mentais problemáticas devem estar no nível notacional, não apenas no nível semântico. A segunda é a operação mental difícil propriamente dita, nos termos do *framework*, que é aquela que ao combinar dois ou três objetos não tão complexos, aumentam mais ainda a dificuldade de manipulá-los.

Em UbiACME, uma operação mental difícil ou operação complicada é ativação e desativação de um contexto, pois está a nível notacional, não apenas no nível semântico. Em UbiACME o aninhamento de elementos não tem limites de profundidade e essa combinação pode tornar-se extensa e demandar considerável esforço metal para compreendê-lo.

8.3.13 Provisoriedade

Essa dimensão está relacionada ao grau do comprometimento com as ações ou os signos, isto é, apesar de algum comprometimento inicial é possível realizar algum trabalho de maneira provisória para poder usá-lo mais tarde com algum grau de alteração, inclusive, dos comprometimentos já feitos, representando alguma possibilidade de voltar atrás.

De uma forma geral, a provisoriedade melhora a flexibilidade pois permite realizar mais facilmente modificações que não são completamente compreendidas ou planejadas antes da sua realização. Isso é especialmente importante em modificações experimentais, pois comumente envolvem um aprendizado durante o desenvolvimento que pode ser necessário para tornar mais claras as decisões adequadas que devem ser tomadas.

A tipagem dinâmica, assim como a inferência de tipos na tipagem estática, funcionam como mecanismos de provisão, pois permitem que algumas porções do programa se ajustem automaticamente às alterações nos tipos dos valores feitas durante o desenvolvimento.

A reflexão computacional pode ser vista como um mecanismo de provisão, pois o programador pode utilizar os mecanismos de intercepção oferecidos para alterar alguns comportamentos e com isso contornar algumas decisões.

Em UbiACME podemos considerar uma provisão os elementos *Type* usados para definir um vocabulário de tipos abstratos de elementos como componentes, conectores, portas e papéis, que poderão ser instanciados para serem usados mais tarde. Já o elemento *Family* é usado para definir estilos arquiteturais em termos de um vocabulário de tipo abstratos de elementos.

8.3.14 Avaliação Progressiva

Essa dimensão se caracteriza pela possibilidade de verificar ou avaliar o produto ainda parcialmente confeccionado. No caso de programas, isso se reflete na possibilidade de compilá-lo ou executá-lo antes de estar completamente terminado ou correto. Um exemplo seria poder avaliar o funcionamento do mecanismo de realização de chamadas remotas de um *Object Request Broker – ORB*, usando como parâmetros, valores cujos tipos já têm uma codificação para formato de transmissão implementada, mesmo que nem todos os tipos da linguagem já tenham sua codificação implementada.

UbiACME por ser um linguagem de definição de arquitetura podemos ter um avaliação

progressiva de uma arquitetura em uma fase inicial de definição sem ter de entrar em detalhes finais ou de implementação da definição.

8.4 Relação entre as Dimensões e as Atividades

Dependendo do tipo de atividades sendo avaliada com o *framework* CDN, as diferentes dimensões podem ter relevâncias diferentes. A Figura 8.3 que indica as relevâncias da algumas dimensões utilizadas neste trabalho para cada tipo de atividade realizada.

Cada dimensão é vista como positiva ou negativa para a flexibilidade do software tanto quanto são consideradas modificações definitivas ou exploratórias, entretanto para cada tipo de modificação a intensidade dessa relevância pode ser mais ou menos acentuada.

Dimensão	**Criação**	**Adição**	**Modificação**	**Exploração**
Viscosidade	Benefíco	Benefíco	Prejudicial	Prejudicial
Visibilidade	Irrelevante	Irrelevante	Aceitável	Benefíco
Compromisso prematuro	Prejudicial	Prejudicial	Irrelevante	Irrelevante
Dependências ocultas	Benefíco	Benefíco	Irrelevante	Irrelevante
Expressividade de Papéis	Benefíco	Benefíco	Aceitável	Aceitável
Propensão a Erros	Benefíco	Benefíco	Benefíco	Aceitável
Abstração	Irrelevante	Irrelevante	Irrelevante	Irrelevante
Notação secundária	Aceitável	Aceitável	Aceitável	Aceitável
Proximidade do Mapeamento	Benefíco	Aceitável	Aceitável	Benefíco
Consistência	Benefíco	Aceitável	Aceitável	Benefíco
Prolixidade	Prejudicial	Prejudicial	Irrelevantel	Irrelevante
Operações Complicadas	Prejudicial	Prejudicial	Irrelevantel	Irrelevante
Provisoriedade	Benefíco	Aceitável	Aceitável	Benefíco
Avaliação Progressiva	Benefíco	Aceitável	Aceitável	Benefíco

Tabela 8.3: Dimensões x Atividades

8.5 Projeto do Experimento Controlado

O experimento foi aplicado a quatro alunos de pós-graduação da Universidade Federal do Rio Grande do Norte, do Departamento de Informática e Matemática Aplicada, com graduação em Ciência da Computação e áreas afins. Esse grupo de quatro alunos foi dividido em dois grupos, nomeados de Grupo I e Grupo II. O documento completo do projeto do experimento pode ser encontrado no anexo A e no endereço:

`http://consiste.dimap.ufrn.br/projects/ubiacme/docs/experimento.pdf`.

O experimento foi projetado com o objetivo de avaliar:

1. A complexidade;
2. A expressividade;

3. A escalabilidade;

4. A compreensão;

5. O esforço de realização de dado conjunto de alterações.

Essas características foram avaliadas no contexto de uma aplicação ubíqua modelada em UbiACME e também em *SysML*.

A escolha de *SysML* para ser comparada com UbiACME justifica-se devido a falta de ferramentas adequadas disponibilizadas por ADLs para computação Ubíqua. *SysML* é uma linguagem de modelagem de propósito geral baseada em UML e com ferramenta disponível *(Papyrus)* para modelagem arquitetural.

8.5.1 Questões de Pesquisa

As questões que permeiam este experimento controlado são:

Q1 - Qual o grau de complexidade para descrever uma aplicação ubíqua usando UbiACME em comparação a *SysML*?

Q2 - Qual o grau de expressividade em relação a características de computação ubíqua para descrever suficientemente sistemas de Computação Ubíqua usando UbiACME em comparação a *SysML*?

Q3 - Qual o grau de escalabilidade para descrever sistemas de Computação Ubíqua usando UbiACME em comparação a *SysML*?

Q4 - Qual o grau de esforço para realizar um dado conjunto de alterações em sistemas de Computação Ubíqua usando UbiACME em comparação a *SysML*?

Q5 - Qual o grau de compreensão com relação à descrição arquitetural em sistemas em Computação Ubíqua usando UbiACME em comparação a *SysML*?

8.5.2 Execução do Experimento

O experimento controlado foi realizado em quatro etapas. Na **primeira etapa** foi realizado um treinamento envolvendo os elementos envolvidos (a saber, UbiACME, *SysML* e a aplicação ubíqua) com o intuito de nivelar o conhecimento dos participantes. Antes de iniciar a segunda etapa, que consistiu na descrição da aplicação ubíqua usando UbiACME

e *SysML*, o ambiente de execução foi preparado, etapa na qual foram instalados os *plug-ins* do *Eclipse* IDE referente às ferramentas utilizadas para as descrições arquiteturais.

Para dar início à **segunda etapa** do experimento, os participantes responderam a um questionário inicial que caracteriza o perfil de cada participante e tem por objetivo coletar informações acerca da experiência profissional deles, habilidades em termos de desenvolvimento de software e familiaridade com os conceitos relacionados a ADLs e Computação Ubíqua. De maneira adicional, parte do modelo da aplicação ubíqua foi fornecida aos membros do Grupo I, bem como dois produtos (de tamanhos diferentes) resultantes da combinação de algumas partes da aplicação. Feito isso, os 2 integrantes do Grupo I realizaram a descrição arquitetural da aplicação ubíqua em UbiACME e os 2 integrantes do Grupo II realizaram a descrição arquitetural da aplicação ubíqua em *SysML*.

Na **terceira etapa** do experimento, após os integrantes do Grupo I realizarem as descrições arquiteturais, eles receberam uma lista com um dado conjunto de alterações que deveriam ser realizadas sobre cada descrição da aplicação em computação Ubíqua e dos respectivos produtos.

Na **quarta e última etapa** do experimento, as descrições arquiteturais feitas pelos integrantes do Grupo I foram analisadas pelos 2 integrantes do Grupo II, que tentaram, a partir de tais descrições, verificar a arquitetura baseando-se nos documentos de especificação, a fim de verificar o grau de compreensão das descrições.

Após a aplicação do experimento, os participantes responderam aos questionários disponíveis na página:

`http://consiste.dimap.ufrn.br/projects/ubiacme/docs/Questionnaires.zip`.

A partir da análise das respostas a esses questionários e dos dados coletados durante o experimento foram extraídos os dados que levam as conclusões que serão apresentadas na próxima seção Os dados completos relacionados ao experimento, tais como os artefatos fornecidos aos participantes podem ser encontrados na página da ferramenta *UbiACME Studio*, no endereço:

`http://consiste.dimap.ufrn.br/projects/ubiacme/experiments`

8.6 Relatório do experimento

A análise dos resultados do experimento baseou-se na análise dos questionários respondidos pelos participantes, do tempo despendido para a descrição realizada pelos participantes e das descrições criadas pelos participantes para UbiACME e *SysML*, baseado nos artefatos fornecidos. Os resultados da análise dos questionários e tempos despendidos foram os seguintes:

Q1 - Qual o grau de **complexidade** para descrever uma aplicação ubíqua usando **UbiACME** em comparação a ***SysML***?

A complexidade no uso de uma determinada linguagem ou ferramenta mede a facilidade em realizar uma tarefa ou um determinado conjunto de tarefas, de modo que, quão maior a facilidade, menor a complexidade. Pois, usando esses critérios e dimensões do CDN, foi avaliado o grau de **complexidade** para descrever uma aplicação de computação ubíqua usando **UbiACME** e ***SysML***. Foi concluído que **UbiACME** tem grau de complexidade menor ou igual a ***SysML*** para este tipo de aplicação, sabendo-se que todas as dimensões cognitivas são utilizadas para avaliar a complexidade entre duas linguagens. **UbiACME** apresentou-se melhor em 9 das 14 dimensões utilizadas na comparação, tais como: **Operações Complicadas, Prolixidade, Dependência Ocultas, Comprometimento Prematuro, Visibilidade, Provisoriedade, Expressividade de Papéis, Proximidade do Mapeamento e Consistência**, sendo que ***SysML*** apresentou melhores resultados para 4 das dimensões: **Propensão a Erros, Viscosidade, Abstração e Notação Secundária**. As linguagens apresentaram o mesmo desempenho em apenas uma dimensão: **Avaliação Progressiva**.

Q2 - Qual o grau de **expressividade** em relação a características de computação ubíqua para descrever suficientemente sistemas de Computação Ubíqua usando **UbiACME** em comparação a ***SysML***?

A **expressividade** está relacionada as seguintes dimensões: **Abstração e Expressividade de Papéis**. Ambas as dimensões contribuem positivamente para a expressividade. Feita as comparações entre as linguagens **UbiACME** e ***SysML*** usando os critérios e as dimensões do CDN, foi concluído que **UbiACME** tem maior grau de expressividade que ***SysML***, pois: **UbiACME** apresentou melhores resultados em 2 das 2 dimensões comparadas.

Q3 - Qual o grau de **escalabilidade** para descrever sistemas de Computação Ubíqua usando **UbiACME** em comparação a ***SysML***?

A **escalabilidade** é afetada negativamente pela **Prolixidade, Propensão a erros e Operações Complicadas**. São características desejáveis uma boa **Notação secundária**, um alto grau de **Avaliação Progressiva** e uma boa **Visibilidade**. Feita as comparações entre as linguagens **UbiACME** e **SysML** usando os critérios e as dimensões do CDN, foi concluído que **UbiACME** tem maior grau de **escalabilidade** que ***SysML***, pois: **UbiACME** apresentou resultados melhores em 3 das 6 dimensões comparadas: **Prolixidade, Operações Complicadas e Visibilidade**, sendo que **SysML** apresentou resultados melhores em 2 dimensões: **Propensão a Erros e Notação Secundária**. Em apenas uma das 6 dimensões comparadas as duas linguagens apresentaram resultados equivalentes: **Avaliação Progressiva**.

Q4 - Qual o grau de **esforço para realizar** um dado conjunto de alterações em sistemas de Computação Ubíqua usando **UbiACME** em comparação a ***SysML***?

O esforço para realizar um dado conjunto de alterações, quando em graus elevados, afeta negativamente o sistema que utiliza a notação, para realização de mudanças em produtos já existentes pelas seguintes dimensões: **Viscosidade, Propensão a erros, Operações Complicadas, Dependências Ocultas e Comprometimento Prematuro**. Já as dimensões **Avaliação Progressiva e Visibilidade** afetam positivamente em quaisquer graus. **UbiACME** apresentou melhores resultados em 4 das 7 dimensões comparadas: **Operações Complicadas, Dependências Ocultas, Comprometimento Prematuro e Visibilidade**. ***SysML*** apresentou resultados melhores em apenas 2 das 7 dimensões comparadas: **Viscosidade e Propensão a Erros**. Ambas as linguagens apresentaram resultados equivalentes em uma dimensão: **Avaliação Progressiva**.

Q5 - Qual o grau de **compreensão** com relação à descrição arquitetural em sistemas em Computação Ubíqua usando **UbiACME** em comparação a ***SysML***?

As dimensões **Visibilidade, Expressividade de papeis, Proximidade do Mapeamento e Consistência** afetam positivamente a compreensão da linguagem, por auxiliarem na compreensão de produtos existentes. **Dependências Ocultas**, por outro lado, afetam negativamente, prejudicando a compreensão. Feitas as comparações entre as linguagens **UbiACME** e ***SysML*** usando os critérios e as dimensões do CDN para avaliar o grau de compreensão com relação à descrição arquitetural em sistemas em computação Ubíqua, observou-se que **UbiACME** apresentou resultados melhores em 5 das 5 dimensões comparadas, portanto possuindo um grau de compreensão maior que ***SysML***.

Como resultado adicional não previsto, os participantes apontaram a necessidade de melhores instrumentos de notação secundária para **UbiACME**. ***SysML*** implementa um

sistema de notas e comentários que poderiam ser adotados por **UbiACME**, que apenas apresenta instrumentos semelhantes a nível textual. Os participantes também relataram que a ferramenta de **UbiACME** impõe resistência a mudanças em descrição gráfica (Viscosidade), de modo que essa precisa ser melhorada para permitir uma melhor manipulação dos modelos existentes.

Devido a grande diferença semântica entre os elementos das linguagens **UbiACME** e ***SysML***, os resultados e comentários dos participantes sugerem que alguns participantes foram induzidos ao erro quando realizaram, em um primeiro instante, descrições em ***SysML***, por tentar replicar as ações tomadas nessa linguagem que já lhes era familiar. É possível que esse fato tenha prejudicado a avaliação da dimensão **Propensão a Erros**, o que não afeta o resultado do experimento.

8.7 Ameaças a Validade

Em termos de ameaças a validade do experimento realizado, podemos citar:

1. Uma ameaça importante foi o número reduzido de participantes que participou do experimento controlado. No entanto, a engenharia de software experimental não estabelece um número mínimo de participantes para uma avaliação, o único problema é que com poucos participantes não é possível generalizar os resultados, mas conseguimos ter evidências sobre o sistema avaliado.

2. Uma outra ameaça foi que as perguntas foram diretas e com o objetivo específico de avaliar cada uma das dimensões, limitando a liberdade de respostas dos participantes. Esse aspecto foi minimizado pelo fato do questionário ter um campo livre, no final, para os participantes pudessem incluir sugestões.

3. Uma outra ameaça foi a escolha da linguagem de modelagem para comparar as especificações do sistema ubíquo, pois a literatura não disponibiliza uma ADL específica para Computação ubíqua com uma ferramenta associada. Por esse motivo, escolhemos a linguagem SysML, uma linguagem de propósito geral que permite a modelagem de quaisquer tipos de sistemas, mas não oferece abstrações específicas para representação de sistemas ubíquos. Esse fato pode ter tornado a avaliação tendenciosa para UbiACME, por essa linguagem ser específica para computação ubíqua.

8.8 Considerações Finais

Este Capítulo apresentou o experimento controlado que avaliou a linguagem UbiACME usando o CDN (Dimensões Cognitivas de Notações), as atividades usadas e a definição das 14 dimensões, com discussão sobre a linguagem UbiACME. Também foi apresentado uma breve descrição do experimento controlado e, finalmente, foi apresentado o relatório do experimento mostrando suas conclusões sobre a comparação das linguagens UbiACME e *SysML*.

Capítulo 9

Considerações Finais e Trabalhos Futuros

Esta tese apresentou UbiACME, uma linguagem de descrição arquitetural para aplicações ubíquas, que permite arquitetos de software realizarem a modelagem arquitetural de aplicações ubíquas. UbiACME inclui suporte a informações específicas de computação ubíqua, como: representação das informações de contexto, qualidade das informações de contextos relacionadas aos serviços sensíveis ao contexto usados pela aplicação e reconfiguração dinâmica. Para exemplificar UbiACME, foi apresentada uma aplicação ubíqua fazendo uso de dois cenários: (i) Piloto automático em autoestradas e (ii) Serviço de localização. Em ambos cenários foram apresentadas as representações gráfica e textual em UbiACME fazendo uso de *UbiACME Studio*, uma ferramenta de modelagem gráfica desenvolvida no contexto deste trabalho para permitir a descrição de aplicações ubíquas. *UbiACME Studio* foi disponibilizado em forma de *plug-in* para o *Eclipse*. Para avaliar UbiACME foi realizado um experimento controlado com os alunos de pós-graduação do Departamento de informática e Matemática aplicada da UFRN. O experimento foi projetado com o objetivo de avaliar a complexidade, a expressividade, a escalabilidade, a compreensão e o esforço de realização de dado conjunto de alterações em um sistema ubíquo modelado usando UbiACME, em comparação com o mesmo sistema modelado usando *SysML*.

Como base para a definição de UbiACME foi realizada uma revisão sistemática de forma a investigar na literatura relacionada com sistemas ubíquos, os elementos que constituem arquitetura de tais sistemas e também se havia arquiteturas de referências para tal domínio. Nessa revisão sistemática foram identificados elementos essenciais para sistemas ubíquos e não foi encontrado nenhum trabalho que apresentasse uma arquitetura de referência para computação ubíqua.

Com base nos elementos encontrados na revisão sistemática, foi definida uma arquitetura de referência para sistemas ubíquos, RA-Ubi, que agrega conhecimento de domínio de computação ubíqua, identifica soluções para problemas nesse domínio e promove a reutilização do conhecimento para o desenvolvimento de aplicações para esse domínio. RA-Ubi foi estabelecida com base no ProSA-RA, um processo que visa a construção, representação e avaliação de arquiteturas de referência. RA-Ubi foi avaliada com base no método definido em Angelove (2008), que propõe uma avaliação de arquitetura referência por meio de um método de propósito geral e verificada através de um *checklist*. Ou seja, foi feita uma comparação dos elementos de RA-Ubi com os elementos das arquiteturas das aplicações de sistemas ubíquos, encontrados na revisão bibliográfica. Dessa forma, avaliou-se se a arquitetura de referência provê todos os elementos essenciais, e se existem elementos fundamentais que não foram incluídos nessa arquitetura. Logo, foi concluído que todos os elementos das arquiteturas das aplicações comparadas com os elementos de RA-Ubi faziam parte dos elementos de RA-Ubi.

Portanto, podemos afirmar que são válidas todas as hipóteses que foram levantadas no capítulo 1: H1: A revisão sistemática contribuiu para identificar as características de aplicação ubíqua; H2: A arquitetura de referência expressa os elementos que caracterizam sistemas ubíquos e a relação entre eles; H3: Uma linguagem de descrição arquitetural, baseada na arquitetura de referência é útil para representar arquiteturas de aplicações ubíquas.

Nesta tese foi proposta uma ADL para representação de aplicações ubíquas com uma ferramenta associada, UbiACME Studio. Consideramos que essa é uma contribuição relevante e inovadora, uma vez que a literatura não reporta nenhuma ADL com características específicas para representação de aplicações ubíquas.

Este capítulo esta estruturado da seguinte forma: a seção 9.1 apresenta as principais contribuições desta tese; a seção 9.2 apresenta as limitações desse trabalho e, finalmente, a seção 9.3 apresenta os trabalhos futuros, a seção 9.4 apresenta os resultados da Tese.

9.1 Contribuições

Esta tese traz contribuições para as áreas de arquitetura de software e computação ubíqua, uma vez que propõe uma ADL para representação de arquiteturas de aplicações ubíquas.

De forma a permitir a modelagem arquitetural de aplicações ubíquas, esta tese teve

como sua principal contribuição a especificação de **UbiACME, uma linguagem de descrição arquitetural para modelagem de aplicações ubíquas**. Além disso, uma outra contribuição foi a disponibilização de **UbiAcme Studio**, uma ferramenta que permite arquitetos de software realizarem modelagens usando UbiACME, tanto de forma textual como de forma gráfica.

As principais contribuições desta tese estão relacionadas a proposição da linguagem UbiACME:

- Uma revisão sistemática que identificou os elementos comuns a sistemas ubíquos, usados como base no projeto de UbiACME.
- Uma arquitetura de referência para sistemas ubíquos, RA-Ubi, baseada na revisão sistemática, que forneceu subsídios para a definição dos elementos de UbiACME.
- Uma linguagem para modelar aplicações ubíquas, UbiACME, que fornece abstrações específicas para sistemas ubíquos.
- Uma ferramenta, *UbiACME Studio*, que permite a modelagem textual e gráfica de aplicações ubíquas em UbiACME.

9.2 Limitações

Há várias limitações neste trabalho associadas com as contribuições:

- Na revisão sistemática: **a limitação está relacionada** à completude desse estudo, ou seja, se de fato todos os artigos da literatura relacionado ao tema foram incluídos.
- Na arquitetura de referência para computação ubíqua: a literatura tem limitações em termos de métodos dedicados à avaliação de arquiteturas de referência (ANGELOV et al., 2008). Dessa forma, avaliamos a RA-Ubi fazendo comparações entre os seus elementos com os elementos da arquiteturas de sistemas ubíquos existentes.
- Na linguagem de descrição de arquitetura, UbiACME: como a literatura não disponibiliza ADLs para computação ubíqua, para avaliar a linguagem, usamos a linguagem *SysML*, que é uma linguagem de propósito geral e de modelagem visual para aplicações de engenharia de sistemas. Como *SysML* não é especifica para computação ubíqua, a avaliação comparativa pode ser considerada tendenciosa para UbiACME.

- Na Ferramenta *UbiACME studio*: na ferramenta há limitação na integração do *Sírius* com o *Xtext*. Com isso, atualmente não é feita a conversão automática de definições textuais da arquitetura em representações gráficas.

9.3 Trabalhos Futuros

Em relação a trabalhos futuros que podem ser derivados desta tese, podemos citar:

1. Realização de outros experimentos usando outras aplicações ubíquas para ampliar a avaliação da linguagem.
2. Especificação formal de UbiACME de forma a definir a semântica formal da linguagem.
3. Extensão da ferramenta *UbiACME Studio*, para permitir que comunicação entre o *Sírius* e o *Xtext*, de forma a conversão de representações textuais de UbiACME em representações gráficas.
4. Validação mais ampla de RA-Ubi, através de experimentos com especialistas da área para verificação da completude da arquitetura.
5. Integrar UbiACME com uma linguagem de programação de forma a permitir a geração de esqueletos de código a partir da descrição arquitetural em UbiACME. O ambiente *UbiACME Studio* deve incluir essa funcionalidade.

9.4 Resultados da Tese

Em termos de resultados adicionais, podemos citar as publicações, todas em conferências internacionais, e a ferramenta UbiACME Studio, disponível para uso.

Publicações:

1. MACHADO, C. A. N. ; SILVA, E. A. F. ; BATISTA, T; LEITE, J. C.; Elisa Yumi Nakagawa . Architectural Elements of Ubiquitous Systems: A Systematic Review. In: The Eighth International Conference on Software Engineering Advances (ICSEA), 2013, Veneza. Proceedings of the Eighth International Conference on Software Engineering Advances. Wilmington, DE, USA: IARIA, 2013. p. 208-213.

2. MACHADO, C. A. N. ; SILVA, E. A. F.; Batista, Thais Vasconcelos; LEITE, J. C.; DELICATO, Flavia C. ;PIRES, P. F. Uma Linguagem de Descrição Arquitetural para Sistemas Ubíquos. In: XVII Ibero-American Conference on Software Engineering (CiBSE), 2014, Pucón - Chile. Proceedings of the XVII Ibero-American Conference on Software Engineering, 2014.

3. MACHADO, C. A. N. ; SILVA, E. A. F.; Batista, Thais Vasconcelos; LEITE, J. C.; Elisa Yumi Nakagawa . RA-Ubi: a Reference Architecture for Ubiquitous Computing. In: European Conference on Software Architecture (ECSA), 2014, Viena - Austria. Proceedings of the 8th European Conference on Software Architecture (ECSA), 2014.

Ferramenta produzida: *UbiACME Studio*. Disponível para download em:

`http://consiste.dimap.ufrn.br/projects/ubiacme`

Referências

HAMMER,. [S.l.: s.n.].

ALLEN, R. *A Formal Approach to Software Architecture*. Tese (Doutorado) — Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

ANGELOV et al. An e-contracting reference architecture. *Journal of Systems and Software*, Elsevier, v. 81, n. 11, p. 1816–1844, 2008.

ANGELOV, S.; GREFEN, P.; GREEFHORST, D. A classification of software reference architectures: Analyzing their success and effectiveness. In: IEEE. *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*. [S.l.], 2009. p. 141–150.

ARSANJANI, A. et al. S3: A service-oriented reference architecture. *IT professional*, IEEE, v. 9, n. 3, p. 10–17, 2007.

AUTOSAR. *AUTomotive Open System ARchitecture*. 2012. Acessado em 15/08/2012, Online, http: //www.autosar.org/.

BARDRAM, J. E. The java context awareness framework (jcaf)–a service infrastructure and programming framework for context-aware applications. In: *Pervasive Computing*. [S.l.]: Springer, 2005. p. 98–115.

BASHROUSH, R. et al. Ali: An extensible architecture description language for industrial applications. In: IEEE. *Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the*. [S.l.], 2008. p. 297–304.

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software architecture in practice. addison-wesley. p. 523, 2003.

BATISTA, C. et al. Monitoramento de metadados para computação ubíqua. In: *XXXIX Seminário Integrado de Software e Hardware (SEMISH 2012), Curitiba, PR, Brasil. Anais do XXXII Congresso da Sociedade Brasileira de Computação (CSBC 2012). Porto Alegre, RS, Brasil: SBC*. [S.l.: s.n.], 2012.

BATISTA, T. et al. Aspectual connectors: Supporting the seamless integration of aspects and adls. 2006.

BATISTA, T.; JOOLIA, A.; COULSON, G. Managing dynamic reconfiguration in component-based systems. In: *EWSA 2005 2 nd European Workshop on Software Architectures*. [S.l.]: Springer, 2005. p. 1–17.

BATORY, D. et al. Creating reference architectures: an example from avionics. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 1995. v. 20, n. SI, p. 27–37.

BLACKAND B.; KNAPP, C. Reference architecture for mobile robotics. Technical report,National Instruments, 2010.

BLACKWELL, A.; GREEN, T. R. G. Notational systems ? the cognitive dimensions of notations framework. *CARROLL, John M. (Ed.). HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science. San Francisco*, v. 7, p. 103+, 2003.

BLACKWELL, A. e. a. Cognitive dimensions of notations: Design tools for congnitive technology. *BEYNON, M.; NEHANIV, C. L.; DAUTENHAHN, K. (Ed.). Anais do Cognitive Technology 2001. Berlin, Alemanha: Springer*, v. 2117, p. 325–341, 2001.

BOER, R. C. de; FARENHORST, R. In search of 'architectural knowledge'. In: *Proceedings of the 3rd International Workshop on Sharing and Reusing Architectural Knowledge*. New York, NY, USA: ACM, 2008. (SHARK '08), p. 71–78. ISBN 978-1-60558-038-8. Disponível em: <http://doi.acm.org/10.1145/1370062.1370080>.

BUDINSKY FRANCK; STEINBERG, D. E. R. *Eclipse Modeling Framework: A Developer's Guide*. 2003.

CHANGYUN, L.; GANSHENG, L. he pinjie, formal dynamic architecture description language d-adl. *Journal of Sotware*, v. 17, n. 6, p. 1349–1359, 2006.

CHEN, H. L. *An intelligent broker architecture for pervasive context-aware systems*. Tese (Doutorado) — University of Maryland, Baltimore County, 2004.

CLARKE, S. Evaluating a new programming language. *In: KADORA, G. F. (Ed.). 13th Annual Workshop of the Psychology of Programming Interest Group. Keele, Reino Unido: Universidade de Keele*, v. 1, p. 275–289, 2001.

CLARKE, S. Describing and measuring api usability with the cognitive dimensions. *In: Cognitive Dimensions of Notations 10th Anniversary Workshop*, p. 131?174, 2006.

CLEMENTS, P. C. A survey of architecture description languages. In: IEEE COMPUTER SOCIETY. *Proceedings of the 8th international workshop on software specification and design*. [S.l.], 1996. p. 16.

DAVIS, A. et al. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. p. 176–185, 2006.

DEY, A. K. Understanding and using context. *Personal Ubiquitous Comput.*, Springer-Verlag, London, UK, UK, v. 5, n. 1, p. 4–7, jan. 2001. ISSN 1617-4909. Disponível em: <http://dx.doi.org/10.1007/s007790170019>.

DYBA, T.; KITCHENHAM, B. A.; JORGENSEN, M. Evidence-based software engineering for practitioners. *Software, IEEE*, IEEE, v. 22, n. 1, p. 58–65, 2005.

EKLUND, U. et al. Experience of introducing reference architectures in the development of automotive electronic systems. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 2005. v. 30, n. 4, p. 1–6.

ENDO, A. T.; SIMAO, A. S. A systematic review on formal testing approaches for web services. Natal, BR, Brazil, p. 89–98, 2010.

ENGSTROM E., R. P.; SKOGLUND, M. A systematic review on regression test selection techniques. p. 14–30, 2010.

EUGSTER, P. T.; GARBINATO, B.; HOLZER, A. Middleware support for context-aware applications. In: *Middleware for Network Eccentric and Mobile Applications*. [S.l.]: Springer, 2009. p. 305–322.

FEILER, P. H.; LEWIS, B.; VESTAL, S. The sae avionics architecture description language (aadl) standard: A basis for model-based architecture-driven embedded systems engineering. In: *RTAS 2003 Workshop on Model-Driven Embedded Systems*. [S.l.: s.n.], 2003.

FERNANDEZ-MONTES, A. et al. Smart environment software reference architecture. In: *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*. Washington, DC, USA: IEEE Computer Society, 2009. (NCM '09), p. 397–403. ISBN 978-0-7695-3769-6. Disponível em: <http://dx.doi.org/10.1109/NCM.2009.115>.

GALSTER, M.; AVGERIOU, P. Empirically-grounded reference architectures: A proposal. In: *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*. New York, NY, USA: ACM, 2011. (QoSA-ISARCS '11), p. 153–158. ISBN 978-1-4503-0724-6. Disponível em: <http://doi.acm.org/10.1145/2000259.2000285>.

GARLAN, D.; ALLEN, R.; OCKERBLOOM, J. Exploiting style in architectural design environments. In: *Proceedings of SIGSOFT'94: The Second ACM SIGSOFT Symposium on the Foundations of Software Engineering*. [S.l.]: ACM Press, 1994.

GARLAN, D.; MONROE, R.; WILE, D. Acme: An architecture description interchange language. In: *in Proceedings of CASCON?97*. [S.l.: s.n.], 1997. p. 169–183.

GARLAN, D.; MONROE, R. T.; WILE, D. Acme: Architectural description of component-based systems. *Foundations of component-based systems*, v. 68, p. 47–68, 2000.

GRAAF, B.; DIJK, H. van; DEURSEN, A. van. Evaluating an embedded software reference architecture - industrial experience report. Manchester, UK, UK, p. 354–363, 2005.

GREEN, T. R.; PETRE, M. Usability analysis of visual programming environments: a "cognitive dimensions" framework. *Journal Of Visual Languages And Computing*, v. 7, p. 131–174, 1996.

GREEN, T. R. G.; BLACKWELL, A. Cognitive dimensions of information artefacts:a tutorial. 1998.

GU, T. et al. A middleware for building context-aware mobile services. In: *In Proceedings of IEEE Vehicular Technology Conference (VTC*. [S.l.: s.n.], 2004.

HAMMER. static: The multipurpose keyword. 2009. Disponível em: <http://www.cprogramming.com/tutorial/statickeyword.html>.

HAN, S. W. et al. A new middleware architecture for ubiquitous computing environment. In: IEEE. *Software Technologies for Future Embedded and Ubiquitous Systems, 2004. Proceedings. Second IEEE Workshop on*. [S.l.], 2004. p. 117–121.

HOFER, T. et al. Context-awareness on mobile devices - the hydrogen approach. In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2002. p. 292–302.

HORN, P. Autonomic computing. IBM's Perspective on the State of Information Technology, 2001.

KARVONEN, H.; KUJALA, T. In-car ubiquitous computing: Driver tutoring messages presented on a head-up display intelligent transportation systems conference. In: ITSC '06. IEEE. [S.l.], 2006. p. 560–565.

KAZMAN, R. et al. Saam: a method for analyzing the properties of software architectures. In: *Proceedings of the 16th international conference on Software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994. (ICSE '94), p. 81–90. ISBN 0-8186-5855-X. Disponível em: <http://dl.acm.org/citation.cfm?id=257734.257746>.

KAZMAN, R. et al. The architecture tradeoff analysis method. In: IEEE. *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*. [S.l.], 1998. p. 68–78.

KITCHENHAM, B. *Procedures for performing systematic reviews*. [S.l.], 2004.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering. In: *Technical Report EBSE 2007-001*. [S.l.]: Keele University and Durham University Joint Report, 2007.

KITCHENHAM, B. A.; DYBA, T.; JORGENSEN, M. Evidence-based software engineering. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (ICSE '04), p. 273–281. ISBN 0-7695-2163-0. Disponível em: <http://dl.acm.org/citation.cfm?id=998675.999432>.

KORPIPAA, P. et al. Managing context information in mobile devices. *Pervasive Computing, IEEE*, IEEE, v. 2, n. 3, p. 42–51, 2003.

KUMAR, S. Challenges for ubiquitous computing. In: IEEE. *Networking and Services, 2009. ICNS'09. Fifth International Conference on*. [S.l.], 2009. p. 526–535.

LIU, Y.; LI, F. Pca: A reference architecture for pervasive computing. In: IEEE. *Pervasive Computing and Applications, 2006 1st International Symposium on*. [S.l.], 2006. p. 99–103.

LOPES, A.; FIADEIRO, J. L. Context-awareness in software architectures. In: MORRISON, R.; OQUENDO, F. (Ed.). *EWSA*. [S.l.]: Springer, 2005. (Lecture Notes in Computer Science, v. 3527), p. 146–161. ISBN 3-540-26275-X.

LOUREIRO A.A.FERREIRA, O. R. A. R. S. T. R. d. M. B. 27º simpósio brasileiro de redes de computadores e sistemas distribuídos. In: . [S.l.: s.n.], 2009.

LUCKHAM, D. C. Rapide: A language and toolset for simulation of distributed systems by partial orderings of events. In: *Princeton University*. [S.l.: s.n.], 1996.

LYYTINEN, K.; YOO, Y. Ubiquitous computing. *Communicationsof the ACM*, v. 45, n. 12, p. 63, 2002.

MACHADO, C. A. M. et al. Architectural elements of ubiquitous systems. a systematic review. Proceedings of The Eighth International Conference on Software Engineering Advances (ICSEA2013), 2013.

MACHADO, C. A. M. et al. Uma linguagem de descrição arquitetural para sistemas ubíquos. XVII Ibero-American Conference on Software Engineering (CiBSE), Pucón , Chile, 2014.

MACHADO, C. A. N. et al. Ra-ubi: a reference architecture for ubiquitous computing. In: European Conference on Software Architecture (ECSA), 2014, Viena - Austria. Proceedings of the 8th European Conference on Software Architecture (ECSA), 2014.

MEDVIDOVIC, N.; TAYLOR, R. N. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 26, n. 1, p. 70–93, jan. 2000. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.825767>.

MEDVIDOVIC, N.; TAYLOR, R. N. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on*, IEEE, v. 26, n. 1, p. 70–93, 2000.

MONROE, R. T. *Capturing Software Architecture Design Expertise with Armani Version 1.0*. [S.l.], 1998.

MULLER, G. A reference architecture primer. *Eindhoven Univ. of Techn., Eindhoven, White paper*, 2008.

NAKAGAWA, E. Y.; ANTONINO, P. O.; BECKER, M. Reference architecture and product line architecture: A subtle but critical difference. In: *Software Architecture*. [S.l.]: Springer, 2011. p. 207–211.

NAKAGAWA, E. Y. et al. Towards a process to design aspect-oriented reference architectures. In: *Proceedings of the XXXV Latin American Informatics Conference (CLEI'2009)(Pelotas, Brazil)*. [S.l.: s.n.], 2009.

NAKAGAWA, E. Y.; OQUENDO, F.; BECKER, M. Ramodel: A reference model for reference architectures. In: IEEE. *Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012 Joint Working IEEE/IFIP Conference on*. [S.l.], 2012. p. 297–301.

NAKAGAWA E. Y.; MALDONADO, J. C. Reference architecture knowledge representation: An experience. In: *In: III Internacional Workshop on Sharing and Reusing Architectural Knowlegde (SHARK 2008)*. Leipzig, Alemanha: [s.n.], 2008. p. 51–54.

NETO, C. de S. S.; RODRIGUES, R. F.; SOARES, L. F. G. Architectural description of qos provisioning for multimedia application support. In: CHEN, Y.-P. P. (Ed.). *10th International Multimedia Modeling Conference (MMM 2004), 5-7 January 2004, Brisbane, Australia*. [S.l.]: IEEE Computer Society, 2004. p. 161–166. ISBN 0-7695-2084-7.

OLIVEIRA, L. B. R.; NAKAGAWA, E. Y. A service-oriented reference architecture for software testing tools. In: *Software Architecture*. [S.l.]: Springer, 2011. p. 405–421.

OLIVEIRA, L. B. R. de et al. Reference models and reference architectures based on service-oriented architecture: a systematic review. In: *Software Architecture*. [S.l.]: Springer, 2010. p. 360–367.

OQUENDO, F. π-adl: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 29, n. 3, p. 1–14, 2004.

PAPYRUSUML, O. Papyrus for sysml. 2014. Disponível em: <http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp>.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *In Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering, EASE?08*. Swinton, UK, UK: British Computer Society, 2008. p. 68?77.

RATIONAL, S. C. Rational unified process: Best practices for software development teams. 2001. Disponível em: <Disponível em: <http://www-128.ibm.com-/developerworks/rational/library/content/03July/1000/1251/1251 bestpractices TP026B.pdf>.>.

RAYCHOUDHURY, V. et al. Middleware for pervasive computing: A survey. *Pervasive Mob. Comput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 9, n. 2, p. 177–200, abr. 2013. ISSN 1574-1192. Disponível em: <http://dx.doi.org/10.1016/j.pmcj.2012.08.006>.

ROMÁN, M. et al. A middleware infrastructure for active spaces. *Pervasive Computing, IEEE*, IEEE, v. 1, n. 4, p. 74–83, 2002.

SAEED, A.; WAHEED, T. An extensive survey of context-aware middleware architectures. In: IEEE. *Electro/Information Technology (EIT), 2010 IEEE International Conference on*. [S.l.], 2010. p. 1–6.

SCHENTAG, A. Using the set command. 2009. Disponível em: <http://www.programmer-corner.com/viewTutorial.php/1>.

SHAW, M.; GARLAN, D. Software architecture: perspectives on an emerging discipline. Prentice Hall, 1996.

SONG, C.-W. et al. Interactive middleware architecture for lifelog based context awareness. *Multimedia Tools and Applications*, Springer, p. 1–14, 2013.

SPÍNOLA, R. O.; TRAVASSOS, G. H. Towards a framework to characterize ubiquitous software projects. *Information and Software Technology*, Elsevier, v. 54, n. 7, p. 759–785, 2012.

STEINBERG, D. et al. *Eclipse Modeling Framework. 2nd. ed. [S.l.]: Addison-Wesley Professional*. 2008.

SUGUMARAN, V. et al. Adaptive middleware architecture for information sharing on mobile phones. In: ACM. *Proceedings of the 2007 ACM symposium on Applied computing*. [S.l.], 2007. p. 800–804.

WANT, R. et al. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, ACM, v. 10, n. 1, p. 91–102, 1992.

WEISER, M. The computer for the 21st century. *Scientific american*, Nature Publishing Group, v. 265, n. 3, p. 94–104, 1991.

WILLIAMS, B.; CARVER, J. Characterizing software architecture changes: A systematic review. 2010.

WU, Q.; LI, Y. Scudadl: An architecture description language for adaptive middleware in ubiquitous computing environments. In: IEEE. *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*. [S.l.], 2009. v. 4, p. 611–614.

XU, T. et al. A context-aware middleware for ambient intelligence. In: ACM. *Proceedings of the Workshop on Posters and Demos Track*. [S.l.], 2011. p. 10.

ZHOU, J. et al. Psc-rm: Reference model for pervasive service composition. *2010 Fifth International Conference on Frontier of Computer Science and Technology*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 705–709, 2009.

APÊNDICE A – BNF de UbiACME

BNF Meta-Syntax

Keyword	Keywords are specified with bold text. Keywords are case insensitive
Non-Terminal	Non-Terminals are specified with italics
(...)	Parentheses group tokens and productions
[...]	Indicates an optional production
(...)?	Indicates a sequence of zero or one elements (synonymous with [])
(...)+	Sequence of one or more elements
(...)*	Sequence of zero or more elements
|	Seperates alternative choices

UbiACME Grammar

UbiACMEDesign ::=	(TypeDeclaration | FamilyDeclaration | DesignAnalysisDeclaration)* [SystemDeclaration] <EOF>

Design Element Types:

FamilyDeclaration ::=	**Family** Identifier ["(" ")"] "=" FamilyBody [";"]
FamilyBody ::=	"{" (TypeDeclaration)* "}"
TypeDeclaration ::=	ElementTypeDeclaration | PropertyTypeDeclaration
ElementTypeDeclaration ::=	ComponentTypeDeclaration | ConnectorTypeDeclaration | PortTypeDeclaration | RoleTypeDeclaration | QoCParameterTypeDeclaration | QoSParameterTypeDeclaration

ComponentTypeDeclaration ::= **Component Type** Identifier "="
parse_ComponentDescription [";"]
|
Component Type Identifier **Extends**
Identifier ("," Identifier)*
With parse_ComponentDescription [";"]

ConnectorTypeDeclaration ::= **Connector Type** Identifier "="
parse_ConnectorDescription [";"]
|
Connector Type Identifier **Extendss**
Identifier ("," Identifier)*
With parse_ConnectorDescription [";"]

PortTypeDeclaration ::= **Port Type** Identifier "="
parse_PortDescription [";"]
|
Port Type Identifier Extends Identifier
("," Identifier)* **With** parse_PortDescription [";"]

RoleTypeDeclaration ::= **Role Type** Identifier "=" parse_RoleDescription [";"
|
Role Type Identifier **Extends** Identifier
("," Identifier)* **With** parse_RoleDescription [";"]

QoCParameterTypeDeclaration ::= **QoCParameter Type** [Dynamic] Identifier "="
parse_QoCParameterDescription [";"]
| **QoCParameter Type** [Dynamic] Identifier
Extends Identifier ("," Identifier)*
With parse_QoCParameterDescription [";"]

QoSParameterTypeDeclaration ::= **QoSParameter Type** [Dynamic] Identifier "="
parse_QoSParameterDescription [";"]
| **QoSParameter Type** [Dynamic] Identifier
Extends Identifier ("," Identifier)*
With parse_QoSParameterDescription [";"]

lookup_ComponentTypeByName ::= Identifier
lookup_ConnectorTypeByName ::= Identifier
lookup_PortTypeByName ::= Identifier
lookup_RoleTypeByName ::= Identifier
lookup_PropertyTypeByName ::= Identifier
lookup_QoCParameterTypeByName ::= Identifier
lookup_QoSParameterTypeByName ::= Identifier

Design Elements:

SystemDeclaration ::= **System** Identifier (":" Identifier)? "=" systemBody [";"]

SystemBody ::=
(**New** lookup_ComponentTypeByName
| "{" (ComponentDeclaration | ComponentsBlock
| ConnectorDeclaration | ConnectorsBlock
| PortDeclaration | PortsBlock
| RoleDeclaration | RolesBlock
| PropertyDeclaration | PropertiesBlock
| AttachmentsDeclaration
| RepresentationDeclaration
| DesignRule
| ContextDeclaration
| PlastikExpression
)* "}")
[**Extended With** SystemBody]

ContextDeclaration ::=
Context Identifier "=" "{"
ActivationExpression "=" BooleanExpression ";"
(PropertyDeclaration | PropertiesBlock
| OnActivateDeclaration | OnDeactivateDeclaration
)*
"}"

OnActivateDeclaration ::=
OnActivate "{"
PlastikExpression*
"}"

OnDeactivateDeclaration::=
OnDeactivate "{"
PlastikExpression* | **Undo**;
"}"

PlastikExpression ::=
PlastikOnDo
| PlastikAction;

PlastikOnDo ::=
on BooleanExpression **do** ("{"
PlastikAction* "}"
| PlastikAction)

PlastikAction ::=
(PlastikDetach
| PlastikActive
| PlastikRemove
| ComponentDeclaration | ComponentsBlock
)

PlastikDetach ::= **Detach** Identifier from Identifier ";"
PlastikActive ::= **Active** Identifier ";"
PlastikRemove ::= **Remove** Identifier ";"

ComponentDeclaration ::=
Component Identifier
[":" lookup_ComponentTypeByName]
("=" parse_ComponentDescription ";"
| ";")

ComponentsBlock ::=
Components "{"
(Identifier
[":" lookup_ComponentTypeByName]
("=" parse_ComponentDescription ";"
| ";")
)*
"}" [";"]

parse_ComponentDescription ::=
(**New** lookup_ComponentTypeByName
|
"{" (PortDeclaration | PortsBlock
| PropertyDeclaration
| PropertiesBlock
| RepresentationDeclaration
| QoCParameterDeclaration
| QoSParameterDeclaration
| DesignRule
| PlastikExpression)*
"}"
)
[**Extended With** parse_ComponentDescription]

ConnectorDeclaration ::=
Connector Identifier
[":" lookup_ConnectorTypeByName]
("=" parse_ConnectorDescription ";"
| ";")

ConnectorsBlock ::=
Connectors "{"
(Identifier
[":" lookup_ConnectorTypeByName]
("=" parse_ConnectorDescription ";" | ";"))*
"}" [";"]

parse_ComponentDescription ::=
(**New** lookup_ComponentTypeByName
| "{" (PortDeclaration | PortsBlock
| PropertyDeclaration
| PropertiesBlock
| RepresentationDeclaration
| QoCParameterDeclaration
| QoSParameterDeclaration
| DesignRule
| PlastikExpression)*
"}"
)
[**Extended With** parse_ComponentDescription]

PortDeclaration ::=
Port Identifier
[“:” lookup_PortTypeByName]
(“=” parse_PortDescription “;” | “;”)

PortsBlock ::=
Ports “{”
(Identifier
[“:” lookup_PortTypeByName]
(“=” parse_PortDescription “;” | “;”))*
“}” [“;”]

parse_PortDescription ::=
(**New** lookup_PortTypeByName
| “{” (PropertyDeclaration | PropertiesBlock
| RepresentationDeclaration | DesignRule
| QoCParameterDeclaration
| QoSParameterDeclaration)*
“}”
)
[**Extended With** parse_PortDescription]

RoleDeclaration ::=
Role Identifier
[“:” lookup_RoleTypeByName]
(“=” parse_RoleDescription “;” | “;”)

RolesBlock ::=
Roles “{”
(Identifier
[“:” lookup_RoleTypeByName]
(“=” parse_RoleDescription “;” | “;”))*
“}” [";”]

parse_RoleDescription ::=
(**New** lookup_RoleTypeByName
| “{” (PropertyDeclaration | PropertiesBlock |
RepresentationDeclaration | DesignRule
| QoCParameterDeclaration
| QoSParameterDeclaration)*
“}”)
[**Extended with** parse_RoleDescription]

QoCParameterDeclaration ::=
QoCParameter Identifier
[“:” lookup_QoCParameterTypeByName]
(“=” parse_QoCParameterDescription “;” | “;”)

QoCParametersBlock ::=
QoCParameters “{”
(Identifier
[“:” lookup_QoCParameterTypeByName]
(“=” parse_QoCParameterDescription “;” | “;”))*
“}” [“;”]

parse_QoCParameterDescription ::= (**New** lookup_QoCParameterTypeByName
| “{” (PropertyDeclaration | PropertiesBlock
| RepresentationDeclaration | DesignRule)*
“}”
)
[**Extended With** parse_QoCParameterDescription

QoSParameterDeclaration ::= **QoSParameter** Identifier
[“:” lookup_QoSParameterTypeByName]
(“=” parse_QoSParameterDescription “;” | “;”)

QoSParametersBlock ::= **QoSParameters** “{”
(Identifier
[“:” lookup_QoSParameterTypeByName]
(“=” parse_QoSParameterDescription “;” | ";"))*
“}” [";"]

parse_QoSParameterDescription ::= (**New** lookup_QoSParameterTypeByName
| “{” (PropertyDeclaration | PropertiesBlock
| RepresentationDeclaration | DesignRule)*
“}”
)
[**Extended With** parse_QoSParameterDescription]

AttachmentDeclaration ::= [Identifier “=”]
Attachment
(Identifier “.” Identifier to Identifier “.” Identifier [“[”
ContextActivationOrExpression “]”]
[“{” (PropertyDeclaration | PropertiesBlock)* “}”]
“;”

AttachmentsDeclaration ::= [Identifier “=”]
Attachments “{”
(Identifier “.” Identifier to Identifier “.” Identifier [“[”
ContextActivationOrExpression “]”]
[“{” (PropertyDeclaration | PropertiesBlock)* “}”]
“;”)*
“}” “;”

ContextActivationOrExpression ::= ContextActivationAndExpression ["OR"
ContextActivationAndExpression];

ContextActivationAndExpression ::= Identifier
| "("ContextActivationOrExpression ")"
| ContextActivationAndExpression "AND"
ContextActivationAndExpression

Properties:

PropertyDeclaration ::= **Property** parse_PropertyDescription ";"

PropertiesBlock ::= **Properties** "{"
[parse_PropertyDescription
(";"parse_PropertyDescription | ";")*
]
"}"[";"]

parse_PropertyDescription ::= [**Property**] Identifier
":" PropertyTypeDescription
["=" PropertyValueDeclaration]
["«" parse_PropertyDescription
(";" parse_PropertyDescription |";")*
"»"
|
"«" "»"
]

PropertyTypeDeclaration ::= **Property Type** Identifier
(";"
| "=" (**Int** ";" |**Long** ";" |**Double** ";" | **Float** ";"
| **String** ";" | **Boolean** ";" | **Any** ";"
| **Enum** ["{" Identifier ("," Identifier)* "}"] ";"
| **Set** ["{" "}"] ";"
| **Set** "{" PropertyTypeDescription "}" ";"
| **Sequence** ["<" ">"] ";"
| **Sequence** "<" PropertyTypeDescription ">" ";"
| **Record** "[" parse_RecordFieldDescription
(";" parse_RecordFieldDescription | ";")* "]" ";"
| **Record** ["[" "]"]";"
|Identifier ";"
)
)

PropertyTypeDescription ::= **Int** | **Long** | **Float** | **Double** | **String**
| **Boolean** | **Any**
| **Set** ["{" [PropertyTypeDescription] "}"]
| **Sequence** ["<" [PropertyTypeDescription] ">"]
| **Record** "[" parse_RecordFieldDescription
(";" parse_RecordFieldDescription | ";")* "]"
| **Record** ["[" "]"]
| **Enum** ["{" Identifier ("," Identifier)* "}"]
| **Enum** ["{" "}"]
| Identifier

parse_RecordFieldDescription ::= Identifier ("," Identifier)*
[":" PropertyTypeDescription]

PropertyValueDeclaration ::= Integer_Literal | Floating_Point_Literal | String_Literal | **False** | **True** | AcmeSetValue | AcmeSequenceValue | AcmeRecordValue | Identifier

AcmeSetValue ::= "{" "}"
| "{" PropertyValueDeclaration
("," PropertyValueDeclaration)* "}"
AcmeSequenceValue ::= "<" ">"|
"<" PropertyValueDeclaration
("," PropertyValueDeclaration)* ">"

AcmeRecordValue ::= "[" RecordFieldValue (";" RecordFieldValue | ";")* "]"

RecordFieldValue ::= Identifier ":" PropertyTypeDescription "="
PropertyValueDeclaration

Representations and Bindings:

RepresentationDeclaration ::= **Representation** "{"
SystemDeclaration
[BindingsMapDeclaration]
"}" [";"]

BindingsMapDeclaration ::= **Bindings** "=" "{" (BindingDeclaration)* "}" [";"]

BindingDeclaration ::= [Identifier "."] Identifier to
[Identifier "."] Identifier
["{" (PropertyDeclaration | PropertiesBlock)* "}"] ";"

Design Rules and Analyses:

DesignRule ::= (**Design**)? (**Invariant** | **Heuristic**)
DesignRuleExpression ";"

DesignRuleExpression ::= QuantifiedExpression | BooleanExpression

QuantifiedExpression ::= (**forall** | **exists**) Identifier ":"
lookup_arbitraryTypeByName **in**
SetExpression "|" DesignRuleExpression

BooleanExpression ::= OrExpression (**and** OrExpression)*

OrExpression ::= ImpliesExpression (**or** ImpliesExpression)*

ImpliesExpression ::= IffExpression ("->" IffExpression)*

IffExpression ::= EqualityExpression ("<->" EqualityExpression)*

EqualityExpression ::= RelationalExpression (**"=="** RelationalExpression
| "!=" RelationalExpression)*

RelationalExpression ::= AdditiveExpression
(“<” AdditiveExpression | “>” AdditiveExpression
| “<=” AdditiveExpression | “=>” AdditiveExpression)*

AdditiveExpression ::= MultiplicativeExpression
(“+” MultiplicativeExpression
| “-” MultiplicativeExpression)*

MultiplicativeExpression ::= UnaryExpression
(“*” UnaryExpression
| “/” UnaryExpression
| “%” UnaryExpression)*

UnaryExpression ::= “!” UnaryExpression
| “-” UnaryExpression
| PrimitiveExpression

PrimitiveExpression ::= “(” DesignRuleExpression “)”
| LiteralConstant | DesignAnalysisCall | Id
Id ::= Identifier (“.” Identifier)*

DesignAnalysisCall ::= Id “("ActualParams “)"

LiteralConstant ::= IntegerLiteral | FloatingPointLiteral | StringLiteral
| **true** | **false**

ActualParams ::= (ActualParam (“,” ActualParam)*)?
FormalParams ::= (FormalParam (“,” FormalParam)*)?
ActualParam ::= LiteralConstant | DesignAnalysisCall | Id
FormalParam ::= Identifier (“,” Identifier)* “:”
(Identifier | **Component** | **Connector** | **Port** | **Role**
|**Int** | **Float** | **String** | **Boolean**)

SetExpression ::= (SetReference | SetFunction | LiteralSet
| SetConstructor)

SetReference ::= Identifier ((“.” Identifier) | (“.” Components)
| (“.” Connectors) | (“.” Ports) | (“."Roles)
| (“.” Representations) | (“.” Properties))+

SetFunction ::= (**Union** | **Intersection** | **Setdiff**)
“(” SetExpression “,” SetExpression “)”

LiteralSet ::= (“{” “}” |
“{” (LiteralConstant | Id) (“,” (LiteralConstant | Id))*
“}”)

SetConstructor ::= “{” **Select** Identifier “:” lookup_arbitraryTypeByName in
SetExpression “|” DesignRuleExpression “}”

APÊNDICE B – Questionarios do Experimento

Sistema: "Sistema de descrição arquitetural"

Produto: Arquiteturas de Software

Notação: UbiAcme | SysML

Dispositivos de ajuda e redefinição: *UbiACME Studio | Papyrus*

Tema: Rastreamento de pacientes e médicos

Mecanismos: UbiACME, SysML

Atividades:

- Criar trecho descrição -> Documento de especificação

- Identificar elementos em descrição -> Descrição completa

- Alterar descrição -> Descrição completa e especificação

- Remover trecho da descrição -> Descrição completa e especificação

Documentos necessários:

- Especificação do sistema

- Descrição exemplo

- Especificação de alteração

- Especificação de remoção

Seção 1: Informações do sistema.

Qual é o nome do sistema?	
Quanto tempo você tem de usá-lo?	
Você se considera proficiente em seu uso?	
Você já usou outros sistemas similares? (Se sim, por favor, nomeá-los)	

Seção 2: Definições.

Talvez seja preciso pensar com cuidado para responder às perguntas nas próximas seções, então nós fornecemos algumas definições e um exemplo para você começar:

Produto
O **produto** é a razão pela qual você está usando o sistema de notação - que as coisas acontecem como resultado final, ou o que as coisas vão ser produzidos como resultado do uso do sistema de notação. Este evento ou objeto é chamado o produto. Qualquer produto que precisa de uma notação para descrever, geralmente, tem uma estrutura complexa.

Notação

A **notação** é a forma como você se comunica com o sistema - você fornecer informações em algum formato especial para descrever o resultado final que você quer, e a notação fornece informações que você pode ler. Notações têm uma estrutura que corresponde, de alguma forma com a estrutura do produto que descrevem. Eles também têm partes (componentes, aspectos, etc), que correspondem de alguma forma a partes do produto.

Notações pode incluir texto, imagens, diagramas, tabelas, símbolos especiais ou várias combinações destes. Alguns sistemas incluem várias notações. Estes podem ser muito semelhantes entre si - por exemplo, quando se usa uma máquina de escrever, o texto que ele produz é apenas letras e caracteres, enquanto a notação sobre as teclas que você pressiona lhe diz exatamente como obter o resultado desejado. Em outros casos, um sistema pode incluir algumas notações que são difíceis para o ser humano para produzir ou a ler. Por exemplo, quando você usa um telefone a notação nos botões é um simples arranjo de dígitos, mas os ruídos que você ouve não são tão fáceis de interpretar (diferentes tons de discagem para cada número, cliques e tons de toque). Um telefone com uma tela, portanto, fornece mais uma notação que é mais fácil para o usuário humano de entender.

Sub-dispositivos

Os sistemas complexos podem incluir várias notações especializadas para ajudar com uma parte específica do trabalho. Alguns destes não pode normalmente ser considerado como parte do sistema, por exemplo, quando você ficar um Post-it na tela do computador para lembrá-lo o que escrever em um documento de processador de texto.

Existem dois tipos destes **sub-dispositivos**.

- O Post-It nota é um exemplo de um **dispositivo auxiliar**. Outro exemplo é quando você fazer anotações de números de telefone no verso de um envelope: o sistema completo é o telefone mais as notas de papel - se você não tem algum tipo de dispositivo auxiliar como o envelope, o telefone seria muito menos útil.
- Uma **redefinição do dispositivo** muda a principal notação de alguma forma - como definir um atalho de teclado, um código de discagem rápida de um telefone ou uma função macro. O dispositivo redefinição permite definir esses atalhos, redefini-los, excluí-los e assim por diante.
-

Note-se que ambos os dispositivos auxiliares e redefinição dos dispositivos necessitam de suas próprias notações que são separadas da notação principal do sistema. Portanto, pedimos que você lhe considere separadamente no resto deste questionário.

Para rever a forma como pretendemos usar estes termos, considere o exemplo de cartas de negócio em um processador de texto. O produto de usar o processador de texto é a letra impressa em papel. A notação é a maneira que a letra aparece na tela - em processadores de texto modernos parece muito semelhante ao que será impresso, mas isso não foi sempre o caso. Se você deseja localizar e substituir uma determinada palavra em todo o documento, você pode chamar-se um dispositivo auxiliar, a busca e substituição de função, geralmente com sua própria janela. Esta janela tem a sua própria notação especial - a maneira que você tem que escrever o texto a ser encontrado e substituído, bem como botões que você pode clicar para encontrar palavras inteiras, ou para encontrar a palavra em letras maiúsculas e minúsculas, etc

Seção 3: Parte do Sistema

Que tarefa ou atividade que você usa no sistema?	
Qual é o produto utilizado no sistema?	
Qual é a principal notação do Sistema?	

Ao usar o sistema, qual a proporção do tempo que você gasta (em porcentagem)?	
Tempo de pesquisa de informações dentro da notação.	%
Tempo de traduzindo quantidades substanciais de informação de alguma outra fonte para o sistema.	%
Tempo para adicionar pequenos pedaços de informações para uma descrição que você criou anteriormente.	%
Tempo para reorganizar e reestruturar as descrições que você criou anteriormente.	%
Tempo de brincadeira com novas ideias na notação, sem ter certeza de qual será o resultado.	%

Seção 4: Questões sobre as notações principais:

[Visibilidade]

Quão fácil é ver ou encontrar as várias partes da notação enquanto durante a criação ou a alteração? Por quê? Que tipo de coisas são mais difíceis de ver ou encontrar? Se você precisa comparar ou combinar diferentes peças, você pode vê-los ao mesmo tempo? Se não, por que não?	

[Viscosidade]

Quando você precisar fazer alterações em trabalhos anteriores, quão fácil é fazer mudanças? Por quê? Há mudanças particulares que são mais difíceis ou especialmente difíceis de fazer? Quais?	

[Prolixidade]

A notação permite a) ser razoavelmente brevemente, ou b) exige uma descrição detalhada? Por quê? Que tipo de coisas requerem mais tempo para descrever?	

[Operações mentais]

Que tipo de coisas requerem o esforço mais mental com esta notação? Alguma coisa em específico parece especialmente complexo ou difícil de exercitar na sua cabeça (por exemplo, combinar várias coisas)? Quais são elas?	

[Propensão a erros]

Existe alguns tipos de erro que parece particularmente comum ou fácil de cometer? Quais? Você cometeu vários pequenos deslizes que irritaram você ou fizeram você se sentir estúpido? Quais são alguns exemplos?	

[Proximidade do mapeamento]

Como intimamente relacionado é a notação para o resultado que você está descrevendo? Por quê? Quais as partes parecem particularmente estranhas de fazer?	

[Expressividade de papeis]

Ao ler a notação, é fácil dizer o que cada parte é no esquema geral? Por quê? Há algumas partes que são particularmente difíceis de interpretar? Quais? Existem peças que você realmente não sabe o que eles querem dizer, mas você lhe coloca apenas porque dessa forma funciona? Quais são elas?	

[Dependencias Ocultas]

Se a estrutura do produto significa que algumas partes estão intimamente relacionados com outras partes, e mudanças afetam outras partes, são dependências são visíveis? Que tipo de dependências estão escondidos? De que forma pode ficar pior quando você está criando uma descrição particularmente grande?	

[Avaliação Progressiva]

Quão fácil é parar no meio da criação de alguma notação, e verificar o seu trabalho até agora? Você pode fazer isso a qualquer momento? Se não, por que não? Você pode descobrir o quanto progresso você fez, ou verificar em que fase em seu trabalho que você está fazendo? Se não, por que não? Você pode experimentar versões parcialmente concluídos do produto? Se não, por que não?	

[Provisoriedade]

É possível esboçar coisas quando você está brincando com ideias, ou quando você não tem certeza de qual caminho a seguir? Quais são as características da notação que ajudaram a fazer isso? Que tipo de coisas que você pode fazer quando você não quer ser muito preciso sobre o resultado exato que você está tentando conseguir?	

[Comprometimento Prematuro]

Quando você está trabalhando com a notação, você pode realizar o trabalho em qualquer ordem que desejar, ou o sistema força-o a pensar no futuro e tomar certas decisões em primeiro lugar? Se sim, quais as decisões que você precisa fazer com antecedência? Que tipo de problemas isso pode causar no seu trabalho?	

[Consistência]

Onde existem diferentes partes da notação que significam coisas semelhantes, é a semelhança clara da forma como eles aparecem? Por favor, dê exemplos. Há lugares em que algumas coisas deveriam ser semelhantes, mas a notação que os torna diferentes? Quais são?	

[Notação Secundária]

É possível fazer anotações para si mesmo, ou expressar informações que não são realmente reconhecidos como parte da notação? Se ele foi impresso em um pedaço de papel que você pode anotar ou rabiscar, o que você escrever ou desenhar?	

[Abstração]

O sistema dará qualquer forma de definir novos tipos ou termos dentro da notação, de modo que você pode estendê-la para descrever coisas novas ou para expressar suas ideias de forma mais clara ou de forma sucinta? Quais são? O sistema insiste que você comece definindo novos termos antes de fazer qualquer outra coisa? Que tipos de termos?	

Depois de concluir este questionário, você pode pensar em maneiras óbvias de melhorar o sistema como um todo? Quais são?

O sistema como um todo poderia ser melhorado especificamente para suas próprias necessidades?

Seção 5 - Perguntas sobre sub-dispositivos:

Qual é o seu nome?	
Que tipo de notação é usada neste sub-dispositivo?	

Qual a porção do tempo (como porcentagem bruta) você gastou utilizando esse dispositivo?	
Procurando informações	%
Traduzindo quantidades substanciais de informações de outra fonte para o sistema	%
Adicionando pequenas quantidades de informação para uma descrição que você criou anteriormente	%
Reorganização e reestruturando descrições que você criou anteriormente	%
Brincando com novas ideias na notação, sem ter certeza de qual será o resultado	%

De que forma é a notação neste sub-dispositivo diferente da notação principal?

Por favor, marque as caixas onde há diferenças, e escrever algumas palavras que explicam a diferença.

[Visibilidade] É fácil de ver diferentes partes?	
[Viscosidade] É fácil de fazer mudanças?	
[Prolixidade] É a notação sucinta ou prolixa?	
[Operações mentais] Fazer algumas coisas exigem esforço mental duro?	
[Proximidade do mapeamento] É fácil indicar para que serve cada parte?	
[Propensão a erros] É fácil cometer erros ou deslizes?	
Expressividade de papeis] A notação é estreitamente relacionada com o resultado?	
[Dependências Ocultas] Existem dependências visíveis?	
[Avaliação Progressiva] É fácil parar e verificar o seu trabalho até agora?	
[Provisoriedade] É possível esboçar coisas?	
[Comprometimento Prematuro] Você pode trabalhar em qualquer ordem que deseje?	
[Consistência] As semelhanças entre as diferentes partes são claras?	
[Abstração] Você pode definir novos termos ou recursos?	
[Notação Secundária] Você pode fazer anotações informais para si mesmo?	

Como poderia o projeto do sistema pode ser melhorado?

ANEXO A – Projeto de Experimento Controlado

UbiACME - Uma Linguagem de Descrição Arquitetural para Sistemas Ubíquos

Autores: Carlos Alberto Nunes Machado, Eduardo Silva.

A.1 Objetivo

Avaliar a complexidade, a expressividade, a escalabilidade, a compreensão, e o esforço de realização de dado conjunto de alterações em um sistema ubíqua da linguagem de descrição arquitetural (ADL) UbiACME em comparação à (ADL) SysML para a descrição arquitetural de aplicações ubíquas. A escolha de SysML para ser comparada com UbiACME, foi devido a falta de ferramentas adequadas disponíveis em ADLs para computação Ubíqua e SysML do inglês *"Systems Modeling Language"* é uma linguagem de modelagem de sistemas, de propósito geral e de modelagem visual para aplicações de engenharia de sistemas.

A.2 Questões, Métricas e Hipóteses

As questões que permeiam este experimento controlado são:

Q1 - Qual o grau de **complexidade** para descrever uma aplicação ubíqua usando UbiACME em comparação a SysML?

Q2 - Qual o grau de **expressividade** em relação a características de computação ubíqua para descrever suficientemente sistemas de Computação Ubíqua usando UbiACME em comparação a SysML?

Q3 - Qual o grau de **escalabilidade** para descrever sistemas de Computação Ubíqua usando UbiACME em comparação a SysML?

Q4 - Qual o grau de**esforço para realizar** um dado conjunto de alterações em sistemas de Computação Ubíqua usando UbiACME em comparação a SysML?

Q5 - Qual o grau de **compreensão** com relação à descrição arquitetural em sistemas de Computação Ubíqua usando UbiACME em comparação a SysML?

Dimensões Cognitivas usadas para avaliar as questões anteriores, esta descritas na Tabela 8.2.

As métricas usadas para cada questão (Q1 a Q5) respectivamente são:

M1 - Todas as dimensões cognitivas são utilizadas para avaliar a questão Q1. As dimensões: Propensão a Erros, Operações Complicadas, Prolixidade, Viscosidade, Dependencias Ocultas e Comprometimento Prematuro, afetam a notação de forma negativa, as demais são desejáveis.

M2 - A expressividade está relacionada as seguintes dimensões: Abstração e Expressividade de Papéis. Todas as dimensões contribuem positivamente para a expressividade.

M3 - A escalabilidade é afetada negativamente pela Prolixidade, Propensão a erros e Operações Complicadas. São características desejáveis a uma boa Notação secundária, um alto grau de Avaliação Progressiva e uma boa Visibilidade.

M4 - As dimensões Viscosidade, Propensão a erros, Operações Complicadas, Dependências Ocultas e Comprometimento Prematuro afetam, quando em graus elevados, negativamente o sistema que utiliza a notação, para realização de mudanças em produtos já existentes. Já as dimensões Avaliação Progressiva e Visibilidade afetam positivamente em quaisquer graus.

M5 - As dimensões Visibilidade, Expressividade de papeis, Proximidade do Mapeamento e Consistência afetam positivamente a compreensão da linguagem, por auxiliarem na compreensão de produtos existentes. Dependências Ocultas, por outro lado, afetam negativamente, prejudicando a compreensão.

As hipóteses nulas e alternativas correspondentes a cada questão (Q1 a Q5) usando as respectivas métricas (M1 a M5) são sintetizadas na Tabela A.1.

M1 (qualitativa): Complexidade (cpx) em termos da dificuldade proporcionada pela ADL para a descrição da aplicação em computação Ubíqua.

M2 (qualitativa): Expressividade (exp) em termos da representação suficiente dos elementos relacionados à aplicação em computação Ubíqua..

M3 (qualitativa): Escalabilidade (esc) em termos do fato da ADL possibilitar descrever sistemas maiores sem promover um grande impacto na descrição arquitetural, i.e. é possível estabelecer um trade-off aceitável entre o crescimento do tamanho da aplicação em computação Ubíqua.

M4 (quantitativa): Esforço em termos do tempo despendido (ta) para realizar um dado número de alterações sobre a aplicação em computação Ubíqua.

M5 (qualitativa): Compreensão (cmp) da descrição arquitetural da aplicação em computação Ubíqua.

Hipótese nula	**Hipótese alternativa 1**	**Hipótese alternativa 2**
H1:cpx-UbiACME=cpxSysML	H1:cpx-UbiACME<cpxSysML	H1:cpx-UbiACME>cpxSysML
H2:exp-UbiACME=expSysML	H2:exp-UbiACME<expSysML	H2:exp-UbiACME>expSysML
H3:esc-UbiACME=escSysML	H3:esc-UbiACME<escSysML	H3:esc-UbiACME>escSysML
H4:ta-UbiACME=taSysML	H4:ta-UbiACME<ermSysML	H4:ta-UbiACME>taSysML
H5:cmp-UbiACME=cmpSysML	H5:cmp-UbiACME<cmpSysML	H5:cmp-UbiACME>cmpSysML

Tabela A.1: Hipóteses Nulas e Alternativas

A.3 Participantes

Para a realização deste experimento controlado, 4 alunos de Pós-Graduação da área de Ciência da Computação da Universidade Federal do Rio Grande do Norte (Natal, Brasil) serão selecionados com base no conhecimento mínimo sobre linguagens de descrição arquitetural e Computação Ubíqua. Com o intuito de nivelar o conhecimento dos participantes, será realizado um treinamento envolvendo (i) as ADLs UbiACME e SysML. Esse grupo de 4 alunos será dividido em dois grupos, nomeados Grupo I e Grupo II. O Grupo I, com 2 integrantes será responsável por fazer a descrição arquitetural do sistema em Computação Ubíqua utilizando as ADLs em questão, simulando assim o papel de arquitetos de software. Por sua vez, o Grupo II, com 2 integrantes, será responsável por analisar as descrições arquiteturais feitas pelo Grupo I, simulando assim o papel de projetistas de software que recebem as descrições arquiteturais realizadas por arquitetos de software.

A.4 Variáveis Dependentes e Independentes

Como anteriormente mencionado, o objetivo deste experimento controlado é avaliar a complexidade, a expressividade, a escalabilidade, a compreensão, e o esforço de realização de dado conjunto de alterações de um sistema na ADL UbiACME em comparação à ADL SysML para a descrição do sistema em Computação Ubíqua. Neste experimento,

existe uma variável independente, que é a estratégia utilizada para a descrição arquitetural usando ADLs, fator esse possui duas alternativas associadas: (i) realizar a descrição arquitetural utilizando a ADL UbiACME, e; (ii) realizar a descrição arquitetural utilizando a ADL SysML. Além disso, há cinco variáveis dependentes a serem mensuradas no experimento e que estão relacionadas à descrição arquitetural do sistema em Computação Ubíqua utilizando as ADLs em questão: (i) complexidade, em termos da dificuldade proporcionada pela ADL para fazer a descrição; (ii) expressividade, em termos da representação suficiente dos elementos relacionados à Computação Ubíqua; (iii) escalabilidade, de maneira a ser possível estabelecer um trade-off aceitável entre o tamanho da aplicação em Computação Ubíqua e respectivos produtos e o grau de dificuldade para fazer a descrição arquitetural; (iv) esforço, que está associado ao tempo despendido para realizar um dado número de alterações sobre o sistema em Computação Ubíqua, e; (v) facilidade de compreensão da descrição.

A.5 Execução

O experimento controlado será realizado em quatro etapas. Na primeira etapa será realizado um treinamento envolvendo os elementos envolvidos (a saber, as ADLs UbiACME e SysML e o sistema em Computação Ubíqua) com o intuito de nivelar o conhecimento dos participantes. Antes de iniciar a segunda etapa, que consistirá na descrição do sistema em Computação Ubíqua usando as ADLs UbiACME e SysML, o ambiente de execução será preparado, etapa na qual serão instalados os plug-ins do Eclipse IDE referente às ferramentas a serem utilizadas para as descrições arquiteturais. É importante salientar que, para o presente experimento, as ferramentas serão utilizadas apenas para verificação de sintaxe gráfica das descrições arquiteturais, e não para avaliação de questões relacionadas à usabilidade das mesmas. Para dar início à segunda etapa do experimento, os participantes responderão a um questionário inicial que caracteriza o perfil de cada participante e tem por objetivo coletar informações acerca da experiência profissional dos mesmos, habilidades em termos de desenvolvimento de software e familiaridade com os conceitos relacionados a ADLs e Computação Ubíqua. De maneira adicional, parte do modelo do sistema em Computação Ubíqua será fornecida aos membros do Grupo I, bem como dois produtos (de tamanhos diferentes) resultantes da combinação de algumas partes do sistema. Feito isso, os 2 integrantes do Grupo I realizarão a descrição arquitetural do sistema em Computação Ubíqua na ADL UbiACME e os 2 integrantes do Grupo II realizarão a descrição arquitetural do sistema em Computação Ubíqua na ADL SysML. Na terceira

etapa do experimento, após os integrantes do Grupo I realizarem as descrições arquiteturais utilizando as ADLs em questão, eles receberão uma lista com um dado conjunto de alterações que deverão ser realizadas sobre cada descrição da aplicação em computação Ubíqua e dos respectivos produtos. Na quarta e última etapa do experimento, as descrições arquiteturais feitas pelos integrantes do Grupo I serão analisadas pelos 2 integrantes do Grupo II, que tentarão, a partir de tais descrições, verificar a arquitetura baseado nos documentos de especificação, a fim de verificar o grau de compreensão das descrições. Por fim, os participantes irão responder um questionário de dimensões cognitivas, que será utilizado pelos pesquisadores para avaliar as notações. Análise dos resultados Após a realização do experimento controlado, os resultados serão analisados em duas perspectivas. Com base no feedback provido pelos participantes através dos questionários aplicados, serão verificadas, comparativamente, a presença de cada dimensão em cada uma das notações utilizadas. Essa comparação será então submetida as métricas, para que seja possível então ranquear as notações a partir das dimensões encontradas. Dessa forma, será possível verificar as hipóteses e produzir um conjunto de conclusões para o experimento. Por fim, também serão elencados possíveis elementos que se constituam como possíveis ameaças à validade do experimento controlado realizado.