Um modelo para ambientes inteligentes baseado em serviços web semânticos

Crhistian Alberto Noriega Guerra
cnoriega@ime.usp.br

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Flávio Soares Corrêa da Silva

São Paulo, agosto de 2007

Um modelo para ambientes inteligentes baseado em serviços web semânticos

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Crhistian Alberto Noriega Guerra
e aprovada pela Comissão Julgadora.

Banca Examinadora:

- Prof. Dr. Flávio Soares Corrêa da Silva(orientador) - IME-USP.
- Prof. Dr. Fabio Kon - IME-USP.
- Prof. Dr. Sergio Takeo Kofuji - POLI-USP.

Agradecimentos

Em primeiro lugar agradeço a Deus por me ajudar a enfrentar todos os desafios, e me dar tudo o que conquistei até agora. Agradeço, também por colocar no meu caminho todas as pessoas que me ajudaram a chegar onde estou. Meus pais, Ninfa e Victor, que me deram os valores e o exemplo para viver o dia a dia. Minha irmã, Heydi, que me ensinou uma maneira diferente de enfrentar os problemas. Minha avó, Alejandrina, que me ensinou o poder da vontade humana. Meus amigos, Ivan, Gustavo, Juan Carlos, Joan, Pablo, Gerard e Alexis, que me ajudaram e trouxeram alegria nos momentos difíceis.

Uma pessoa importante no meu caminho a quem sempre agradecerei é meu orientador, o professor Flávio, que a pesar da formalidade de chamá-lo de professor o considero amigo.

Resumo

Um ambiente inteligente é um sistema de computação ubíqua e sensível ao contexto onde os sistemas computacionais embutidos no ambiente, a comunicação entre dispositivos e o ambiente, e a acessibilidade aos serviços do ambiente são transparentes ao usuário.

O presente trabalho tem como objetivo propor um modelo para ambientes inteligentes baseado em *serviços web semânticos*, em que os *serviços* disponíveis para os dispositivos do ambiente são proporcionados como serviços web e a interação *dispositivo - ambiente* é feita em um contexto de computação móvel, onde a disponibilidade dos serviços e a informação de contexto do dispositivo mudam freqüentemente.

No modelo proposto todas as funcionalidades do ambiente são fornecidas como serviços. Estes serviços são descobertos e executados automaticamente com a finalidade de ajudar o usuário a desenvolver tarefas específicas, permitindo ao usuário se concentrar nas tarefas e não na interação com o ambiente. O modelo se fundamenta na oferta de serviços dirigida pela tarefa a ser desenvolvida, o que é conhecido como *Task-driven Computing*.

Por outro lado, para a automação do processo de descoberta e execução dos serviços é necessário ter uma especificação não ambígua da semântica dos serviços. Empregamos para isso a ontologia WSMO (Web Services Modeling Ontology) que fornece os elementos necessários para a descrição dos serviços disponíveis no ambiente e o contexto do dispositivo.

Finalmente, como prova de conceitos do modelo proposto, foi implementado um ambiente inteligente para uma biblioteca. A ativação de um ambiente inteligente baseado no modelo proposto se baseia na definição de ontologias, descrição semântica dos serviços no ambiente e a implementação de serviços web tradicionais.

Palavras-chave: ambientes inteligentes, serviços web semânticos, computação dirigida por tarefas, sistemas de manutenção da verdade.

Abstract

A smart environment is a system computing ubiquitous computing and context awareness, in which the computational systems embedded in the environment, the communication between devices and the environment, and the accessibility to services are transparent to the users.

The aim of this work is to propose a semantic web services based model for smart environments, in which *services* are offered to devices as web services and the *device - environment* interactions are based on a mobile computing environment, in which the contextual information and availability of services change frequently.

In the proposed model all functionalities in the environment are offered as services. These services are automatically discovered and executed to support the user in a specific task, allowing to the user to focus on his task and not in the interactions with the environment. The model is based on a task-driven offer of services and on *task-driven computing*.

To automate the discovery and execution of services, we need a non-ambiguous specification of the semantic of services. We use the WSMO ontology (Web Services Modeling Ontology), which provides the required elements for description of the services in the environment and the context device.

Finally, as a conceptual proof of the proposed model, we implemented a smart environment for a library. In the proposed model the activation of a smart environment is based in the ontologies definition, semantic description of the services in the environment, and the implementation of typical web services.

Keywords: smart environments, semantic web services, task-driven computing, truth maintenance systems.

Sumário

1 Introdução **1**
1.1 Motivação 4
1.2 Justificativa 5
1.3 Objetivos 7

2 Conceitos Gerais **9**
2.1 Computação Ubíqua 10
2.2 Sensibilidade a contexto 12
2.2.1 Modelagem do Contexto 14
2.3 Interação implícita Homem-Computador 16
2.4 Computação dirigida por tarefas 18
2.5 Sistemas de manutenção da verdade 20
2.5.1 ATMS 21
2.5.2 Propriedades das rotulações do ATMS 23

3 Ambientes Inteligentes **25**
3.1 Definição de ambientes inteligentes 26
3.2 Projeto Oxygen 27
3.2.1 Tecnologias de dispositivos 28
3.2.2 Tecnologias de rede 28
3.2.3 Tecnologias de software 30
3.2.4 Tecnologias de percepção 31
3.2.5 Tecnologias de usuários 32
3.3 Projeto Aura 33
3.3.1 Gerenciador da tarefa (Prism) 34
3.3.2 Fornecedores de serviços 36
3.3.3 Observador de contexto 36
3.3.4 Gerenciador do ambiente 36
3.4 Projeto Gator Tech: Smarthouse 37

3.4.1 Camada física 37
3.4.2 Camada de sensores 38
3.4.3 Camada de serviços 38
3.4.4 Camada de conhecimento 39
3.4.5 Camada de gerenciamento do contexto 39
3.4.6 Camada de aplicação 39
3.5 Projeto MoCA 40
3.5.1 Monitor 41
3.5.2 CIS - Context Information Service 42
3.5.3 DS - Discovery Service 42
3.5.4 CS - Configuration Service 42
3.5.5 LIS - Location Information Service 43
3.5.6 SRM - Symbolic Region Manager 43
3.6 Projeto WSAmI 43
3.6.1 Módulos e serviços do middleware 46
3.6.2 Descoberta de serviços web 48
3.7 Discussão dos projetos 50

4 Serviços Web Semânticos em Ambientes Inteligentes 54
4.1 Serviços web semânticos 54
4.2 WSMO na descrição de serviços 59
4.2.1 Ontologias 60
4.2.2 Web Services 64
4.2.3 Metas 68
4.2.4 Mediadores 70
4.3 WSML e F-Logic 72
4.3.1 Web Services Modeling Language - WSML 72
4.3.2 F-Logic 73

5 Modelo Proposto 78
5.1 Arquitetura do modelo 79
5.1.1 Repositórios 81
5.1.2 Gerenciador de contexto 83
5.1.3 Descobridor de serviços 83
5.1.4 Invocador de serviços 84
5.1.5 Gerenciador de localização 84
5.1.6 Base de conhecimento 84
5.1.7 Processo de descoberta e execução 85
5.2 Descoberta de serviços 86
5.2.1 Mecanismo de descoberta 87

5.2.2 Montagem do ATMS 89
5.2.3 Funções do ATMS 94
5.3 Execução dos serviços 95
5.3.1 Propriedades de execução 95
5.3.2 Criação de objetos Java 96
5.4 Serviços da arquitetura 97

6 Implementação do Modelo e Experimentos 101
6.1 Implementação do servidor do ambiente 101
6.1.1 Projeto do servidor do ambiente 102
6.1.2 Serviço web do servidor do ambiente 105
6.1.3 Motores para os dispositivos do ambiente 109
6.2 Implementação do cliente do ambiente 111
6.2.1 Interface genérica do usuário 111
6.2.2 Sincronização com a base de conhecimentos 114
6.3 O ambiente de teste 115
6.3.1 Informação contextual 116
6.3.2 Ontologias implementadas 117
6.3.3 Serviços implementados 118
6.3.4 Descoberta e execução dos serviços 120
6.3.5 Interface cliente para o ambiente da biblioteca 125

7 Conclusões e Trabalhos Futuros 126
7.1 Cenários para o modelo proposto 126
7.1.1 Um ambiente reativo à presença de pessoas 126
7.1.2 Um ambiente para uma área extensa 129
7.1.3 Um ambiente com composição de serviços 131
7.2 Conclusões 132
7.3 Trabalhos futuros 135
7.3.1 Um middleware inteligente 135
7.3.2 Filtragem colaborativa para ambientes inteligentes . . 137

Lista de Figuras

2.1 Exemplo de um ATMS. 24

3.1 Arquitetura do Aura. 35
3.2 Arquitetura do Gator Tech. 38
3.3 Arquitetura do MoCA. 41
3.4 Esqueleto da interface abstracta WSAmI. 44
3.5 Esqueleto da interface concreta WSAmI. 45
3.6 União das interfaces abstrata e concreta do serviço web. . . . 46
3.7 Arquitetura do WSAmI. 47
3.8 Algoritmo de busca de instâncias de serviços web. 50

4.1 Diagrama entidade-relação dos elementos da WSMO. 60
4.2 Classe `Ontologia`. 61
4.3 Exemplo de especificação de uma ontologia WSMO 62
4.4 Classe `Concept` e `Attribute`. 63
4.5 Classe `Service`. 64
4.6 Descrição WSMO do serviço `Library`. 65
4.7 Classe `Capability` e classe `Axiom`. 66
4.8 `Capability` do serviço `Library`. 67
4.9 Classe `Interface`. 68
4.10 Exemplo da `Choreography` do `Library` serviço web. 69
4.11 Regra de transição do serviço `Library`. 70
4.12 Exemplo da especificação de Metas 71
4.13 Um programa em *F-Logic* 74

5.1 Componentes da arquitetura proposta. 80
5.2 Arquitetura do modelo proposto. 82
5.3 Regra de descoberta de serviços para o serviço `Library`. . . . 88
5.4 ATMS e a base de conhecimentos do dispositivo. 92
5.5 Relação causa-efeito do ATMS. 93

5.6 Atualização da base de conhecimentos e o ATMS 95
5.7 Interface semântica do serviço `Library`. 96

6.1 Diagrama de classes do servidor do ambiente. 103
6.2 Interação entre os principais componentes do modelo. 106
6.3 Interação de componentes para a descoberta de serviços. . . . 107
6.4 Interação entre uma aplicação cliente e o servidor do ambiente. 110
6.5 Servidor do ambiente. 111
6.6 Tela de registro. 112
6.7 Tela principal para o usuário. 113
6.8 Tela de ajuda pro-ativa. 113
6.9 Tela de múltiplos objetos. 114
6.10 Dispositivo com a aplicação cliente do ambiente. 115
6.11 Mapa das áreas simbólicas na biblioteca no *Mapper* do MoCA. 116
6.12 Roteador Wi-Fi da rede. 117
6.13 Tela do usuário para o ambiente da Biblioteca 125

7.1 Arquitetura do middleware sugerido. 136

Capítulo 1

Introdução

A interação com computadores nas atividades diárias se justifica desde que os sistemas computacionais ajudem nas atividades dos usuários. Um sistema computacional deve fornecer a seus usuários os *serviços* necessários para completar suas tarefas, considerando que um serviço é um fornecedor de recursos e processamento. Esses serviços devem ser consistentes com a tarefa do usuário e ser os mais adequados aos dispositivos disponíveis para cada usuário. Os *ambientes inteligentes* têm por objetivo tornar transparente para pessoas em um ambiente a interação com sistemas computacionais.

Por exemplo, um aluno que ingressa pela primeira vez em uma biblioteca da universidade pode acessar através do seu *PDA*[1] os *livros disponíveis* de uma *lista de livros* das *disciplinas* em que o aluno se encontra matriculado. Também o aluno pode acessar a *lista de livros relacionados* com algum livro de sua preferência, ou com o *tema* que vem sendo tratado na sua disciplina. A qualquer momento, o aluno pode receber a *localização* do livro selecionado dentro da biblioteca, relativa à localização do próprio aluno.

Para conseguir a transparência desta interação, todo esse processo deve ser feito de maneira automática. O ambiente deve achar os serviços necessários baseado no objetivo informado pelo aluno e no contexto em que ele se encontre –por exemplo, se o aluno se aproximar de algum livro, as informações daquele livro podem ser para ele apresentadas. Para a implementação deste tipo de ambiente, segundo [1, 18, 37] é necessário: (1) ubiqüidade da computação, (2) ubiqüidade da comunicação e (3) interfaces inteligentes com os usuários. Além desses pontos, segundo [42], estes ambientes devem ter a capacidade de coletar e processar informação contextual dos usuários e dispositivos.

[1] Personal Digital Assistant

A implementação destes pontos requer a colaboração de diferentes áreas da Ciência da Computação. Por exemplo, para que todos os sistemas envolvidos possuam a mesma interpretação das palavras chave envolvidas nas consultas (no exemplo anterior, as palavras em itálico), podem ser utilizadas tecnologias associadas à *web semântica* e *ontologias*. Protocolos e tecnologias da computação móvel podem dar suporte à ubiqüidade da comunicação [10]. Sistemas de middleware específicos podem ser implementados para permitir a ubiqüidade da computação e adaptabilidade do ambiente e dispositivos [7]. Finalmente, sistemas distribuídos podem permitir coletar e processar informação contextual dos dispositivos no ambiente [40].

Baseados no estudo sobre arquiteturas baseadas em serviços apresentados em [30, 44] e nos projetos sobre ambientes inteligentes apresentados no Capítulo 3 desta dissertação, propomos uma arquitetura baseada em serviços para nosso modelo para ambientes inteligentes. Especificamente, usamos os *serviços web* como implementação dessas arquiteturas, sendo que um serviço web é um provedor de serviços e dados no ambiente. Para permitir a descoberta e execução automática dos serviços mais adequados para atender cada usuário, os serviços devem ser especificados de forma não ambígua. A *web semântica* [15] incorporada aos serviços web tem esse objetivo, surgindo assim os *serviços web semânticos* [4]. Além disso, uma aplicação cliente pode precisar de diferentes serviços para completar sua operação. Nesse caso, é necessário coordenar a interoperabilidade entre a aplicação cliente e os diferentes serviços envolvidos, mantendo a consistência entre o conjunto de serviços executados e o objetivo do usuário, como apresentado em [38, 46].

Uma ontologia para serviços web semânticos é a ontologia *Web Services Modeling Ontology - WSMO* que faz parte do arcabouço *Web Services Management Framework - WSMF* [17] para serviços web semânticos. A WSMO define quatro elementos necessários para a especificação de serviços web: (1) meta, (2) descrição do serviço, (3) mediadores e (4) ontologias. A WSMO baseia-se em *F-Logic* [26], que por sua vez está baseada em lógica descritiva. A descrição semântica dos serviços no modelo proposto é feita empregando esta ontologia.

Por outro lado, em [27] encontramos uma proposta que considera um ambiente inteligente como um jogo de computador inteligente (*Game IA*). Nessa proposta o ambiente –assim como um jogo de computador– deve reagir de maneira adequada às entidades que neles se encontram. O ambiente deve rastrear e interpretar o comportamento das entidades frente a determinadas situações. Uma característica dos ambientes inteligentes é a sua extrema variabilidade, que resulta da mobilidade das pessoas e da disponibilidade dos serviços: o contexto do usuário, a oferta e a necessidade de serviços

mudam o tempo todo [42].

Para lidar como esta variabilidade, no presente trabalho sugerimos o uso de um sistema de manutenção de verdade baseado em hipóteses [12] (*Assumption-based Truth Maintenance System - ATMS*) para manter uma visão atualizada dos serviços disponíveis no ambiente. O ATMS mantém de maneira ágil a relação entre as metas do usuário e os serviços a serem executados, ou mais especificamente as relações entre as pré-condições/entradas e efeitos/pós-condições que resultam da execução de um serviço. Quando o usuário muda de contexto ou quando novos serviços são disponibilizados, as relações atualizadas permitem achar novos serviços para o novo contexto. O ATMS acha os conjuntos de pré-condições mínimas que produzem certo efeito, considerando a descrição do comportamento do serviço. A descoberta de serviços se baseia então em achar um conjunto de pré-condições que produzam o efeito desejado.

Finalmente, como prova de conceito do modelo proposto, apresentamos o ambiente inteligente desenvolvido para uma biblioteca. Para a ativação de um ambiente inteligente empregando o modelo proposto é necessário: (1) definição das ontologias para a descrição dos serviços, (2) especificação semântica dos serviços no ambiente e (3) implementação dos serviços como serviços web.

Nas próximas seções apresentamos a motivação, justificativa e objetivos desta dissertação. O Capítulo 2 apresenta os conceitos gerais relacionados a ambientes inteligentes e aplicações sensíveis ao contexto. O Capítulo 3 apresenta o conceito de ambientes inteligentes e os requisitos para sua implementação. Também são apresentados projetos de ambientes inteligentes existentes. No Capítulo 4 apresentamos a ontologia WSMO, mostrando detalhes na especificação semântica dos serviços para o ambiente. Também nesse capítulo apresentamos um middleware para ambientes inteligentes baseado em serviços web. No Capítulo 5 apresentamos a arquitetura do sistema e uma descrição do modelo proposto. No Capítulo 6 apresentamos detalhes da implementação do modelo e os experimentos realizados. Finalmente, no Capítulo 7 apresentamos a discussão dos resultados e as conclusões obtidas com base na criação de um ambiente inteligente. Também são apresentados cenários de ambientes inteligentes onde o modelo proposto pode ser utilizado e trabalhos futuros que podem ser realizados a partir do modelo proposto.

1.1 Motivação

Seguindo os objetivos da computação ubíqua [20, 53], é necessário também desenvolver a *ubiqüidade dos serviços* para os usuários em um ambiente. Uma vez estabelecida uma tarefa a ser executada pelo usuário, os serviços devem ser descobertos e executados automaticamente em função da tarefa e do contexto do dispositivo e do usuário. O usuário não deve perceber que existe um sistema computacional com o qual esteja interagindo. Ou seja, conforme apresentado em [52], o usuário não deve se preocupar em encontrar ou executar os serviços necessários de suporte à sua tarefa.

Em ambientes que são novos para um usuário, os mecanismos para a descoberta e execução automática dos serviços mais relevantes assumem maior importância. Do ponto de vista do usuário, isso implica em o ambiente reagir às suas necessidades.

Com a finalidade de conseguir essa ubiqüidade dos serviços, empregamos serviços web como mecanismo de comunicação entre dispositivos móveis em ambientes inteligentes. Através dos serviços web sobre os protocolos *SOAP/HTTP*, aproveitamos a ubiqüidade da Internet para a comunicação entre os dispositivos e o ambiente. Por outro lado, a incorporação da web semântica [15] a serviços web permite a especificação não ambígua da semântica de serviços web. Sucintamente, a especificação semântica de serviços web define as entradas e saídas do serviço e como deve ser executado [4]. Isto decorre de os serviços web semânticos permitirem automatizar as seguintes tarefas [4, 5]:

1. *Localizar um serviço (Descoberta):* Uma aplicação pode automaticamente descobrir um serviço web apropriado. As tecnologias atuais para serviços web, (como WSDL e UDDI) não permitem a uma aplicação determinar o *quê* fornece um serviço web. A web semântica descreve as propriedades e capacidades de tal forma que a aplicação cliente pode determinar automaticamente como atingir seu propósito.

2. *Executar um serviço (Execução):* Uma aplicação deve automaticamente determinar como ativar ou executar um serviço. Por exemplo, se o serviço for um procedimento com múltiplas chamadas, a aplicação precisa saber como interagir com o serviço para completar a seqüência.

3. *Empregar serviços em conjunto (Composição):* Uma aplicação pode encadear serviços web para atingir um certo objetivo. Os serviços web devem interagir uns com os outros para unir resultados em algum

resultado válido. Dessa forma uma aplicação pode criar novos serviços web compondo serviços já existentes.

4. *Acompanhar a execução do serviço web (Monitoração):* Uma aplicação precisa verificar as propriedades e monitorar a execução do serviço web para dessa forma poder tomar certas decisões sobre a consulta feita.

Assim, os serviços web semânticos nos permitem descobrir e executar os serviços de forma transparente para o usuário, para ajudar o usuário a executar uma tarefa no ambiente. Assim, o usuário pode se dedicar exclusivamente à execução da sua tarefa, sem se preocupar com a descoberta e execução dos serviços adequados.

A motivação neste trabalho é tratar os pontos 1 e 2 no cenário de ambientes inteligentes, ou seja a descoberta automática dos serviços apropriados para uma tarefa em um determinado ambiente e a execução automática desses serviços. Em contraposição com o cenário apresentado pela web semântica, o uso de serviços web semânticos em um ambiente inteligente exige que se levem em conta: (1) as mudanças freqüentes na disponibilidade dos serviços e (2) as mudanças freqüentes de contexto dos usuários e dispositivos [41, 52].

1.2 Justificativa

Em [1, 22] é definido um ambiente inteligente como sendo baseado em três tecnologias: computação ubíqua, comunicação ubíqua e interfaces de usuário inteligentes. Além dessas três tecnologias, em [18] é definido que um ambiente inteligente deve perceber e responder de maneira inteligente à presença de pessoas no ambiente. Para este último ponto, em [41] é mostrado que a implementação de um ambiente inteligente deve ser uma aplicação sensível ao contexto do usuário e do dispositivo. Baseados nestas definições, consideramos que a implementação de um ambiente inteligente requer:

Interação homem-computador: As pessoas não devem perceber que interagem com algum tipo de sistema computacional. A interação homem-computador deve ser o mais natural possível.

Interoperabilidade da comunicação: Qualquer tipo de dispositivo deve ser capaz de se comunicar com o ambiente e com outros dispositivos. O ambiente não deve fazer pressupostos sobre os protocolos de comunicação nem o tipo de dispositivos que o usuário pode ter.

Ubiqüidade da computação: As pessoas não precisam carregar equipamentos especiais para interagir com o ambiente. Recursos computacionais devem estar embutidos no ambiente em objetos com os quais as pessoas convivem diariamente.

Inteligência do ambiente: A maneira como o ambiente interage com os dispositivos deve ser visto de maneira "inteligente" pelos usuários. Um ambiente pode ser puramente reativo ou, como no caso do nosso trabalho, o ambiente pode ajudar ativamente os usuários em alguma tarefa específica.

Escalabilidade do ambiente: Novas funcionalidades devem ser facilmente implantadas no ambiente. Por exemplo, se um novo sensor for instalado no ambiente, este deve ficar disponível automaticamente para os outros dispositivos. Desta maneira, novas funcionalidades podem ser oferecidas pelos mesmos dispositivos no ambiente.

A *interação homem-computador* do modelo proposto segue o modelo de *computação dirigida por tarefas* [52]. Desta maneira, o usuário especifica a tarefa que quer desenvolver no ambiente, e o ambiente fornece e executa automaticamente os serviços adequados para esse fim. Este processo deve considerar mudanças no contexto do usuário e na disponibilidade dos serviços.

Um ambiente inteligente pode cobrir uma área ampla (como o campus de uma universidade) e neste cenário os serviços web são uma boa opção de tecnologia de comunicação. Eles fazem uso da *ubiqüidade da comunicação* (através da Internet), permitindo a um usuário se conectar em qualquer lugar e com qualquer tipo de dispositivo. Atualmente, o HTTP é o protocolo da Internet implementado na maioria de dispositivos móveis [3, 20].

O uso da ontologia WSMO para a modelagem do contexto do usuário e especificação semântica dos serviços nos permite empregar tecnologias da lógica descritiva para a descoberta dos serviços mais adequados à tarefa do usuário, ativando a *inteligência do ambiente*. Além disso, o uso do ATMS para o processo de descoberta se justifica pois nos permite implementar um mecanismo de lógica não-monotônica que mantém de maneira dinâmica as dependências entre hipóteses e efeitos. Esta dinâmica é necessária considerando a visão apresentada em [27], em que a implementação de um ambiente inteligente é caracterizada como um jogo com inteligência artificial. Seguindo esta abordagem, em [23] se apresenta um ATMS simplificado encarregado de manter as ações de um agente no jogo.

A arquitetura baseada em serviços proposta considera que qualquer funcionalidade deve ser fornecida como um serviço, que é um totalmente de-

sacoplado e altamente interoperável [44, 51]. Assim, qualquer dispositivo (fixo ou móvel) pode fornecer funcionalidades através de um serviço, e um disposi-tivo também pode ser controlado através de um serviço. Por isso, o emprego de uma arquitetura baseada em serviços nos permite a *escalabilidade do ambiente*.

1.3 Objetivos

O objetivo principal neste trabalho é propor um modelo e uma arquitetura para ambientes inteligentes baseados em serviços web semânticos, levando em consideração os pontos apresentados na seção anterior sobre a implementação de um ambiente inteligente. Como prova de conceitos do modelo e da arquitetura propostos implementamos um ambiente inteligente para uma biblioteca.

Dividimos o objetivo principal nos seguintes sub-objetivos:

1. Propor uma arquitetura centralizada que dê suporte à descoberta e execução automática de serviços com base nas especificações semânticas dos serviços e o contexto dos usuários.

2. Desenvolver um mecanismo para a descoberta e execução automática de serviços que permita lidar com a variabilidade de contexto dos usuários e a disponibilidade dos serviços.

3. Empregar a WSMO como ontologia para a especificação semântica de serviços oferecidos num ambiente e a descrição de contextos. A especificação desses serviços leva em consideração a especificação do contexto do dispositivo e a especificação do contexto em que um serviço está disponibilizado.

4. Empregar serviços web sobre *SOAP/HTTP*[2] como meio de comunicação ubíqua entre os dispositivos e o ambiente.

Especificamente, a implementação de um ambiente inteligente empregando o modelo proposto possui as seguintes funções básicas:

1. Descoberta e execução automática de serviços com base na informação do contexto do dispositivo e do usuário.

[2] O protocolo principal empregado pela Web é o HTTP - Hypertext Transfer Protocol. O envio de pacotes SOAP sobre esse protocolo permite a execução remota de métodos pela Internet.

2. Atualização dinâmica dos serviços que ajudem o usuário a desenvolver sua tarefa frente a mudanças no seu contexto ou de disponibilidade dos serviços.

3. Orientação pró-ativa do usuário quanto aos serviços. Isto é, quando não for possível achar ou executar um serviço automaticamente, o ambiente dá a orientação necessária ao usuário para escolher e executar o serviço.

Para o caso da implementação do ambiente inteligente para a biblioteca, empregando o modelo e arquitetura proposta, o ambiente fornece as seguintes funcionalidades:

1. Baseado no perfil do usuário (dados do aluno e disciplinas matriculadas) fornecer uma lista de livros relacionados aos temas das disciplinas em que se encontra matriculado.

2. Baseado no contexto do dispositivo (sua localização dentro da biblioteca) fornecer uma lista de livros disponíveis na biblioteca e relacionados aos temas de interesse do usuário.

3. Guiar ao usuário dentro da biblioteca para achar o livro selecionado.

4. A todo momento, proporcionar ao usuário todas as informações de um livro: localização, disponibilidade, dados dos exemplares e livros relacionados.

Capítulo 2

Conceitos Gerais

Os dispositivos computacionais se encontram em constante evolução, desde a aparição de computadores de grande porte, seguidos dos computadores pessoais, até a atualidade em que um ou vários dispositivos computacionais móveis de pequena escala podem ser transportados por pessoas, como por exemplo *PDAs*, telefones celulares, *Smartphones*, PSP[1], e dispositivos como etiquetas *RFID*[2] que podem ser encontrados em diversos objetos do mundo real.

Com estes avanços na tecnologia de computação, foi criado o conceito de *computação ubíqua* que se referem à integração de dispositivos computacionais no mundo físico das pessoas, permitindo ativar a ubiqüidade da computação, comunicação e informação. No presente trabalho adotamos as definições encontradas em [20] e [53] sobre computação ubíqua.

Neste novo cenário computacional a representação e processamento de informação contextual dos dispositivos e dos usuários assume maior importância, permitindo implementar aplicações *sensíveis ao contexto* [43] e adaptáveis automaticamente a tais contextos. Além disso, a computação promovida por estes dispositivos permite uma nova *interação homem-computador* [41]. Seguindo a visão da computação ubíqua, nesta interação as pessoas não percebem que interagem com sistemas computacionais.

Nesse capítulo são apresentados conceitos relacionados à computação ubíqua, aplicações sensíveis ao contexto e interação implícita homem-computador. Também é apresentado o modelo computacional dirigido por tarefas (*task-driven computing* [52]) adotado pelo projeto Aura [45].

[1] Play Station portátil

[2] Radio-frequency identification

2.1 Computação Ubíqua

Segundo a definição de Mark Weiser[3], a computação ubíqua propõe uma visão contrária à da realidade virtual [53], na qual as pessoas se adaptam a um mundo criado pelos computadores. A computação ubíqua pretende que sejam os computadores a se adaptar ao mundo físico das pessoas.

Os dispositivos devem estar adaptados ao cotidiano dos indivíduos e em harmonia com outros objetos presentes. Tais ambientes são comumente chamados de *Espaços Ativos* ou *Super Spaces*. Segundo Weiser uma das principais características desses ambientes é que a interação entre os usuários e os dispositivos ocorre de forma tranqüila (*Calm Technology* ou *Computação Invisível* [41]). Os usuários não precisam perceber que estão interagindo com máquinas. Assim, o usuário somente recebe os benefícios da interação.

Algumas implicações dessas definições são:

- Os computadores deveriam estar embutidos em objetos físicos manipuláveis pelas pessoas e que se encontrem no ambiente cotidiano delas, como eletrodomésticos e outros;
- A integração dos computadores no ambiente não deveria fornecer uma realidade aumentada aos usuários;
- A interação deveria surgir de maneira natural, sendo os computadores e não as pessoas a mudar seu comportamento. Assim, os computadores deveriam ficar "invisíveis" para as pessoas;
- O usuário não deveria carregar dispositivos complicados para interagir com os ambientes físicos.

Para permitir o desenvolvimento da computação ubíqua é necessário que os dispositivos e/ou aplicações encarregadas da interação com os usuários sejam capazes de entender o ambiente físico em que se encontram [14]; para alcançar as metas da computação ubíqua se faz necessária uma infra-estrutura sofisticada de dispositivos. Isto implica os seguintes pontos:

- Os dispositivos deveriam poder interpretar o contexto em que os usuários se encontram no momento que interagem com o dispositivo;
- Os dispositivos deveriam ser capazes de lidar com as mudanças do ambiente físico. A computação móvel trata desse problema mas do ponto de vista dos computadores, estabelecendo protocolos de comunicação e middleware de contexto, como o apresentado em [9];

[3] considerado o pai da computação ubíqua.

- Os dispositivos deveriam estar interconectados para poder ter uma representação consistente do mundo físico e poder fornecer informação consistente aos usuários.

A computação ubíqua, no referente às tecnologias computacionais, tem a visão de poder acessar e processar dados em qualquer momento, qualquer lugar e por qualquer dispositivo. Nesse sentido, a computação ubíqua se baseia nos seguintes princípios [20]:

1. *Descentralização:* Isto implica na distribuição das responsabilidades de processamento entre uma variedade de pequenos dispositivos, em vez de um computador centralizado. A descentralização traz consigo o problema da sincronização e atualização de dados distribuídos entre os dispositivos.

2. *Diversificação:* Esse princípio envolve a funcionalidade dos dispositivos. A criação de dispositivos implica em compartilhar funcionalidades entre dispositivos: dever-se-ia poder executar as mesmas aplicações em dispositivos com a mesma funcionalidade. Por exemplo, uma página web poderia ser consultada usando um telefone celular ou um fax que possua uma tela adequada.

3. *Conectividade:* Implica na visão de conectividade ilimitada. Dispositivos fabricados com diferentes funcionalidades e tecnologias deveriam poder se comunicar entre si. Esse paradigma implica o uso de tecnologias de comunicação, protocolos como WAP, IrDA, Bluetooth e UMTS que facilitam a comunicação entre diversos dispositivos. A conectividade também implica na necessidade de mecanismos que permitam a incorporação dinâmica de dispositivos e serviços. Jini (Java), UPnP (Microsoft) e SLP são exemplos desses mecanismos.

4. *Simplicidade:* Os paradigmas anteriores devem surgir de maneira natural e simples. Interfaces simples com os diferentes dispositivos devem facilitar a interação com os sistemas. Interfaces simples não implicam interfaces primitivas. Aplicações com reconhecimento de voz são exemplos dessas interfaces.

A implementação desses pontos implica na necessidade de colaboração de várias áreas da ciência da computação, já que esta interação é feita em diferentes níveis de conceitualização, dos protocolos adequados de comunicação e localização até a inferência das intenções dos usuários. De maneira geral,

é necessária a intervenção das seguintes áreas da computação para esse fim: (1) sistemas distribuídos, (2) interação homem-computador, (3) computação móvel, (4) inteligência artificial e (5) visão computacional.

Neste trabalho, no que se refere ao ponto 3 (sobre conectividade entre dispositivos), empregamos como tecnologia de comunicação distribuída os serviços web. A execução remota de serviços web está implementada sobre a troca de pacotes *SOAP* e sobre o protocolo *HTTP*, que é o protocolo principal da Internet [3].

2.2 Sensibilidade a contexto

Segundo a definição em [14], contexto é qualquer informação que pode ser empregada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar ou um objeto que é considerado relevante na interação entre um usuário e uma aplicação. Em poucas palavras, contexto significa *informação da situação* da entidade.

Em um determinado ambiente, o contexto de um dispositivo é capturado do ambiente em que ele se encontra a partir de um conjunto de sensores capazes de perceber estímulos físicos do ambiente como movimentação, temperatura, nível de ruído, largura de banda, etc. A seguinte lista mostra alguns exemplos de informação de contexto:

- Identidade;
- Informação espacial, localização, orientação, velocidade, aceleração;
- Informação temporal, hora do dia, período do ano;
- Informação do ambiente, temperatura, qualidade de sinal, luminidade, ruído;
- Situação social, quem é o usuário, pessoas que estão próximas;
- Recursos livres, dispositivos e *hot spots*;
- Estado dos recursos, bateria, largura de banda, tipo de tela;
- Agentes e horários.

Segundo [42] estes exemplos podem ser classificados em três grupos de contexto: (1) contexto computacional: informação relevante para o processamento computacional, largura de banda, capacidade computacional, energia, tempo de resposta, etc; (2) contexto do usuário: referente ao perfil do

usuário, permissões, função do usuário, etc; (3) contexto físico: informação referente ao ambiente físico da entidade, como temperatura, ruído, etc.

A percepção do contexto ajuda a computação ubíqua a entender o ambiente físico em que uma entidade se encontra. Assim, é necessário modelar, capturar e interpretar a informação referente aos diferentes tipos de contexto. Por exemplo, seria de grande ajuda poder inferir o tipo de aplicação a ser executada num dispositivo baseado na largura de banda disponível, tipo de dispositivo ou perfil do usuário.

Uma aplicação é sensível ao contexto se emprega o contexto para prover informação e/ou serviços *relevantes* aos usuários [11]; a relevância de serviços e dados pode ser interpretada segundo o domínio de aplicação, mas no sentido mais geral se associa à tarefa do usuário conforme apresentado em [52].

O mecanismo de oferta desses serviços ou informação caracteriza como é construída uma aplicação sensível ao contexto, dependendo de se o fornecimento é manual ou automático, se é feito algum processamento ou é solicitada alguma informação. Com base nessa interação podemos classificar da seguinte forma o modo de interação com os serviços e informação [43]:

- Seleção por proximidade;
- Reconfiguração automática do contexto;
- Informação contextual e comandos;
- Ações ativadas pelo contexto.

Segundo esta classificação é possível determinar o momento em que será empregada a informação de contexto por alguma aplicação, podendo ser: (1) no momento em que a informação do contexto muda são informadas à aplicação cliente as mudanças na informação; (2) a aplicação requer explicitamente a informação necessária do contexto. Ambas abordagens têm problemas e vantagens com respeito uma à outra. O mais comum é empregar as duas abordagens numa mesma aplicação, dependendo da funcionalidade desejada.

Uma aplicação sensível ao contexto pode ser [8]: (1) ativa: a aplicação adapta o seu comportamento automaticamente ao contexto percebido; (2) passiva: a aplicação mostra ao usuário a informação de acordo com o contexto.

É comum que um determinado dispositivo mude de contexto segundo as tarefas e necessidades dos usuários. Em dispositivos móveis é importante acompanhar a seqüência de contextos em que se desenvolve o dispositivo. Isso

pode ajudar a oferecer informação relevante ao usuário ou mesmo prever o curso de ações do usuário [11].

O processamento da informação proveniente do contexto é um ponto chave em aplicações sensíveis ao contexto. Em aplicações reais, os componentes encarregados da fusão/interpretação dos dados do contexto estão localizados numa camada entre a camada de aplicação e a camada de transporte, como middleware de contexto que processa a informação de camadas inferiores para construir informação mais relevante para usuários ou aplicações em camadas superiores, como é o caso apresentado em [9, 40].

2.2.1 Modelagem do Contexto

Uma vez capturada a informação do meio físico, é necessário fazer a modelagem dessa informação. Nessa modelagem são definidos como pontos básicos os nomes, propriedades e atributos de todas as entidades relevantes para uma aplicação que faz uso do contexto de um dispositivo.

É sobre essa modelagem que muitas abordagens operam para conseguir inferir nova informação e poder fornecer dados e serviços aos dispositivos. Assim, o grau de refinamento e precisão na modelagem determina a percepção do ambiente pela aplicação.

Alguma técnicas empregadas na modelagem de contexto são [47]:

1. Duplas objeto-valor, semelhantes ao modelo empregado pelo LIME (*Linda in a Mobile Environment*)[10, 34];

2. Markup Scheme Models, baseados em XML para a especificação das propriedades e atributos das entidades. O modelo mais empregado é o *Composite Capabilities/Preferences Profiles(CC/PP)*[4] definido pelo W3C;

3. Modelos baseados em grafos para a representação das relações entre as entidades, entre eles o *Contextual Extended ORM*;

4. Modelos Orientados a Objetos, que fazem uma modelagem empregando o paradigma orientado a objetos, facilitando a reutilização e encapsulamento de propriedades. Esse modelo é empregado pelo TEA Project[5];

5. Modelos Baseados em Lógica, que facilitam a inferência de contextos e empregam diferentes lógicas para a representação dos conceitos.

[4] `http://www.w3.org/Mobile/CCPP/`
[5] `https://www.themeit.com/developmentguidelines.htm`

Alguns exemplos desse modelos podem ser encontrado na página do AAAI[6];

6. Modelos Baseados em Ontologias, que empregam ontologias como uma representação do conceitos. Nesse caso é necessário especificar uma linguagem e com ela as entidades e suas propriedades. A ontologia CoOL *Context Ontology Language*[48] é um exemplo dessa abordagem.

Em [14] são apresentadas as características principais que uma técnica para modelagem de contexto deveria apresentar:

- Permitir a composição e administração de modelos de contextos dinâmicos em termos de tempo, nos quais a topologia da rede e recursos mudam no tempo;
- Permitir validação de conhecimento parcial em situações em que não é possível validar todo o conhecimento do contexto. Isso possibilita tratar com informação incompleta e ambígua;
- Permitir o tratamento de informação com diferentes níveis na qualidade de informação; a qualidade de informação muda segundo os sensores empregados;
- Possuir um bom nível de formalização, pois sua sintaxe e semântica estão bem definidas;
- Ser aplicado em modelos de ambientes já existentes, como serviços web.

Como um exemplo de uma aplicação dirigida pelo contexto do usuário temos o caso de um estudante numa biblioteca munido de um PDA que requer a informação de livros que o possam ajudar nas suas disciplinas. Dependendo da bibliografia dos cursos em que se encontra matriculado o aluno, o ambiente lhe fornece a informação e localização dos exemplares do livro que ele precisa. Além disso, dependendo da proximidade do estudante de uma área temática da biblioteca (que o ambiente pode interpretar como o interesse por aquela área) o ambiente lhe fornece uma lista de livros relacionados.

Nesse exemplo, as informações fornecidas pelo sistema dependem do contexto do usuário, por exemplo da proximidade do estudante a alguma área da biblioteca (localização do estudante e dos livros), dos dados do estudante (perfil do usuário, se é aluno de pós-graduação ou não, as disciplinas matriculadas, etc), e do dispositivo empregado.

[6] `http://www.aaai.org/Press/Reports/Symposia/Fall/fs-95-02.html`

2.3 Interação implícita Homem-Computador

Os sistemas computacionais tradicionais possuem a idéia de interação *explícita* entre o usuário e os sistemas [41]. O procedimento básico nesse modelo é resumido em: (1) o usuário solicita ao sistema fazer uma determinada ação, (2) a ação é executada pelo computador mediante interfaces determinadas (linha de comandos, reconhecimento de voz, etc.), (3) o sistema devolve a resposta apropriada, que em algumas operações pode ser vazia. As interfaces com esses tipos de sistemas têm a característica que os usuários explicitamente solicitam uma ação, eles têm que fazer manualmente as configurações dos serviços ou programas que pretendem acessar.

Em [29] se apresenta um mecanismo que ajuda o usuário a configurar e compor diversos serviços que podem existir num determinado ambiente. Nesse modelo, quando um dispositivo entra num ambiente novo são apresentados o conjunto de serviços disponíveis e suas possíveis composições. Quando o usuário decide executar uma ação o sistema fornece uma ajuda para a execução e configuração dos serviços escolhidos. Esta ajuda se baseia na informação do dispositivo e do seu contexto. A informação do contexto capturada nesse caso corresponde a: serviços fornecidos por outros dispositivos, objetos no mesmo dispositivo ou serviços fornecidos pelo ambiente de modo geral, etc. O exemplo apresentado neste artigo, mostra os serviços envolvidos na troca de cartões de visita numa reunião. Qualquer dispositivo que deseje pegar o cartão de Bob representa o fato mediante a composição de serviços `[Contact Provider (on Bob's PDA) + Add into Contact List]`. Nesse comando são executados dois serviços, o `Contact Provider` que requer o provedor de cartões de visita (que nesse caso é `Bob's PDA`) que tem que ser proporcionado manualmente pelo usuário, e `Add into Contact List`. Essa última função não recebe nenhum parâmetro e é local ao dispositivo cliente; assim, o PDA de Bob deve prover um serviço que pode ser empregado pelos outros dispositivos para adquirir o cartão de visita.

Esse tipo de interação contradiz os princípios da computação ubíqua sobre a "invisibilidade" dos computadores, e portanto é difícil implementar um ambiente inteligente somente com esse tipo de interação.

Buscamos um tipo de interação em que o usuário não seja ciente da existência de uma entidade que fornece serviços, os quais teriam que ser configurados pelo usuário. A *interação implícita* homem-computador se define como a interação com o ambiente e seus artefatos componentes (software e/ou hardware) com o objetivo de alcançar uma meta determinada; nesses processos o sistema adquire entradas implícitas do usuário e apresenta saídas implícitas [41].

- As entradas implícitas são aquelas que não são fornecidas diretamente pelo usuário. O contexto do usuário é um tipo dessas entradas. Elas não são consideradas como metas mas são capturadas, reconhecidas e interpretadas pelo sistema como parâmetros em seu processamento;

- As saídas implícitas se referem a saídas do sistema e são introduzidas no ambiente e tarefas do usuário sem ser percebidas por ele.

A idéia básica de entrada implícita é que o sistema é capaz de detectar a interação entre o usuário e seu ambiente físico e a tarefa que vem desenvolvendo. Baseado nessa percepção, o sistema pode antecipar as ações do usuário e prover melhor suporte à tarefa que está sendo desenvolvida.

Este tipo de interação permite ao usuário se focar na tarefa que vem desenvolvendo e não na descoberta nem execução dos serviços necessários para atingir seu objetivo [29]. Assim, a computação dirigida por tarefa que será apresentada na próxima seção exige a implementação de um mecanismo para prover ao ambiente interação implícita entre os dispositivos envolvidos no ambiente. Nesse cenário surge o conceito de invisibilidade computacional, que possui duas visões:

- A invisibilidade se refere a que o usuário não perceba o sistema que está usando num determinado ambiente enquanto desenvolve uma tarefa;

- A invisibilidade do sistema está relacionada com a familiaridade do usuário com o sistema. O usuário usa o sistema, mas pelo uso constante deixa de perceber a existência dele.

Muitas vezes as entradas explícitas são claras e levam o sistema a um comportamento determinístico. Por outro lado, as entradas implícitas (como o contexto) são ambíguas e levam a comportamentos diferentes do sistema. Por exemplo, na interação explícita, quando um usuário interage com uma interface, as mudanças apresentadas na interface são compreensíveis pelos usuários por que eles explicitamente pediram ao sistema para fazer alguma tarefa a resposta do sistema é a mesma para os mesmos dados de entrada. Na interação implícita as respostas não são as mesmas pois dependem do contexto, e muitas vezes o usuário não percebe suas própria mudança de contexto. Portanto, essas mudanças devem surgir de maneira natural para o usuário, para evitar a sua desorientação.

2.4 Computação dirigida por tarefas

Os dispositivos móveis permitem que o usuário se movimente por diferentes ambientes, e portanto diferentes contextos. Assim, quando o usuário executa uma tarefa no seu dispositivo móvel, a mudança de contexto exige a reconfiguração dos serviços e informação associados a essa tarefa. Nesse cenário, alguns serviços podem deixar de ser fornecidos e alguns dados podem se tornar inconsistentes. Por outro lado, novos serviços e novas informações podem ficar disponíveis para o usuário, que fica encarregado de reconfigurar manualmente os novos serviços a fim de atingir seus objetivos. Isso implica em ter que lembrar o estado de execução da tarefa e os recursos empregados a fim de poder movimentar seu contexto computacional para um novo ambiente.

Em [52] se apresentam o seguintes pontos a levar em consideração para superar esses problemas:

1. Manter um histórico do contexto computacional;
2. Encontrar novos recursos disponíveis no novo ambiente;
3. encontrar os tipos de serviços e/ou aplicações disponibilizadas no novo ambiente;
4. Configurar os serviços escolhidos para dar suporte aos novos contextos computacionais.

O usuário se vê obrigado a manter um histórico do contexto computacional e configurar os serviços/aplicações adequadas para continuar com sua tarefa. Assim, o usuário se foca nessas atividades em lugar de se focar na sua tarefa. A computação dirigida por tarefas (*Task-driven computing*) tenta resolver esses problemas com base nos seguintes pontos [45]:

- Manutenção do sistema, isto é manter um resumo globalmente acessível de cada tarefa e seu estado executada em cada dispositivo;
- Novos tipos de serviços que permitam sua configuração automática;
- Seleção e configuração automática de serviços que permitam restabelecer os contextos computacionais;
- Orientação pró-ativa na ajuda do usuário para executar sua tarefa;
- Administração automática de serviços dinâmicos e condições de recursos.

A idéia da computação dirigida por tarefas é permitir aos usuários lidar com uma descrição de alto nível das tarefas, em vez de lidar com serviços individuais. Em [52] são definidos dois conceitos para fazer essa distinção:

Tarefa: Representa as intenções do usuário. Pode ser uma tarefa simples que implica uma só atividade, ou tarefas complexas que envolvem a execução de várias atividades. Por exemplo, fazer uma viagem à França envolve achar os vôos disponíveis, reservar hotel, etc. Uma tarefa deve poder ser independente da plataforma e da aplicação.

Sessão: É o conjunto de dispositivos que oferecem os serviços concretos necessários para terminar uma tarefa, podendo ser dispositivos fixos ou móveis.

Assim, a computação dirigida por tarefas associa o quê o usuário deseja fazer (tarefas) com a operabilidade real fornecida pelo sistema (sessões). Os pontos básicos que propõe a computação dirigida por tarefas para conseguir esta associação são:

1. Administração de Tarefas. Exige um mecanismo para descrever formalmente as intenções dos usuários, assim como o estado das tarefas. Isto envolve:

   - Especificação de tarefas independentes da plataforma e aplicação. Isto requer uma linguagem para essa especificação;
   - Administração explícita do contexto computacional. Isto é a administração do estado das tarefas executadas pelos dispositivos. O repositório de todas essas tarefas deve ser acessível e atualizado globalmente.

2. Construção e encapsulamento de serviços. Exige a construção de componentes de software auto-contidos e com uma interface bem definida. Isto facilita o seu emprego como parte de qualquer tarefa. Isto envolve, serviços com especificação bem definida de suas interfaces, bem como dos meta-dados que recebe, para dar suporte à configuração automática de serviços.

3. Administração de Sessões/Serviços. Exige um mecanismo para a configuração, descoberta e coordenação automática entre os serviços fornecidos. Isto envolve, orientação pró-ativa: quando um serviço não pode ser executado por algum motivo, o sistema faz sugestões para o usuário poder decidir o que fazer.

4. Administração do Ambiente. Exige um mecanismo para a recuperação automática de informação de dispositivos e serviços localizados no ambiente físico.

Atualmente existem sistemas de registro e descoberta dinâmica de serviços que implementam algumas dessas funcionalidades, como o Jini (Java), Universal Plug and Play - UPnP (Microsoft) e Service Location Protocol - SLP. Eles foram desenvolvidos para uso em sistemas distribuídos e portanto, só trabalham com serviços individuais. Os mecanismos da computação dirigida por tarefas implementam suas funcionalidades acima dessa camada de serviços. Esses sistemas exigem pré-configuração dos serviços a serem executados nos sistemas clientes, o que faz necessária a configuração manual pelo usuário.

No referente à composição de serviços, existem especificações para sistemas distribuídos que permitem tal composição, como EJB (*Enterprise Java Beans*), CORBA ou Web Services. Esses modelos superam o problema de *como* compor os serviços, mas a pergunta de *quais* serviços compor de maneira automática ainda está em aberto e diversas abordagens vêm sendo exploradas [45, 46, 52].

2.5 Sistemas de manutenção da verdade

Um *sistema de manutenção da verdade* (*Truth Maintenance Systems - TMS*) permite a implementação de lógicas não monotônicas para atualizar uma base de conhecimentos com a adição ou a retirada de fatos. Os sistemas de manutenção de verdade se baseiam em adicionar explicitamente um conjunto de hipóteses (A) a um conjunto de fatos (K). As hipóteses não são necessariamente verdadeiras mas parecem razoáveis, e permitem derivar um conjunto de conclusões (N) [24]. Assim, temos:

$$K \cup A \models N$$

Um TMS normalmente está acoplado a um sistema de inferência encarregado de: (1) ajustar as hipóteses e premissas ao TMS e (2) codificar e enviar justificativas ao TMS. O TMS mantém relações de dependências entre conclusões e hipóteses à medida que o motor de inferência adiciona novas entradas. Assim, o funcionamento do TMS é incremental desde que atualiza seu estado com a adição de novas entradas.

O TMS tem contato direto com o sistema de inferência, mas ele mesmo não faz nenhuma inferência. O funcionamento de um TMS é controlado

por seus algoritmos de organização e propagação de estados. Um nó do TMS representa um *datum* do motor de inferência, ou seja um fato, regra ou fórmula empregada pelo motor de inferência.

Qualquer TMS fornece a seguinte informação:

1. Justificativas para conclusões.
2. Recuperação de inconsistências.
3. Cache de inferências.
4. Raciocínio default.

Todo TMS define os seguintes elementos:

Nós: Representam os *datum* do motor de inferência e podem ser:

- *Hipóteses:* Representam suposições que podem ser adicionadas o retraídas e são fornecidas explicitamente pelo motor de inferência.
- *Premissas:* Representam fatos indiscutíveis que não podem ser alterados. Não precisam de nenhuma justificativa.
- *Contradições:* Representam nós que levam a alguma contradição.

Justificativas: As deduções do motor de inferência são enviadas como restrições ao TMS. Uma justificativa é a forma mais simples de restrição.

Rotulações: Descrevem o estado de crença de um nó.

A maneira como é representado cada um desses elementos define um tipo de TMS. Por exemplo, as rotulações do JTMS (*Justification based Truth Maintenance System*) são valores `IN` ou `OUT`. Já as rotulações do ATMS (*Assumption-based Truth Maintenance System*) são conjuntos de hipóteses.

2.5.1 Sistema de manutenção da verdade baseados em hipóteses

O ATMS é um tipo de TMS baseado em hipóteses. Os algoritmos, elementos e estrutura do ATMS são detalhados em [12]. O ATMS permite manipular múltiplos contextos, o que evita recalcular as rotulações múltiplas vezes. O ATMS não trabalha com adição ou retirada de hipóteses e não leva em consideração o estado de uma hipótese. O ATMS mantém como rotulações de um nó todos os conjuntos de suposições que implicam no nó.

A seguir descrevemos os elementos do ATMS:

Nós: Os nós do ATMS tem a forma $< datum\ \{rot\} >$, onde rot representa a rotulação do nó. Cada nó pode ser:

- Premissa: são justificativas sem antecedentes e são tratadas como qualquer outro nó e têm a forma:

$$< P, \{\} >$$

- Hipótese: são nós especiais que representam as hipóteses enviadas pelo sistema de inferência e têm a forma:

$$< H, \{\{H\}\} >$$

- Contradição: O ATMS não comunica a ocorrência de uma contradição ao motor de inferência, que não precisa manipular contradições. São tratados como `no-good`.

Justificativas: O sistema de inferência envia justificativas ao ATMS que representam cláusulas de Horn e têm a forma.

$$< f, \{\{A, B\}\} >$$

onde A e B são hipóteses e f um fato. Esta justificativa representa a seguinte cláusula de Horn:

$$A \wedge B \models f$$

Um `no-good` é uma justificativa que representa cláusulas negativas. Ou seja, a conjunção dos seus elementos produz uma contradição.

$$<\perp, \{\{H_1, H_2\}\} >$$

onde H_1 e H_2 são hipóteses. Este `no-good` representa a contradição

$$H_1 \wedge H_2 \models \perp$$

Rotulações: As rotulações de um nó são o conjunto de hipóteses que justificam o nó. Por exemplo, para o nó f temos a rotulação $\{\{H_1, H_2\}, \{G_1, G_2\}\}$ que representa o nó:

$$< f, \{\{H_1, H_2\}, \{G_1, G_2\}\} >$$

a rotulação se interpreta como uma disjunção de cláusulas de Horn:

$$H_1 \wedge H_2 \models f$$

$$\vee$$

$$G_1 \wedge G_2 \models f$$

Note-se que as rotulações do ATMS são conjuntos de hipóteses, cada um desses conjuntos é chamado de *environment* (ambiente). No Capítulo 5 empregamos a palavra *environment* para evitar uma confusão com o termo ambiente empregado para ambientes inteligentes. Chama-se de *contexto* ao conjunto de nós que podem ser derivados a partir de um *environment*.

Um `no-good` não pode ocorrer dentro de um *environment* (nenhum `no-good` é sub-conjunto de um *environment*), já que seria produzida uma contradição.

Graficamente para o seguinte conjunto de nós temos o ATMS apresentado na Figura 2.1:

$< A, \{\} >$
$< g, \{\{e, f\}\} >$
$< e, \{\{B, C\}\} >$
$< f, \{\{C, D\}\} >$
$< \perp, \{\{A, B\}\} >$ (`no-good`)

As letras em maiúsculo indicam hipóteses do ATMS, exceto por A que é uma premisa pois não tem hipóteses associadas. As letras em minúsculo são nós enviados à base de conhecimento pelo sistema de inferência.

Um nó n é armazenado em um *environment* E sse n pode ser derivado a partir de E e uma teoria Th, que representam respectivamente as justificativas e `no-goods`:

$$E, Th \models n$$

2.5.2 Propriedades das rotulações do ATMS

Todos os *rótulos* de cada nó apresentam as seguintes propriedades:

Consistência: Uma rotulação L para um nó n é consistente se todos seus *environments* são consistentes.

Suficiência: Uma rotulação L para um nó n é suficiente sse n é conseqüência de todo *environment* $E \cup Th \in L$.

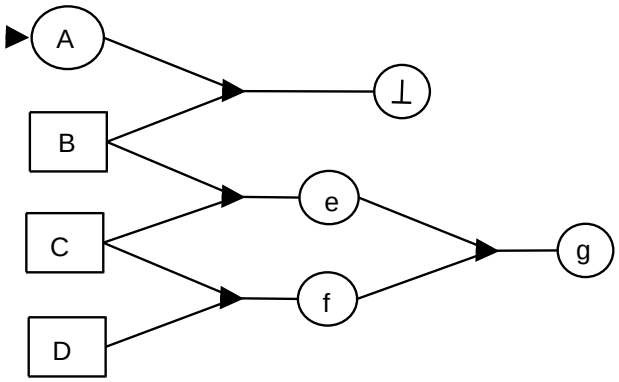

Figura 2.1: Exemplo de um ATMS.

Completude: Uma rotulação L para um nó n é completa sse todo *environment* consistente $E \notin L$ tal que $E, Th \models n$ implica que E é um super-conjunto de algum $E' \in L$ ($E' \subset E$).

Minimalidade: Uma rotulação L para um nó n é minimal sse para todo elemento $E \in L$ não existe $E' \in L$ tal que $E' \subset E$ e $E', Th \models n$.

Cada vez que uma justificativa é adicionada ao ATMS é necessário executar o algoritmo de propagação para atualizar as rotulações dos nós. Em [12] se apresenta o algoritmo completo para a propagação de justificativas no ATMS mantendo rotulações com as propriedades descritas. Esse algoritmo é *NP-Completo*. Para n hipóteses temos $\begin{pmatrix} n \\ k \end{pmatrix}$ possíveis *environments* com k hipóteses.

Capítulo 3

Ambientes Inteligentes

A pesquisa sobre *ambientes inteligentes* tem por objetivo fazer que ambientes reajam de maneira *inteligente* à presença de pessoas, atendendo suas necessidades. Além disso, as pessoas não devem perceber que interagem com um sistema computacional. Os ambientes inteligentes transformam o mundo físico do ambiente em uma interface pela qual os usuários interagem com sistemas computacionais.

Neste capítulo apresentamos as definições de ambientes inteligentes consideradas nesta dissertação. Também apresentamos as arquiteturas e abordagens propostas por projetos em ambientes inteligentes que influenciaram o modelo proposto. Dentre estes projetos temos os projetos *Oxygen* do MIT [35] e *AURA* da universidade Carnegie Mellon [45] como os mais destacados nesta área. O projeto *GatorTech Smarthouse* da Universidade da Florida [19] propõe uma arquitetura baseada em serviços para a implementação de uma casa inteligente. O projeto *MoCA* (Mobile Collaboration Architecture)[40] da Pontifícia Universidade Católica do Rio de Janeiro permite a implementação de aplicações distribuídas sensíveis ao contexto. Finalmente, o projeto *WSAmI* (Web Services in Ambiente Intelligence) [21] explora a capacidade do uso de serviços web em ambiente inteligentes.

Estes projetos oferecem um conjunto de tecnologias que dão suporte ao desenvolvimento de ambientes inteligentes. Este suporte é oferecido em diferentes níveis como: (1) comunicação, para permitir a interconexão entre os dispositivos; (2) adaptabilidade, para permitir criar aplicações adaptáveis ao contexto do dispositivo; (3) interação com o usuário, implementações de interfaces e aplicações para usuários; (4) modelo de computação, implementação de ambientes de execução e sistemas operacionais e (5) informação de contexto e middleware que permitem obter e manipular informação de

contexto dos dispositivos.

Neste capítulo fazemos uma descrição dos aspectos mais relevantes de cada projeto enfatizando os aspectos que foram úteis para o modelo proposto nessa dissertação. Assim, o projeto Oxygen é o projeto que possui a maior variedade de ferramentas e aplicações a diferentes níveis que dão suporte à criação de ambientes inteligentes. O projeto Aura propõe uma arquitetura e um modelo computacional entre os dispositivos e o ambiente. Também a arquitetura baseada no gerenciamento de serviços do projeto GatorTech é considerada nosso modelo. Na seção do projeto MoCA se faz uma descrição da infra-estrutura fornecida para a coleta de informação contextual dos dispositivos. Finalmente, apresentamos o projeto WSAmI que apresenta uma arquitetura baseada em serviços web e protocolos para a descoberta desses serviços web.

3.1 Definição de ambientes inteligentes

A definição mais conhecida de ambientes inteligentes é a apresentada e discutida em [18], onde Gagglioli define um ambiente inteligente como:

Um ambiente digital sensível e que responde à presença de pessoas[...] a visão de um ambiente inteligente se baseia em duas características: inteligência e inclusão. Inteligência pelo fato de o ambiente poder analisar o contexto e se adaptar às pessoas e objetos que se encontram no ambiente, aparentando inclusive emoções. A inclusão se refere a que pequenos dispositivos computacionais sejam parte das atividades diárias dos usuários sem que sejam percebidos...

A definição apresentada em [1] estabelece que um ambiente inteligente:

...permite a interação entre pessoas no mundo real e o mundo digital através de dispositivos que ativam a ubiqüidade da computação[...] a interface deste tipo de sistemas é o próprio ambiente, capturando informações do meio físico como movimentação, reconhecimento de som ou imagens, etc. a partir da ubiqüidade de dispositivos computacionais incluídos no ambiente.

Uma definição mais completa de ambientes inteligentes foi feita pela ISTAG (*Information Society Technologies Advisory Group*) [16, 22]:

O conceito de ambiente inteligente fornece uma visão ampla de como a sociedade da informação se desenvolverá. O foco de ambientes inteligentes

está em: amigabilidade com o usuário, serviços de suporte mais eficientes, melhoria das capacidades do usuário e suporte à interação humana[...] as pessoas se encontram rodeadas por interfaces intuitivas e inteligentes incluídas em todo tipo de objetos. Um ambiente inteligente será capaz de reconhecer e responder à presença de diferentes indivíduos[...] sendo que um ambiente inteligente trabalhe de maneira não obtrusiva, suave e de maneira invisível para o usuário.

As definições de ambientes inteligentes são muito genéricas com respeito às tecnologias e áreas envolvidas na sua implementação. Assim, em [1] e [37] é definida a colaboração básica das seguintes áreas de pesquisa: (1) computação ubíqua, (2) comunicação ubíqua e (3) interfaces inteligentes. Cada área tenta dar suporte às diferentes funcionalidades requeridas pelas definições de ambientes inteligentes.

Por exemplo, a definição que trata os ambientes inteligentes como uma interface natural entre pessoas e dispositivos computacionais requer o uso de tecnologias e ferramentas para o reconhecimento de voz e imagens. E a definição que exige uma resposta inteligente do ambiente requer que o ambiente interprete o comportamento das pessoas e cria a oportunidade para que métodos de planejamento em inteligência artificial sejam empregados.

Em [16, 22] são apresentadas as diretrizes que a implementação de ambientes inteligentes deve considerar:

- Permitir ativar a compatibilidade de software e hardware;
- Criação de plataformas abertas;
- Desenvolvimento de tecnologias intuitivas;
- Especificação compartilhada de meta-conteúdo de serviços;
- Tecnologias de segurança e confiabilidade;
- Suporte ao desenvolvimento de pessoas.

Os projetos mostrados a seguir desenvolveram ferramentas, aplicações e tecnologias para suporte à implementação de ambientes inteligentes, cobrindo as definições apresentadas sobre ambientes inteligentes.

3.2 Projeto Oxygen

Projeto do MIT iniciado em 2000. O projeto Oxygen [35] é um conjunto de projetos que dão suporte à implementação de ambientes inteligentes.

Estes sub-projetos fornecem diferen-tes funcionalidades, como interação homem-computador, comunicação entre os dispositivos e o ambiente, automatização do acesso a serviços e dados, aplicações para dispositivos especiais e ferramentas para a construção de aplicações cliente para ambientes inteligentes.

A seguir fazemos uma breve descrição de cada uma das tecnologias que fazem parte do Oxygen. Estas tecnologias são classificadas nos seguintes níveis: dispositivos, rede, software, percepção e usuários.

3.2.1 Tecnologias de dispositivos

Neste nível o Oxygen desenvolveu dispositivos especiais para interagir com o ambiente, relativos a computação, comunicação e percepção do ambiente. Esses dispositivos podem estar embutidos no ambiente ou serem transportados pelos usuários.

Os dispositivos embutidos no ambiente são compostos por um arranjo de câmeras e microfones que implementam a percepção natural dos usuários pelo ambiente mediante reconhecimento de voz e imagens. Estes dispositivos são chamados de *E21*. Os dispositivos transportados pelos usuários podem se autoconfigurar para permitir a comunicação com os elementos do ambiente e também permitem a interação com o ambiente. Estes dispositivos são chamados de *H21s*. Assim, o projeto desenvolveu um protótipo de *H21* chamado de *Handy-21* que é similar a um PDA com uma câmera e um detector de rádio-freqüência ativando o reconhecimento de voz e imagens para o usuário.

No referente a aplicações para o ambiente, o *Cricket* é um sistema que fornece informação sobre localização, orientação e espaços geográficos no ambiente. Este sistema se baseia em pontos de acesso de RF (rádio-freqüência) e emprega a potência do sinal que detectam os dispositivos para calcular a sua posição no ambiente. É necessário montar pontos de acesso RF no ambiente e por isso este sistema trabalha em ambientes fechados (*indoor*).

3.2.2 Tecnologias de rede

O Oxygen desenvolveu um conjunto de tecnologias para redes chamadas de *N21*. Estas tecnologias permitem montar redes flexíveis, descentralizadas e autoconfiguráveis entre dispositivos móveis e/ou fixos. Estas tecnologias de redes implementa algoritmos, middleware e protocolos que permitem:

- Montar redes automaticamente entre grupos de dispositivos que são facilmente adaptáveis à mobilidade e mudanças dos dispositivos. Grupos de dispositivos podem compartilhar a mesma informação. Os gru-

pos podem ser formados por proximidade dos dispositivos, podendo um dispositivo pertencer a vários grupos;

- Localização automática de recursos compartilhados na rede. A busca de recursos é feita a partir de intenções e funcionalidades requeridas (por exemplo, a impressora mais próxima do dispositivo). Assim, a rede encaminha transparentemente os dados para os recursos que satisfaçam essas intenções;

- Segurança, autenticação e confidencialidade entre as comunicações dos dispositivos. Esses aspectos são controlados por regras pré-estabelecidas na rede; por exemplo, antes de estabelecer a comunicação entre dois dispositivos são estabelecidos os privilégios dos recursos e informação que possui cada dispositivo. Assim, as redes só permitem manipular os itens que têm as permissões necessárias;

- Adaptabilidade às condições da rede como: tráfego, erros nas transmissões sem fio, latência, consumo de energia, tipo de informação transmitida e requisitos da aplicação. É possível mudar de protocolo de comunicação segundo os requisitos e condições da transmissão; por exemplo, no campus de uma universidade a transmissão entre dispositivos seria feita através de TCP/IP, mas numa sala poderia ser feita usando Bluetooth.

A seguir listamos os principais protocolos e aplicações implementadas pelo Oxygen seguindo as idéias descritas:

- *Grid* é um protocolo implementado para redes sem fio ad-hoc que não precisa de estações base. Todos os dispositivos ficam encarregados de repassar os pacotes.

- *Span* é um outro protocolo para redes ad-hoc sem fio que emprega a energia dos dispositivos eficientemente evitando que dispositivos fiquem esperando ou retransmitindo sinais, com isso causando desperdício de energia.

- *Chord* é um protocolo escalável para buscas distribuídas em redes ponto-a-ponto. Baseado neste protocolo foi criado o *Cooperative File System* (*CFS*), que oferece persistência a um grupo de agentes que trabalham coordenadamente.

- *Migrate* é um modelo para redes ponto-a-ponto que gerencia o estado de sessões de aplicações servidoras com clientes segundo mudanças que poderiam acontecer na rede.

3.2.3 Tecnologias de software

O Oxygen desenvolveu aplicações que têm como finalidade adaptar o ambiente aos usuários perante mudanças ou falhas. A adaptabilidade dever ser automática, sem a intervenção do usuário e sem interromper os processos ou serviços fornecidos. As principais mudanças consideradas são as que decorrem da mobilidade dos dispositivos e disponibilidade dos recursos. Assim, a adaptabilidade é conseguida através de protocolos de rede e bases de conhecimento dos usuários que armazenam o conhecimento que o ambiente possui dos usuários e dispositivos.

As tecnologias de software desenvolvidas pelo Oxygen permitem a construção de sistemas baseados em componentes distribuídos, os quais podem ser configurados com algum comportamento específico, e permitem a mobilidade de código e recuperação frente a quedas de conexão. As arquiteturas propostas pelo Oxygen se baseiam em três conceitos:

Abstração: Fornece um nível de abstração aos dispositivos que se encontram no ambiente. Um dispositivo fica visível para uma aplicação através de um componente ou um objeto de software.

Especificação: Permite descrever as abstrações empregadas. Uma especificação define a configuração do sistema, repositórios de módulos, dependências de módulos, capacidades dos módulos e comportamento dos módulos.

Persistência de objetos: Permite a manipulação de código e objetos num banco de dados orientado a objetos. A persistência tem suporte em todas as tecnologias do Oxygen.

Pebbles é uma plataforma de componentes implementando esses conceitos e permite compor componentes dinamicamente empregando técnicas de planejamento ou *model-checking*. Cada componente em *Pebbles* possui uma definição das suas interfaces definindo assinatura de métodos, documentação de código e informação adicional para os usuários.

O *MetaGlue* permite a comunicação entre vários agentes de software que residem nos dispositivos do ambiente. Separa o software para os cliente do software necessário para o controle de dispositivos no ambiente. O *MetaGlue*

permite a descoberta de serviços, comunicação a longa escala e permite ao usuário tratar com recursos do ambiente.

SUDS (*Software Upgrades in Distributed Systems*) permite atualizar automaticamente código dos objetos de um banco de dados orientado a objetos. As atualizações são feitas transparentemente sem afetar os processos que poderiam estar em execução.

IOA é uma linguagem para expressar projetos de software em diferentes níveis de abstração, desde a definição do comportamento até uma especificação capaz de gerar código. Com esta linguagem é possível simular e empregar raciocínio automático sobre os modelos projetados.

3.2.4 Tecnologias de percepção

O Oxygen ativa a interação entre o ambiente e o usuário através de dois meios: reconhecimento de voz e imagens.

O reconhecimento de voz permite criar sentenças para aplicações a partir de comandos de voz do usuário. Permite também sintetizar voz a partir dos resultados de uma aplicação. Qualquer componente do Oxygen pode empregar o reconhecimento de voz através de interfaces bem definidas.

O mecanismo de reconhecimento de voz define quatro processos: (1) reconhecimento de voz, que converte a voz do usuário em palavras; (2) entendimento da linguagem, interpreta o conjunto de palavras como linhas de comandos para as aplicações; (3) geração de linguagem, interpreta os resultados das aplicações como sentenças em linguagem natural e (4) síntese da voz, a partir das sentenças geradas. O entendimento da linguagem pode ser adaptado pelo uso de palavras chaves no domínio do usuário. Cada um dos quatro processos para o reconhecimento de voz foram implementados nos sistemas *SUMMIT*, *TINA*, *Dialog Manager*, e *GENESIS* respectivamente.

Galaxy é uma implementação do modelo proposto para o reconhecimento de voz que permite a criação de aplicações dotadas com reconhecimento de voz. Além disso, o *Galaxy* se encarrega do controle do fluxo de dados entre o cliente e o "servidor de voz".

Orion é um agente, já mais elaborado, que executa comandos de voz do usuário de maneira assíncrona. Ao completar uma tarefa, ele contacta o usuário para lhe mostrar os resultados depois de um tempo especificado pelo próprio usuário.

SpeechBuilder permite criar aplicações que empregam reconhecimento de voz a partir da semântica de conceitos e atributos definidos pelo desenvolvedor da aplicação.

3.2.5 Tecnologias de usuários

As tecnologias de usuários definidos pelo Oxygen foram divididas em três áreas: (1) acesso ao conhecimento, (2) automatização e (3) colaboração.

Acesso ao conhecimento

O Oxygen propõe a ubiqüidade dos dados (acesso a dados de qualquer lugar e a qualquer momento) numa base de conhecimento, onde uma base de conhecimento é mantida por cada usuário. Nesse cenário é possível manter um repositório de conhecimento compartilhado entre vários usuários. Para acessar esse conhecimento uma aplicação cliente pode fazer consultas diretas ao repositório ou procurar por essa informação na web.

São propostas as seguintes tarefas na manutenção de uma base de conhecimentos: (1) Representação dos dados, armazenados como texto, imagens, voz, etc.; (2) Aquisição de dados, em que é armazenada informação que o usuário introduz no sistema; (3) Métodos de acesso automático: quando uma nova informação é adicionada a uma base de conhecimentos, serviços especificados podem ser executados automaticamente; (4) Observadores automáticos, encarregados de ajustar automaticamente a informação de uma base de conhecimento visando os interesses do usuário.

Haystack é uma plataforma para criar, visualizar e organizar informação pessoal do usuário que emprega RDF como linguagem para a modelagem da informação. Com o propósito de manipular a informação automaticamente, o Oxygen emprega a web semântica. Com este propósito desenvolveram o *START* que converte consultas em linguagem natural para consultas a bases de conhecimento, podendo acessar texto, imagens, vídeo, som, etc.

Automatização

O Oxygen propõe a automatização das transações entre nós da rede para permitir controlar o ambiente físico do usuário. Agentes executam *scripts* determinados pelos usuários, acessando bases de conhecimento na procura de informação adicional para poder executar apropriadamente uma ação. Os scripts determinam as preferências introduzidas por os próprios usuários. Os agentes também podem adaptar o ambiente às preferências do usuário, quando o usuário entra em um novo ambiente.

Para conseguir interação entre objetos físicos e objetos virtuais (agentes), é utilizada uma interface comum a todos os objetos físicos e virtuais. A interface é um componente de hardware que permite converter informação física em informação de software. Através da interface é possível estabelecer

a interação entre diferentes objetos através de scripts; isto permite compor componentes a partir de objetos como reconhecimento de voz, imagens, etc.

GOALS é um sistema que guia a construção desse tipo de componentes. Dado um objeto inicial (chamado de contexto) e um conjunto de metas, o *GOALS* faz uma busca heurística por objetos que podem satisfazer as metas quando são integrados ao objeto inicial.

Colaboração

A visão de colaboração do Oxygen se baseia em que os computadores suportem as interações entre pessoas no ambiente. Assim, o ambiente pode ajudar a notificar aos usuário sobre tarefas que deve realizar em colaboração com outras pessoas, se tornando num assistente pessoal do usuário. Isto é feito através de uma análise do contexto das interações entre os usuários. Esta abordagem faz necessário acessar as bases de conhecimento dos usuários e efetuar ações no ambiente através da automação antes vista. A detecção do contexto de um usuário pode ajudar a determinar a tarefa que está executando e é possível detectar outros usuários no ambiente que poderiam colaborar com essa tarefa.

A detecção do contexto pode ser feita a partir das informações processadas por módulos de reconhecimento de imagens, voz, etc. Assim, o ambiente pode detectar que pessoas estão numa reunião e pode efetuar ações para cada participante na reunião –por exemplo, disponibilizar arquivos que um usuário deseje compartilhar.

MeetingView é uma ferramenta que implementa estas idéias que permite acompanhar uma reunião e apresentar um resumo do conteúdo tratado. *Annotea* é outra ferramenta que permite compartilhar pela web anotações de usuários feitas em RDF. *ASSIST* é um assistente para a preparação de artigos, mantendo versões de rascunhos feitos pelos usuários. A informação pode ser obtida a partir de texto, voz ou imagens capturadas pelo ambiente.

3.3 Projeto Aura

Projeto da universidade Carnegie Mellon iniciado em 2002. A arquitetura proposta pelo projeto Aura [45] se baseia no modelo do *computação dirigida por tarefas* [52] (modelo computacional usado também no presente projeto). Em resumo, a *computação dirigida por tarefas* se baseia em manter ativas as tarefas dos usuários considerando a disponibilidade dos recursos e a mobilidade dos dispositivos. O conceito de *computação dirigida por tarefas* foi apresentado com maior detalhe na Seção 2.4 desta dissertação. Nesta seção

apresentamos a arquitetura proposta pelo Aura para dar suporte a esse tipo de computação.

A arquitetura está baseada em tarefas que são especificadas pelos usuários. Uma tarefa pode ser uma atividade como: "editar ou escrever um arquivo de texto", "fazer uma apresentação", "procurar uma lista de livros", etc. A mobilidade dos dispositivos, disponibilidade dos recursos e tarefas de longa duração exigem que a arquitetura permita lidar com mudanças nas tarefas, no contexto e no ambiente físico, permitindo que o usuário não perceba essas mudanças e não tenha que reconfigurar seu contexto (computacional e físico) para continuar com a tarefa que vem desenvolvendo.

Assim, a arquitetura do Aura tenta maximizar a disponibilidade dos recursos e minimizar a atenção do usuário na configuração dos serviços empregados na sua tarefa. Algumas definições empregadas na arquitetura são:

- *Tarefa:* uma tarefa é definida como um conjunto de serviços estruturados. Um serviço é instanciado com um conjunto de recursos no ambiente;
- *Ambiente:* conjunto de dispositivos e aplicações acessíveis ao usuário numa determinada localização;
- *Recursos:* podem ser dispositivos e/ou aplicações no ambiente.

Para que o ambiente consiga ajudar o usuário numa tarefa ele deve encontrar os recursos necessários para instanciar os serviços que definem a tarefa. Para isto, é feita uma descrição abstrata dos serviços. Quando o ambiente ou o contexto do usuário mudam, os serviços devem continuar instanciados com os novos recursos correntes no ambiente.

A arquitetura define a *aura do usuário* como a entidade principal que representa as tarefas de um usuário. A *aura* é encarregada de: (1) capturar o contexto físico que afeta a tarefa do usuário e (2) organizar os recursos apropriados para uma determinada tarefa. Para este fim a arquitetura define a necessidade de alguns mecanismos que permitam: (1) especificar quais novas características são requeridas e (2) a necessidade de entidades (*placeholders*) encarregadas de capturar a natureza da tarefa e as preferências dos usuários.

Na Figura 3.1 apresenta-se uma visão geral da arquitetura proposta:

3.3.1 Gerenciador da tarefa (Prism)

Representa a *aura do usuário* descrita anteriormente. O principal objetivo deste componente é permitir ao usuário concentrar-se na tarefa que está

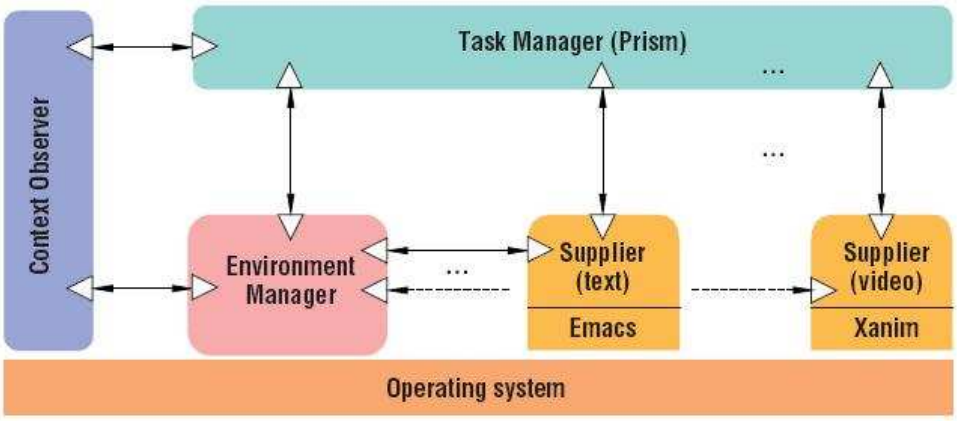

Figura 3.1: Arquitetura do Aura[45].

desenvolvendo. Isto é conseguido lidando com quatro tipos de mudanças:

- O usuário muda de ambiente: O gerenciador de tarefa permite a migração de toda a informação relacionada à tarefa do usuário ao novo ambiente. O Prism também deve negociar os recursos no novo ambiente para poder continuar sua tarefa.

- O ambiente muda: O Prism monitora o ambiente através dos fornecedores de serviços e, frente a mudanças no ambiente, solicita ao gerenciador do ambiente novos recursos para poder continuar a tarefa do usuário.

- Mudança na tarefa: Quando o usuário deseja encerrar uma tarefa para iniciar uma nova, o Prism armazena num repositório o *acompanhamento* de serviços e recursos usados naquela tarefa, podendo reutilizar informação e serviços da tarefa antiga.

- Mudança no contexto: A descrição dos serviços possui restrições dos serviços no ambiente. Assim, quando as restrições não foram satisfeitas o Prism: (1) coordena a suspensão da tarefa, ou (2) ajusta o contexto do usuário a essas restrições.

O Prism é um ambiente de execução de tarefas independente da plataforma. Toda a arquitetura definida pelo Aura permite migrar o Prism a ambientes que possuam uma instância da arquitetura. Quando o usuário muda de ambiente, toda a informação e estado do Prism é transportada ao novo

ambiente para continuar a execução da tarefa. Neste novo ambiente todos os componentes da arquitetura trabalham para continuar com a execução da tarefa com a menor atenção do usuário e com a reutilização de recursos.

A seguir fazemos uma breve descrição dos principais componentes definidos nesta arquitetura.

3.3.2 Fornecedores de serviços

São os componentes que representam serviços abstratos. Estes componentes são encarregados de abstrair aplicações reais como serviços, fornecendo uma interface comum a todas as aplicações e expondo uma descrição do serviço.

Estes componentes são *wrappers* com a descrição abstrata do serviço e a configuração da aplicação. Na prática, estes componentes fazem a ligação entre a API de uma aplicação com a API do ambiente (do Aura).

A descrição dos serviços é feita usando uma linguagem de marcação baseada em XML. Cada serviço pode empregar marcações particulares com significado próprio, mas as marcações e significado devem ser compartilhados com outros fornecedores de serviços.

3.3.3 Observador de contexto

Este componente se encarrega de monitorar o contexto do dispositivo e notificar mudanças ao Prism e ao gerenciador do ambiente. A informação recolhida vem de sensores localizados no ambiente, com diferentes graus de confiabilidade e granularidade.

3.3.4 Gerenciador do ambiente

Este componente é o encarregado de detectar os fornecedores de serviços disponíveis num ambiente. Quando novos serviços são incorporados ao ambiente, são registrados no gerenciador do ambiente junto com as sua descrição abstrata do serviço. Quando é necessário instanciar uma tarefa com serviços o gerenciador do ambiente é consultado quanto a serviços abstratos disponíveis segundo as necessidades da tarefa e as restrições do ambiente ou usuário.

Assim, o gerenciador do ambiente é o encarregado da descoberta de serviços, selecionando os melhores serviços que cubram as necessidades dos usuários considerando a especificação e restrições do serviço abstrato.

A implementação desta arquitetura está baseada em duas arquiteturas adaptativas implementadas isoladamente: *Coda* e *Odyssey*. Estas arquiteturas foram implementadas antes da criação do Aura, e posteriormente modificadas para se alinhar às necessidades da arquitetura do Aura. O *Coda* é um sistema de arquivos distribuído transparente para os usuários que permite o acesso a arquivos num cenário de computação móvel. O *Odyssey* dá suporte à criação de aplicações capazes de se adaptar a mudanças de contexto.

3.4 Projeto Gator Tech: Smarthouse

Projeto da universidade da Florida. Este projeto propõe uma arquitetura de serviços baseada em OSGi (*Open Services Gateway initiative*) [1] onde todo atuador, sensor ou dispositivo no ambiente é controlado como um serviço OSGi e implantado como um módulo (*bundle*) no ambiente.

O OSGi quando usado como arcabouço para a arquitetura, permite a criação de um ambiente onde diferentes plataformas e tipos de dispositivos podem ser incorporados facilmente. A única exigência é que as aplicações sejam implementadas em Java. A visão deste projeto é de um ambiente inteligente como um sistema modular distribuído cujos dispositivos são serviços facilmente "plugáveis" no ambiente.

São definidas seis camadas para abstrair as funcionalidades da arquitetura proposta. As camadas vão do nível físico até o nível de aplicação, permitindo ter a representação de uma entidade no mundo físico disponível como um componente de software (serviço OSGi) na camada de aplicação. A seguir descrevemos cada uma destas camadas e seus componentes, conforme apresentado na Figura 3.2.

Como parte deste projeto, foi construída uma casa inteligente (*Smarthouse*) capaz de monitorar a saúde de pessoas com necessidades específicas.

3.4.1 Camada física

Esta camada representa objetos físicos contidos no ambiente, que são classificados como sensores ou atuadores. Nesta camada o projeto Gator Tech implementou a casa com vários tipos de sensores e atuadores em diferentes objetos e partes da casa. Por exemplo, foi colocado um leitor RFID na porta da casa para detectar as pessoas que estão na casa, e sensores na cama

[1] Sucintamente, OSGi é uma especificação para o desenvolvimento de componente dinâmicos que permite desenvolver aplicações baseadas em componentes, permitindo a implantação dos módulos de maneira distribuída, remota, confiável e em diversas plataformas.

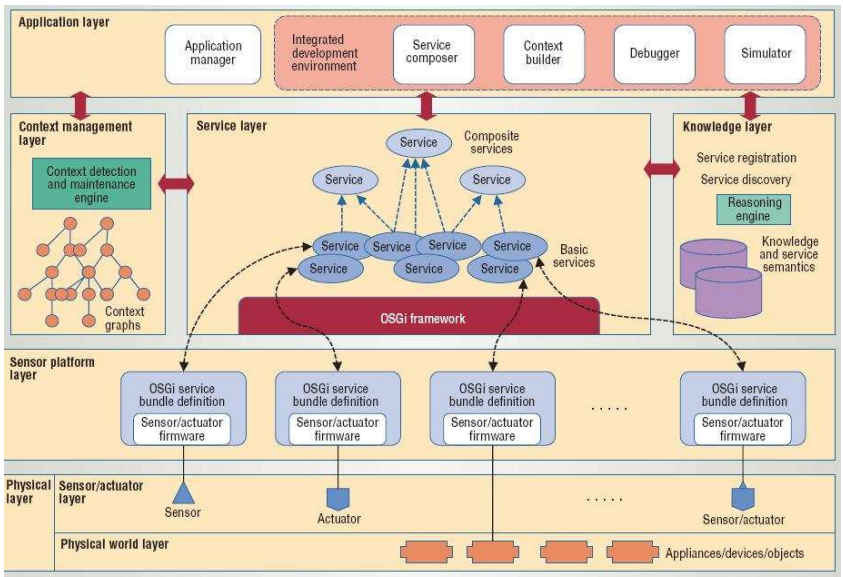

Figura 3.2: Arquitetura do Gator Tech[19].

para detectar padrões de sono, vários sensores no banheiro, na lavanderia, na sala, na geladeira. no forno de microondas e em outros eletrodomésticos para detectar atividades básicas das pessoas no ambiente.

O componente físico mais importante deste projeto é o *SmartFloor*. O *SmartFloor* são sensores colocados no chão da casa, que funcionam como um sistema de localização de pessoas. Assim, o sistema pode detectar a localização das pessoas dentro da casa.

3.4.2 Camada de sensores

Esta camada serve como mediador entre a camada física e a camada de serviços. Nesta camada é feita a abstração dos sensores e atuadores que se encontram no mundo físico como componentes de software, mais especificamente como serviços. Isto permite que os programadores possam embutir estes dispositivos em aplicações sem conhecer o funcionamento deles.

3.4.3 Camada de serviços

Esta camada possui um repositório de serviços ativos no ambiente. Um *serviço básico* representa um dispositivo físico acessado através da camada de

sensores. Nesta camada pode-se criar *serviços compostos* a partir de serviços básicos.

A implementação de serviços OSGi para o acesso aos dispositivos no ambiente permite que o ambiente possa gerenciar o ciclo de vida de cada dispositivo. Assim, por exemplo, quando um novo dispositivo é adicionado ao ambiente, ele só precisa ser registrado no repositório de serviços para se tornar disponível no ambiente.

No vocabulário do OSGi, um componente na camada de sensores representaria um *bundle* OSGi, ou seja um componente implementado com a API proprietária do dispositivo que permite controlá-lo. Já um serviço OSGi representaria uma implementação que permite controlar o ciclo de vida e funcionamento desse componente, e que é disponibilizado para aplicações em camadas superiores através de uma interface definida e conhecida.

Nesta camada poderiam ficar disponíveis serviços padronizados para o ambiente, como por exemplo um serviço de reconhecimento de voz que poderia ser usado por qualquer aplicação. Isto permite uma integração fácil entre os serviços.

3.4.4 Camada de conhecimento

Esta camada contém uma ontologia dos serviços oferecidos e dos dispositivos ativos no ambiente. Esta ontologia permite empregar métodos automáticos de raciocínio para a descoberta e composição de serviços.

Os mecanismos de raciocínio empregados para esta tarefa ainda não tinham sido definidos nesta camada.

3.4.5 Camada de gerenciamento do contexto

Esta camada permite manter um registro dos contextos de interesse para as aplicações, permitindo a ativação de serviços e a definição dos estados que não são permitidos no ambiente. Nesta camada, um *gerenciador de contexto* monitora atualizações e mudanças no contexto. Um contexto é também implementado como um serviço OSGi no arcabouço.

3.4.6 Camada de aplicação

Nesta camada se apresentam ferramentas gráficas que permitem associar serviços (comportamentos) a contextos. O *construtor de contexto* se encarrega de ativar e executar os serviços segundo as associações feitas aos contextos. A declaração de comportamentos também implica em definir os

contextos que não são permitidos no ambiente, bem como associar serviços de recuperação aos contextos não permitidos.

Nesta camada, os desenvolvedores podem empregar o *compositor de serviços* para a descoberta e composição de serviços. Isto é, podem criar novos serviços compondo serviços básicos.

O projeto *Gator Tech* estabelece que a implementação de um ambiente inteligente envolve três atividades: (1) engenharia de contexto, (2) engenharia de software e (3) associação de comportamento a contextos. Por exemplo, para regular a temperatura de um aquecedor deveríamos associar esse comportamento ao um certo contexto que descreve o estado de um termômetro no ambiente.

Todo atuador no ambiente possui um *efeito intencional* que é descrito na especifição do serviço que representa esse atuador. A partir destes efeitos, é possível determinar quais contextos são permitidos. Mudanças no contexto podem ocorrer devido a atuadores ou eventos físicos no ambiente.

3.5 Projeto MoCA

Projeto da universidade PUC-RIO iniciado em 2004. MoCA (Mobile Collaboration Architecture)[40] é um middleware que oferece uma arquitetura para a implementação de aplicações sensíveis ao contexto para colaboração entre dispositivos móveis.

O MoCA permite a coleta, agregação e acesso a diferentes tipos de informação do contexto de um usuário ou de uma comunidade de usuários. A informação contextual pode ser compartilhada entre várias aplicações. Esta arquitetura permite processar esta informação considerando: (1) a mobilidade dos usuários e (2) as aplicações clientes devem poder acessar esta informação em qualquer momento.

É fornecida uma API para a implementação de clientes, servidores e aplicações proxy (*ProxyFramework*), permitindo o acesso de forma transparente aos serviços que a arquitetura oferece para a manipulação de contexto. É definido um arcabouço para aplicações proxy que permite a implementação de adaptabilidade disparada por eventos associados a mudanças no contexto.

Qualquer aplicação na arquitetura possui três tipos de elementos: clientes, servidores e proxies. Os proxies se encarregam de intermediar todas as comunicações entre o cliente e o servidor fornecendo funcionalidades como encriptação, compressão, segurança, etc.

A arquitetura do MoCA apresentada na figura 3.3 fornece os seguintes serviços de middleware.

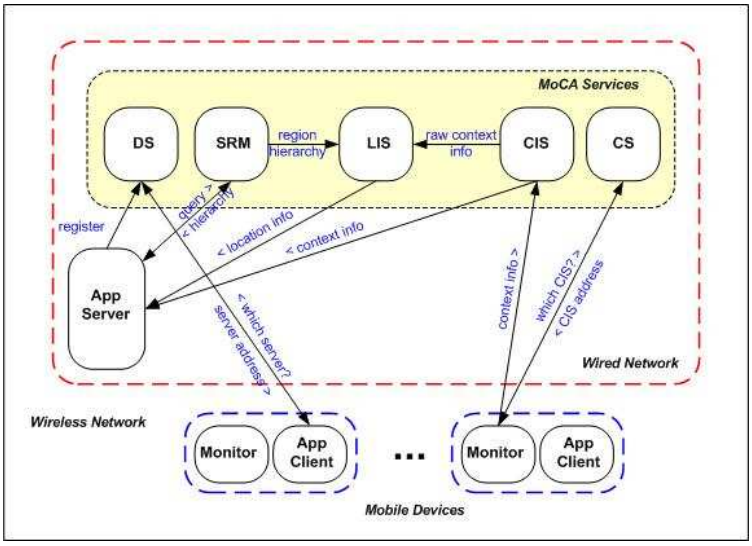

Figura 3.3: Arquitetura do MoCA[40].

3.5.1 Monitor

É um *daemon* que é executado em cada dispositivo móvel e é encarregado de coletar e enviar informação contextual do dispositivo ao serviço de informação de contexto (CIS) que pode ser executado em um ou vários nós fixos da rede.

O Monitor coleta o seguinte tipo de informação contextual:

- Uso de CPU;
- Qualidade da conexão;
- Quantidade de energia remanescente no dispositivo;
- Memória livre no dispositivo;
- Potência do sinal dos pontos de acesso visíveis pelo dispositivo e pelo ponto de acesso corrente.

O Monitor foi implementado para Windows XP em nós fixos e Windows CE 5.0 para Pocket PC.

3.5.2 CIS - Context Information Service

Este serviço armazena, coleta e processa toda a informação de contexto enviada pelos monitores nos dispositivos. O CIS é um serviço distribuído que é executado num nó fixo da rede. O CIS pode ser acessado de qualquer outro componente da arquitetura. Este serviço é encarregado de procurar informação contextual de um dispositivo entre em outros CIS que poderiam se encontrar na rede.

O serviço associa a um identificador (tipicamente o endereço MAC do dispositivo) toda a informação contextual recebida. Assim, aplicações clientes podem consultar tais informações passando esse identificador.

Aplicações clientes podem registrar seu interesse por certo tipo de informação contextual através de CIS *subscrições*. Assim, quando mudanças no contexto são detectadas pelo CIS ele envia *notificações* às aplicações registradas.

O MoCA definiu uma linguagem de regras para expressar o interesse por informação de contexto. Estas regras são chamadas de *regras de interesse*, e têm a forma: `{(CPU > 90) OR (Roaming = True)}`,`{(FreeMem < 10KB) AND (APChange = True)}`. A primeira regra indica o interesse por dispositivos cujo porcentagem de uso de CPU seja maior que 90% ou a capacidade de *roaming* do dispositivo esteja ativa. Já a segunda regra estabelece o interesse por dispositivos com memória livre menor que 10KB que tenham mudado de ponto de acesso.

3.5.3 DS - Discovery Service

Armazena informação das aplicações e serviços registrados no middleware. Estas informações podem ser nomes, propriedades e endereços, dentre outras.

3.5.4 CS - Configuration Service

Este serviço armazena e gerencia informação referente às configurações dos dispositivos móveis no ambiente. Desta maneira, os dispositivos podem acessar serviços do middleware como o CIS e DS. Podem existir vários serviços CIS e DS numa rede, assim o CS armazena os serviços onde o Monitor do dispositivo estiver enviando a sua informação contextual (IP e porta do serviço); também é armazenada a freqüência com que o monitor envia nova informação contextual.

A informação é indexada segundo o endereço MAC dos dispositivos, o

que permite identificar de maneira única a informação distribuída dos dispositivos dentro da rede.

3.5.5 LIS - Location Information Service

Este serviço fornece a localização lógica de um dispositivo comparando a potência das sinais de pontos de acesso audíveis do dispositivo com padrões de sinais associados previamente a pontos de referência na região.

O LIS usa a informação coletada pelo CIS para achar a localização *aproximada* do dispositivo com base na média das *distâncias Euclidianas* do dispositivo aos pontos de referência. É por isso que a localização de um dispositivo depende do número de pontos de acesso e de pontos de referência empregados.

3.5.6 SRM - Symbolic Region Manager

O SRM permite gerenciar áreas simbólicas, estruturadas em sub-áreas lógicas.

3.6 Projeto WSAmI

Projeto do INRIA da França iniciado em 2002. O objetivo do projeto WSAmI (*Web Services in Ambient Intelligence*) [21], é empregar a ubiqüidade da Internet para dar suporte a ambientes inteligentes cujas entidades possam interagir empregando serviços web. A-ssim, no projeto foi desenvolvido um middleware distribuído que dá suporte a serviços web num contexto de computação móvel, definindo serviços de publicação e busca de serviços web, permitindo que serviços web possam ser executados em dispositivos móveis e fixos.

A abordagem está baseada puramente em serviços web e emprega tecnologias atualmente usadas para a descrição de serviços web como: WSDL (*Web Services Description Language*) para a descrição das interfaces dos serviços web; e WSCL (*Web Services Conversation Language*) para a descrição do comportamento dos serviços web. Essas linguagens são empregadas atualmente para a geração automática de código mas, como será apresentado no seguinte capítulo, não permitem a descoberta, execução, composição e monitoração automática de serviços web [4]. *O WSAmI permite ativar a busca de instâncias de serviços web no contexto da computação móvel.* Essas instâncias representam entidades de software executáveis que são fornecidas por dispositivos móveis e/ou fixos no ambiente.

```
<definitions name="urn:LibraryService" ... >
   <types>
    ...
   </types>
   <message name="isbnQuery">
      <part name="isbn" type="xsd:string"/>
   </message>
   ...
   <portType name="LibraryInterface">
     <operation name="booksForPosition">
        <input message="tns:userPosition"/>
        <output message="tns:booklistResponse"/>
     </operation>
   ...
   </portType>
   <binding name="LibrarySOAPBinding" type="tns:LibraryInterface">
     <soap:binding style="rpc" transport="..."/>
     <operation name="booksForPosition">
         ...
     </operation>
      ...
   </binding>
</definitions>
```

Figura 3.4: Esqueleto da interface abstracta WSAmI.

O WSAmI define duas interfaces para a descrição de serviços web: (1) a interface abstrata baseada em WSDL e WSCL e (2) a interface concreta baseada puramente em WSDL. A especificação da interface abstrata é feita sobre instâncias de serviços. Uma instância de serviço é um ponto de execução (*end point*) de um serviço web onde são enviadas chamadas remotas. A interface abstrata contém a interface WSDL sem o elemento `service`[2] e a interface concreta contém o ponto de execução que oferece uma implementação para essa interface WSDL.

Cada interface abstrata do serviço web descreve instâncias concretas de serviços web, que são retornadas ao cliente que solicitou um serviço web. A-

[2]O elemento `service` numa interface completa WSDL contém o ponto de execução (*end point*), que é uma *URL* do serviço web para onde serão enviadas os envelopes *SOAP*.

ssim, quando um serviço é implantado o WSAmI consegue montar interfaces completas WSDL.

Na Figura 3.4 é apresentado o esqueleto da interface abstrata do web service `Library`. A descrição da interface WSDL é feita da maneira convencional, exceto que não é definido o ponto de execução do serviço web. Esse elemento é descrito na interface concreta do serviço web que importa a interface abstrata, conforme mostrado na Figura 3.5.

```
<definitions name="urn:LibraryService"
             targetNamespace="urn:LibraryService"
             xmlns:tns="urn:LibraryService"
             xmlns:typens="urn:LibraryService"

   <import namespace="urn:LibraryService"
           location="LibraryInterface.wsdl" />

   <service name="LibraryWebService">
      <port name="Library" binding="tns:LibrarySOAPBinding">
         <soap:address location="http://localhost:8080/
                                services/Library"/>
      </port>
   </service>
</definitions>
```

Figura 3.5: Esqueleto da interface concreta WSAmI.

A união destas interface é feita em um arquivo de configuração do serviço web implantando junto com o serviço web no WSAmI. A Figura 3.6 mostra este arquivo. Um outro arquivo de configuração associa um arquivo WSCL com o comportamento do serviço web a sua interface abstrata.

O WSAmI contempla a execução de serviços web em computadores fixos e em dispositivos móveis com capacidade suficiente para a execução de um servidor web. Para o primeiro caso o WSAmI emprega o servidor *Tomcat* e no caso dos dispositivos móveis o servidor web *Jetty*. No caso de computadores fixos, esse middleware emprega *AXIS* como motor *SOAP* e para os dispositivos móveis foi desenvolvido o motor *SOAP CSOAP* (tanto para o lado do cliente como para o lado do servidor), compatível com a especifi-

```
<?xml version="1.0" encoding="UTF-8"?>
<wsami xmlns="http://www-rocq.inria.fr/arles/wsami"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

   <Definition name="Library"
               targetNamespace="http://localhost:8080/wsami/">
      <Service name="Library">
         <Abstract hrefSchema="http://localhost:8080/
                               wsami/LibraryAbstract.wsami"/>
         <Concrete hrefSchema="http://localhost:8080/
                               wsami/LibraryConcrete.wsdl"/>
      </Service>
   </Definition>
</wsami>
```

Figura 3.6: União das interfaces abstrata e concreta do serviço web.

cação *Jax-RPC*[3]. Este motor *SOAP* é mais leve, o que permite que ele seja utilizado em dispositivos con menores recursos computacionais.

3.6.1 Módulos e serviços do middleware

Na Figura 3.7 se apresentam os módulos que compõem o middleware do WSAmI. A idéia é permitir que outras aplicações sejam construídas sobre o middleware, facilitando para essas aplicações a busca, registro e execução dos serviços web no ambiente.

A seguir fazemos uma descrição dos módulos mais importantes dessa arquitetura:

Core Broker: oferece o mediador central do middleware que gerencia os serviços web disponíveis no dispositivo. Este módulo é o encarregado de controlar o ciclo de vida dos serviços web e a recepção e processamento das mensagens *SOAP*. Este modulo depende do tipo do dispositivo para escolher o motor *SOAP* a ser executado (Axis ou CSOAP).

[3] *Java API for XML-Based RPC*. Atualemente é a especificação base para a troca e processamento de mensagem *SOAP* para serviços web em Java (`http://java.sun.com/webservices/jaxrpc/`).

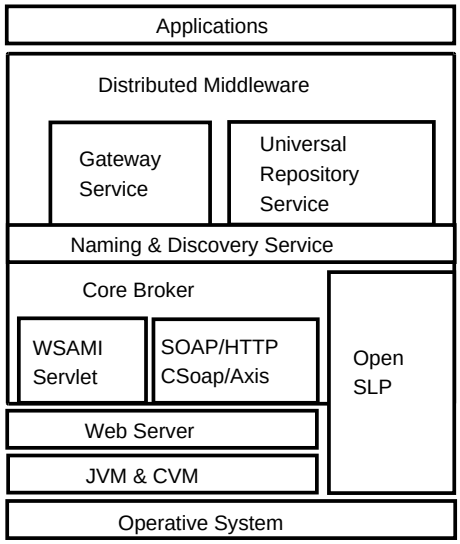

Figura 3.7: Arquitetura do WSAmI [21].

Serviço de registro e descoberta: ND - Naming & Discovery Service. Permite a recuperação de uma instância de serviços web na rede local ou ampla, de acordo com: (1) um nome ou uma especificação declarativa do serviço web e (2) o ambiente em que o serviço web é requisitado mediante as propriedades não funcionais associadas aos serviços.

Serviço de Repositório Universal: UR - Universal Repository Service. É um serviço para o registro e descoberta dinâmica de serviço web. Este módulo é uma implementação UDDI que mantem um registro global dos serviços web no ambiente.

Serviço Gateway: permite a um dispositivo se conectar desde uma rede sem fio a uma rede fixa. Isto é útil quando um dispositivo se encontra no modo de rede *ad-hoc* e precisa executar um serviço web que se encontra fora da rede ad-hoc em que ele se encontra.

Os módulos ND e UR são fornecidos como serviços web em cada dispositivo que executa o middleware, para poderem ser acessados por qualquer outro dispositivo. Assim, esses módulos são chamados de *serviço ND* e *serviço UR* respectivamente.

3.6.2 Descoberta de serviços web

O WSAmI faz a descoberta das *instâncias de serviços web* locais e remotos baseado somente no URI da interface abstrata que descreve o serviço web. O *casamento* é puramente sintático entre o URI dos documentos, ou seja, o cliente envia um URI e o WSAmI retorna uma instância do serviço web cuja interface abstrata possui o mesmo URI. O WSAmI emprega também características não funcionais dos serviços web (propriedades como qualidade de serviços, largura de banda, segurança, etc) associadas às interfaces abstratas para serem avaliadas no matching.

O WSAmI emprega o protocolo SLP (*Services Location Protocol*) para registrar seu próprio serviço de busca (serviço ND), que se baseia em registros de serviços web disponíveis. Assim, o WSAmI faz o registro de serviços web em dois repositórios: (1) repositório local, que mantém os serviços web que o próprio dispositivo disponibiliza; (2) repositório remoto, que atua como um cache de web services remotos requisitados anteriormente pelos clientes. O cache é mantido com uma política do menos recentemente usado (LRU).

Cada entrada do repositório remoto contém: (1) o endereço URI da interface abstrata do serviço web, (2) uma lista de instâncias de serviços web conhecidos na Internet (rede ampla) e (3) uma lista de instâncias descobertas na rede local. Por sua vez cada elemento dessas listas contém: (1) o endereço URL do serviço web, (2) um apontador para a instância corrente, (3) informação sobre interceptadores empregados e (4) um marcador que indica se o nó (hospedeiro) é conectado a fonte ou não.

Cada registro no repositório local contém: (1) o URI do documento que define a interface abstrata do serviço web, (2) o URI da interface concreta do serviço web definida em WSDL, (3) o URI do documento que define o serviço web (a união das interfaces abstrata e concreta) e (4) a lista de serviços web requeridos para a execução do serviços web.

A seguir descrevemos o algoritmo de descoberta de serviços web do WSAmI.

Serviço ND: Uma instância do serviço ND é iniciado em todo dispositivo com o middleware do WSAmI sendo executado. Cada serviço ND contém um repositório dos serviços web locais e remotos que conhece, conforme especificado anteriormente. Além disso, em cada dispositivo se inicia o serviço SLP em modo de agente de serviço[4], onde é registrado

[4] O protocolo SLP define três tipos de entidades que interagem no protocolo: (1) o agente de diretório, (2) o agente de serviço e (3) o agente de usuário. O agente de serviço fornece a localização de um ou vários serviços.

o próprio serviço ND. O registro no serviço SLP é com uma entrada do tipo `service:ND:http://128.93.8.102:8080/services/ND`.

Periodicamente é feita a recuperação de serviços ND disponíveis no ambiente através do protocolo SLP. O WSAmI registra junto com o serviço ND um atributo que indica se o dispositivo está conectado à fonte ou não (*power plugged*). Na recuperação dos serviços ND está informação também é recuperada. Neste processo de recuperação todos os dispositivos com o serviço ND rodando atualizam um repositório com os serviços ND remotos acessíveis e se o dispositivo que o contém está conectado à fonte ou não.

Troca de IP: Os dispositivos podem mudar de endereço IP (por trocar de rede ou por mudar o modo de comunicação –infra-estrutura ou ad-hoc). Quando o serviço ND é iniciado no dispositivo é iniciado um processo que consulta por mudanças no IP do dispositivo. Se fosse detectada alguma mudança, o serviço SLP sendo executado no dispositivo é reiniciado com o novo IP e é registrado nele o serviço ND do próprio dispositivo.

É necessário manter a consistência na localização de serviços e a acessibilidade dos serviços do dispositivo. Os serviços do dispositivo que se encontram no repositório UR da rede são atualizados com o novo IP. Isto permite atualizar ao algoritmo de descoberta atualizar referências a serviços do dispositivo.

Algoritmo de descoberta: O WSAmI procura as instâncias de um serviço web a partir da URI da interface abstrata que o cliente solicitou. Isto implica que o cliente que procura um serviço web deve conhecer a interface abstrata do serviço web que pretende acessar. O WSAmI faz a busca por instâncias de serviços web que possuam essa interface abstrata e que se encontrem disponíveis nos dispositivos do ambiente. A busca dessas instâncias é baseada em três repositórios: (1) repositório local, que mantém instâncias de serviços web locais; (2) repositório remoto, que contem instâncias de serviços web remotos, que foram descobertos anteriormente; e (3) repositório de serviços ND, contém os serviços ND disponíveis em outros dispositivos.

A busca de instâncias de serviços web é iniciada chamando os métodos `ND.getService` ou `ND.getService_ByInstanceURL` do serviço ND, passando como parâmetro a URI da interface abstrata do serviço web ou a URL da instância de um serviço. Na Figura 3.8 são mostrados os passos do algoritmo de descoberta. É necessário ressaltar que os

passos 1-4 são efetuados primeiro para dispositivos conectados à fonte (tipicamente dispositivos fixos) e se não for encontrada nenhuma instância, os mesmos passos são repetidos para os dispositivos não conectados à fonte (tipicamente dispositivos móveis). A informação sobre o tipo de dispositivo é coletada junto com a recuperação de serviços ND, conforme explicado anteriormente. Caso que não fosse encontrada nenhuma instância no repositório remoto do dispositivo a requisição é re-enviada aos serviços ND visíveis para o dispositivo, achados pelo serviço SLP (passo 3). O dispositivo que recebe a requisição procura pelo serviço web no seu repositório local (passo 4). Se até o fim do passo 4 não for encontrada nenhuma instância, a busca é propagada pela web (passo 5a e 5b). Se o dispositivo não puder acessar diretamente a web, o dispositivo procura por algum dispositivo como o serviço Gateway rodando (passo 5a). A busca na internet está baseada em algum repositório universal, que no caso do WSAmI é implementado pelo serviço UR (passo 6). Assim, o processo de busca acaba consultando o repositório de serviços como último recurso.

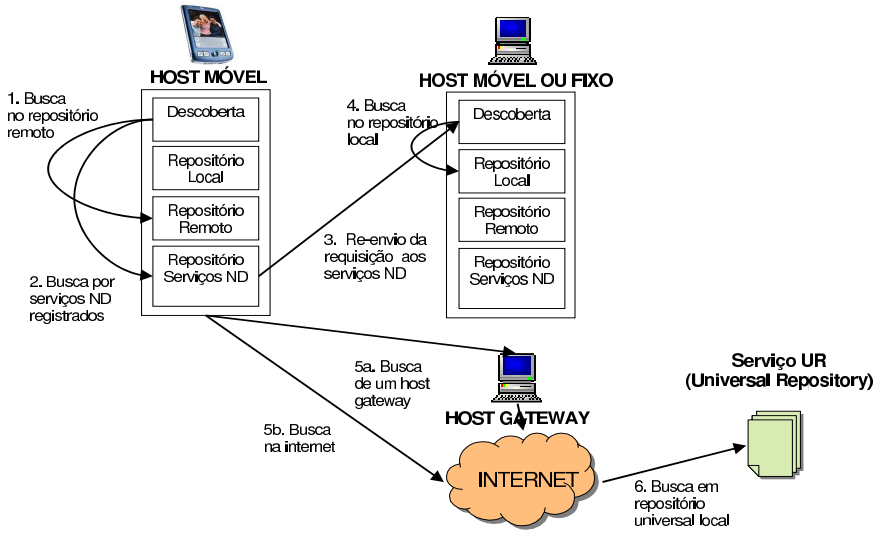

Figura 3.8: Algoritmo de busca de instâncias de serviços web.

3.7 Discussão dos projetos

A partir dos projetos de ambiente inteligentes estudados, podemos extrair algumas características que eles têm em comum no referente a arquitetura e

funcionalidades:

Interfaces adequadas com os usuários: Este ponto se refere à interação homem-computador que acontece no ambiente. O projeto Oxygen desenvolveu ferramentas que permitem implementar aplicações clientes e servidoras de reconhecimento de voz e imagens. O objetivo é criar interfaces simples para os usuários; interfaces simples não implica interfaces primitivas.

A interface do usuário com o ambiente define o quanto o usuário percebe que está interagindo com um sistema computacional. Como no caso do projeto Aura, que reconfigura automaticamente os dispositivos para que o usuário nem perceba mudanças de contexto que ele mesmo pode ter provocado.

O objetivo neste ponto é que o usuário não perceba que faz parte de um sistema computacional.

Ubiqüidade da comunicação e informação: Como as entidades do ambiente possuem absoluta mobilidade, é necessário que os dispositivos não percam comunicação com o ambiente ou com outros dispositivos no ambiente. O projeto Oxygen desenvolveu arcabouços que permitem a criação de redes de dispositivos móveis, que são dinâmicas e autoconfiguráveis. Algumas aplicações implementadas pelo Oxygem ativam um sistema de arquivos distribuído robusto e com tolerância a falhas que garante a ubiqüidade de informação num ambiente móvel.

O projeto WSAmI, por outro lado, aproveita a ubiqüidade da Internet através dos serviços web para o acesso a informação e serviços. Como é possível acessar a web a partir de quase qualquer dispositivo e os serviços web não possuem estado[5], esta abordagem é a que oferece a maior ubiqüidade de comunicação e informação.

Já o projeto Aura pretende alcançar esta ubiqüidade através da sua arquitetura, que possui uma entidade (*Aura do usuário - Prism*) encarregada de manter esta ubiqüidade perante mudanças de contexto dos dispositivos. Esta abordagem está mais de acordo com os métodos empregados em agentes móveis.

A abordagem do projeto MoCA utiliza métodos dos sistemas distribuídos, onde são empregados protocolos para a localização de serviços,

[5] A especificação de serviços web os define como componentes de software que não mantêm estado com o cliente. Porém, existem implementações, como a feita por WebLogic, que permitem estados nos serviços web.

proxies, interceptadores e replicação.

Minimizar a configuração de novas entidades no ambiente: Conforme será discutido na próxima seção, uma arquitetura baseada em serviços permite tratar qualquer componente de software ou hardware de maneira desacoplada e altamente interoperável. Estas duas características facilitam a integração de novas entidades a um sistema existente, sendo que cada entidade pode ser representada como um serviço.

Assim, os projetos Aura, Gator Tech e WSAmI empregam uma arquitetura baseada em serviços, onde todos os dispositivos e aplicações no ambiente são gerenciados e acessados como serviços.

Uma arquitetura baseada em serviços pode ser implementada com diferentes tecnologias, como é o caso do projeto Gator Tech que emprega o arcabouço OSGi ou o projeto WSAmI que emprega serviços web tradicionais. Já o projeto Aura dá uma definição mais abstrata de serviços no ambiente, deixando livre a implementação de um serviço.

Coleta, armazenamento e processamento da informação de contexto: Em todos os projetos é evidente a necessidade de empregar informação de contexto dos dispositivos e usuários que se encontram no ambiente. Isto exige que o ambiente possua um mecanismo para coletar, processar e representar informação contextual adequadamente.

A localização do usuário é a informação contextual da qual é possível tirar mais proveito. Conforme se apresenta em [42], é possível empregar a localização para disparar automaticamente processos no ambiente que podem ser vistos como respostas do ambiente ao usuário.

Além disso, outro tipo de informação contextual pode ser empregado para adaptar a maneira como o ambiente interage com os dispositivos. Por exemplo, personalizar a largura de banda, tipos de mensagens na comunicação, tipo de aplicação a ser executada, etc.

Os *proxies* do MoCA, o observador de contexto do Aura, a camada de gerenciamento de contexto do Gator Tech e as tecnologias de percepção do Oxygen são componentes encarregados desta funcionalidade.

A partir desta informação contextual é possível ativar diferentes funcionalidades em aplicações ou componentes da mesma arquitetura. Assim, no Oxygen todas as aplicações podem acessar informação contextual. Por exemplo, as aplicações de rede podem empregar as tecnologias de percepção para detectar quantos usuário estão na sala e escolher o melhor canal de comunicação, Bluetooth, Wi-Fi, etc.

A acessibilidade e gerenciamento da informação contextual têm a ver com a ubiqüidade de comunicação e informação; ou seja, qualquer informação contextual dever se confiável e acessível de qualquer lugar, em qualquer momento e por qualquer dispositivo. O MoCA como middleware para dispositivos móveis cobre esta necessidade.

Conforme foi apresentado na Seção 2.2 a modelagem de contexto é importante para permitir um processamento do contexto que ofereça informações relevantes. Nenhum dos projetos emprega ontologias para a modelagem de contexto. O projeto Gator Tech considera empregar linguagens e mecanismos de raciocínio baseados em ontologias na sua camada de conhecimento, mais isso ainda não foi implementado.

Previsão e monitoração das intenções dos usuários: Isto tem a ver com a *inteligência* do ambiente. O objetivo é que o ambiente aja de maneira inteligente para o usuário. O ambiente pode ser puramente reativo a mudanças de contexto do usuário, como é o caso do projeto Gator Tech; ou pode ajudar o usuário a realizar tarefas de longa duração, como é o caso do projeto Aura.

Os pontos analisados nesta seção, sobre os projetos de ambientes inteligentes, são empregados nos próximos capítulos para o desenvolvimento do modelo proposto.

Capítulo 4

Serviços Web Semânticos em Ambientes Inteligentes

Neste trabalho empregamos a ontologia WSMO (*Web Services Modeling Ontology*) [39] para a descrição semântica dos serviços que se encontram no ambiente. Esta ontologia foi projetada para a especificação de serviços web semânticos para permitir a descoberta, execução, monitoração e composição automática de serviços web. Essas características nos permitem ativar um ambiente inteligente que possa fornecer a seus usuários os serviços adequados para desenvolver uma tarefa especificada.

O emprego de uma ontologia para serviços web semânticos em um ambiente de computação móvel e sensível ao contexto exige que se leve em consideração aspectos como: mobilidade, descrição do contexto e localização. Neste capítulo apresentamos os principais componentes da WSMO e mostramos como ela foi usada para a especificação de serviços.

Começamos este capítulo com a descrição: (1) da arquitetura de serviços web como sistemas distribuídos baseados em *HTTP/SOAP*, e (2) dos serviços web semânticos, que são extensões da web semântica para serviços web. Nas seções seguintes são descritos os elementos que compõem uma especificação de serviços web usando a WSMO. Apresentamos também a linguagem formal em que se baseia a WSMO, a WSML (*Web Services Modeling Language*) [55], e a sua relação com a *F-Logic* [26].

4.1 Serviços web semânticos

As tecnologias na implementação de sistemas computacionais distribuídos passou por várias gerações. Numa primeira geração os sistemas forne-

ciam processamento centralizado que era acessado e controlado por terminais simples sem capacidades significativas de processamento de dados. Na segunda geração, os sistemas foram organizados em servidores de propósito geral fornecendo processamento, dados, arquivos e aplicações a clientes que interagiam com os servidores fornecendo entradas e saídas. Nesta geração se encontram os sistemas multi-camada. A terceira geração define que cada nó no sistema é simultaneamente um cliente e um servidor, podendo acessar outros servidores ou oferecer recursos de processamento, dados, etc. Este tipo de sistemas são chamados de ponto-a-ponto (*peer-to-peer*). A quarta geração de sistemas são os cooperativos, onde um conjunto de componentes autônomos, ativos e heterogêneos são ativados conjuntamente para fornecer capacidades de processamento.

Os sistemas pertencentes a esta última geração são chamados de sistemas abertos, onde o termo aberto se refere a que os componentes envolvidos são autônomos, heterogêneos e as configurações do sistema podem mudar dinamicamente. A implementação deste tipo de sistemas envolve [44, 51]:

1. *Autonomia:* Cada componente trabalha isoladamente sob seu próprio controle.

2. *Heterogeneidade:* Os componentes no sistema são diferentes em seu projeto e construção.

3. *Dinamismo:* Os componentes podem mudar seu comportamento dependendo da sua configuração no sistema. Componentes podem ser adicionados ou retirados do sistema.

A computação orientada a serviços (*service-oriented computing*) é o paradigma que permite lidar com estas necessidades, permitindo implementar os sistemas da quarta geração. Esses sistemas apresentam, como os sistemas multi-agentes (MAS), a idéia de construir agentes autônomos que possam resolver problemas de maneira cooperativa e inteligente [49].

Assim como no paradigma orientado a objetos –onde os objetos são os componentes principais– os serviços são os principais componentes deste paradigma. Arquiteturas orientadas a serviços fornecem as seguintes capacidades [44]:

Desacoplamento: Permite uma alta flexibilidade na comunicação entre os componentes do sistema.

Implementação independente: Podem ser empregadas diferentes tecnologias na implementação dos serviços.

Configuração Flexível: Desde que cada serviço é um componente de software altamente desacoplado é fácil ser configurado sem afetar outros componentes do sistema.

Gerenciamento do ciclo de vida: Um serviço pode implementar interfaces para gerenciamento do seu ciclo de vida.

Granularidade: A implementação de um serviço pode cobrir diferentes níveis nas funcionalidades do sistema.

Os *serviços web* são uma implementação deste tipo de arquitetura e computação. Um serviço web, conforme definido pelo W3C [51], é uma abstração de uma funcionalidade que deveria ser implementada por um agente concreto. Um agente é um componente de software ou hardware que envia ou recebe mensagens e o serviço é o recurso caracterizado pela sua funcionalidade abstrata. A arquitetura dos serviços web (*Web Services Architecture*) é uma implementação da arquitetura orientada a serviços (*Service Oriented Architecture*) onde os sistemas distribuídos que interagem são definidos como serviços. Um serviço é uma função bem definida, auto contida e que não depende do contexto ou estado de outros serviços [30, 44].

A especificação de serviços web os define como sistemas distribuídos que empregam pacotes SOAP (*Simple Object Access Protocol*) sobre HTTP (*Hypertext Transfer Protocol*) para a troca de mensagens. A descrição da interface dos serviços é feita em WSDL (*Web Service Description Language*). A interface dos serviços se refere à descrição dos métodos e parâmetros que o serviço disponibiliza para chamadas remotas.

Assim, os serviços web são aplicações disponíveis na web que fornecem dados e serviços a outras aplicações. Para conseguir maior interoperabilidade entre eles foi definida uma arquitetura em camadas que implementa os diferentes níveis na arquitetura de serviços web. Linguagens têm sido propostas que permitem a busca, publicação e descrição de serviços web [51]:

1. Serviço de Publicação: UDDI.
2. Descrição: WSDL.
3. Dados: XML, XML-RPC.
4. Mensagens XML: SOAP.
5. Transporte: Basicamente HTTP, podendo ser também FTP, SMTP, IIOP ou Java RMI.

A arquitetura de serviços web considera um *provedor de serviços*, encarregado de anunciar os serviços que são fornecidos por ele. O provedor registra os seus serviços no repositório UDDI (Universal Discovery and Description Integration, registro global para serviços web), que cumpre o papel de um *publicador de serviços*. Um *cliente de serviços* consulta e descobre a interface de um serviço web nesse repositório. A partir dessa interface a aplicação cliente pode se comunicar e utilizar os serviços do serviço web.

A interface do serviço web é descrita em um documento WSDL, que é um documento XML que contém um conjunto de definições que descrevem as operações que podem ser executadas no serviço web, a localização do serviço web e o protocolo de transporte a ser empregado. A interface WSDL do serviço pode ser utilizada para a geração automática de código, permitindo a chamada remota das operações do serviço web.

Os serviços web permitem chamadas a métodos remotos, em diferentes plataformas e linguagens de programação, a partir do documento WSDL. Isso permite desconsiderar o problema de *como* executar um serviço, mas não resolve o problema da execução e descoberta automática de serviços, ou seja, *quais* serviços executar.

A *web semântica* [15] tem por objetivo melhorar as potencialidades da web através da criação de ferramentas e de padrões que permitam atribuir significados claros (*semântica única*) aos conteúdos das páginas web, bem como facilitar a sua publicação e manutenção. As páginas construídas usando a filosofia da web semântica passam a fazer parte de um meio universal para a troca de informação [51]. O meio é universal porque estas páginas podem ser lidas por humanos e por máquinas e portanto podem ser apresentadas, por exemplo, graficamente (num monitor ou impressão em papel) ou ser lidas por um sintetizador de voz.

A web semântica emprega ontologias como uma ferramenta essencial para a generalização de conceitos compartilhados na web. Para que duas ontologias possam interagir é necessário contar com uma sintaxe compatível. Para isso foram criadas linguagens para a especificação de ontologias, ou em termos mais gerais, para o compartilhamento de conhecimento [15]. Linguagens como DAML, DAML-OIL e OWL permitem a especificação de ontologias específicas para a web.

Para trazer as potencialidades da web semântica aos serviços web surgiram ontologias especialmente criadas para a especificação semântica de serviços web, surgindo assim os *serviços web semânticos*. Uma linguagem para especificação de ontologias como esse propósito é a OWL-S (*Ontology Web Language for Services*)[31] que é uma extensão da OWL. Uma outra ontologia é a WSMO (*Web Services Model Ontology*)[39], que é parte de um

arcabouço ainda em desenvolvimento, especialmente projetado para a execução, descoberta e monitoração de serviços web semânticos, o WSMF (*Web Service Management Framework*) [17].

O emprego da web semântica tem por objetivo uma *especificação não ambígua* dos serviços web, incluindo uma especificação dos meta-dados envolvidos na troca de mensagens e o modelo de comunicação entre as aplicações cliente e o serviço web. Uma vantagem dessa incorporação é que é possível fazer buscas automáticas dos serviços adequados para atingir uma determinada requisição de uma aplicação cliente. O WSDL permite informar a uma aplicação como trabalha um serviço web, OWL-S e WSMO descrevem *o quê* o serviço web oferece e *o como*. Nesse sentido, OWL-S e WSMO complementam WSDL, proporcionando informação adicional necessária para que uma aplicação possa descobrir, compor, monitorar e executar serviços web automaticamente [5].

Intituivamente a especificação de serviços web semânticos requer os seguintes pontos [39, 54]:

1. Especificar *o quê faz* o serviço web: as pré-condições, entradas, pós-condições e efeitos dos serviços web. Em OWL-S isso se define no documento *Service Profile*, e em WSMO se define no conceito `Capability`. A descoberta automática de serviços web é feita com base nesse elemento.

2. Especificar *como trabalha* o serviço web: o fluxo de execução do serviço web, ou mais simplesmente, o comportamento do serviço web. Ou seja, a seqüência e dependências entre as funcionalidades do serviço web. Em OWL-S esse fluxo é definido no documento *Service Process*, e em WSMO é definido no conceito `Interface`, que por sua vez está dividido no ponto do vista da aplicação cliente e o ponto de vista de outros serviços web.

3. Definição de interfaces WSDL: o mapeamento entre os objetos definidos na ontologia e os tipo de dados e operações definidos na interface WSDL do serviço web. Em OWL-S esse mapeamento é feito no documento *Service Grounding*. A WSMO ainda não tem uma especificação final sobre o mapeamento a WSDL, mas é possível seguir o modelo empregado pela OWL-S.

Uma diferença importante na descrição do comportamento do serviços web é que a WSMO faz essa descrição baseada em autômatos de estados finitos empregando sentenças do tipo `if <condição> then <atualização>`

em lugar de construções do tipo `If-Then-Else`, `Sequential`, `Composed`, etc. empregadas pela OWL-S.

Além disso, a WSMO acrescenta a descrição da interoperabilidade entre dois serviços web, ou seja, a forma como um serviço web opera em conjunto com outros serviços web. Essa interação é definida no conceito `Orchestration` que faz parte do conceito `Interface`. A descrição da interação entre os clientes e o serviço web (também fornecida dela OWL-S) é feita no conceito `Choreography`.

Esses dois últimos pontos são detalhados nas próximas seções deste capítulo.

4.2 WSMO na descrição de serviços

A WSMO (*Web Services Modeling Ontology*) é uma ontologia que faz parte de um arcabouço para serviços web semânticos, o WSMF (*Web Service Mana-gement Framework*) o qual define um cenário para a descoberta, execução, monitoração e composição automática de serviços web semânticos. Também faz parte do WSMF a linguagem WSML (*Web Services Modeling Language*) sobre a qual se baseou a WSMO. O WSMF usa a WSMO para a descrição dos diferentes elementos que compõem a especificação de um serviços web semântico [17].

É necessário fazer uma distinção entre o WSMF e a WSMO. O WSMF define a especificação do arcabouço necessário para a ativação de serviços web semântico, como especificação semântica, ambientes de execução, linguagens, etc. A WSMO detalha essa especificação em uma ontologia, criando os conceitos necessários para representar cada um dos elementos definidos nessa especificação. Assim, a WSMO define o meta-modelo definido pelo WSMF.

A WSMO, até sua última versão 1.2, define quatro elementos como componentes da definição de um serviço web semântico: (1) Ontologias, que definem a terminologia a ser empregada por outros elementos WSMO; (2) Mediadores, empregados para a associação entre elementos WSMO; (3) Descrição semântica do serviço web e (4) Metas, que descrevem os objetivos que os usuários pretendem alcançar com a execução de um serviço. A relação entre esses elementos é apresentada na Figura 4.1.

Cada um desses elementos faz parte da especificação completa de um web service semântico. Cada elemento é definido de maneira individual (nenhum é incluído dentro de outro) e a associação entre elementos é feita de maneira explícita. Por exemplo, para poder fazer a descrição de um serviço web é necessário declarar as ontologias empregadas na sua especificação. Isto

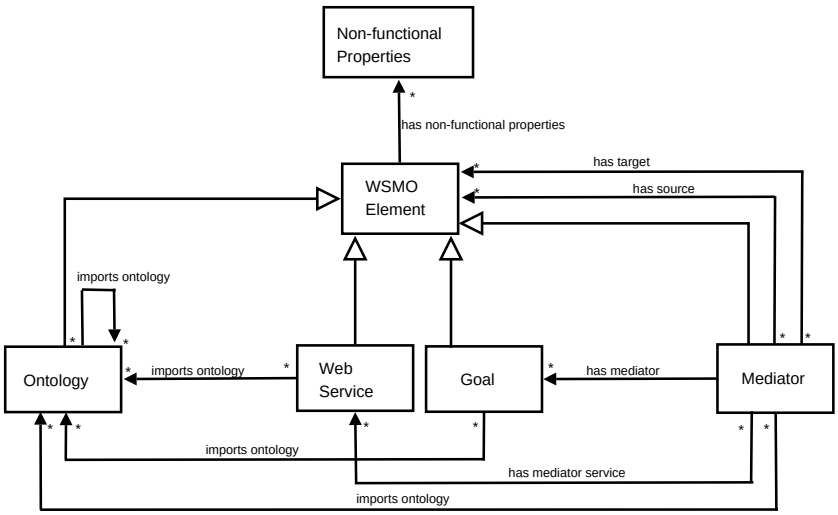

Figura 4.1: Diagrama entidade-relação dos elementos da WSMO [39].

facilita a reutilização e compartilhamento dos elementos.

A seguir apresentamos mais detalhadamente cada um desses elementos e mostramos as considerações empregadas na especificação dos serviços para a modelagem de contexto e execução automática de serviços em um ambiente inteligente.

4.2.1 Ontologias

Na WSMO as ontologias são a ligação conceitual com o mundo real, definidas e aceitas por comunidades de usuários. As ontologias definem uma terminologia de acordo comum que fornece conceitos e relações entre conjunto de conceitos. Com o objetivo de capturar a semântica implícita nos dados representados, as ontologias também permitem a especificação de axiomas e relações entre os conceitos representados.

As definições de conceitos numa ontologia são empregadas por todos os elementos que compõem o serviço web semântico, formando assim um repositório de conceitos. Na especificação de qualquer elemento é necessário declarar quais as ontologias empregadas. Um serviço web semântico pode estar baseado num conjunto de ontologias, e algumas das quais podem ser usadas também por outros serviços web. As ontologias facilitam o compartilhamento do conhecimento entre um conjunto de serviços web. Uma característica importante da WSMO (que a diferencia da OWL-S) é que ela

define as ontologias como elementos separados da descrição do serviço web semântico.

A WSMO chama de classes os conceitos que pertencem ao meta-modelo e são definidos pelo WSMF. `Ontology`, `Web Service`, `Choreography`, `Goal`, `Interface`, etc. são algumas dessas classes. No resto do texto referenciamos como classes esses conceitos da WSMO. Na Figura 4.2 se mostra a classe `Ontology` e suas propriedades. As propriedades `hasConcept, hasRelation, hasFunction, hasInstance e hasAxiom` são listas que podem conter múltiplas definições das classes `Concept, Relation, Function Instance e Axiom`, respectivamente. A definição de uma ontologia implica definir uma lista de conceitos, relações ou funções entre esses conceitos, instâncias de conceitos e axiomas sob a ontologia.

```
1   Class Ontology
2      hasNonFunctionalProperties type nonFunctionalProperties
3      importsOntology type ontology
4      usesMediator type ooMediator
5      hasConcept type concept
6      hasRelation type relation
7      hasFunction type function
8      hasInstance type instance
9      hasAxiom type axiom
```

Figura 4.2: Classe `Ontologia`.

A classe `Ontology` é usada para a especificação de uma ontologia em WSMO. Na Figura 4.3 se mostra uma parte da ontologia `LibraryOnto` usada para a especificação do serviço `Library`, apresentada nas seções seguintes do capítulo. Esta ontologia emprega conceitos importados de outras ontologias (sentenças `import` das linhas 11-15), as referências a esse conceitos é feita empregando os espaços de nomes (*namespaces*) declarados nas linhas 2-7. Na linha 1 é declarado o espaço de nomes da própria ontologia empregado para a definição, ou seja, tal que todos os conceitos, axiomas, instâncias, relações, etc. declarados nesta ontologia possam ser referenciados usando esse espaço de nomes.

Nas linhas 19-26 da Figura 4.3 se faz a declaração do conceito `Book` que representa um livro e seus atributos `hasName`, `hasISBN`, `hasTheme`, etc. A definição de conceitos se baseia no meta-modelo da classe `Concept` que possui as propriedades mostradas na Figura 4.4. A definição dos atributos do

```
1   namespace { _"http://ambienttask/ontologies/LibraryOnto#",
2     lib _"http://ambienttask/ontologies/LibraryOnto#",
3     loc _"http://ambienttask/ontologies/LocationOnto#",
4     peo _"http://ambienttask/ontologies/PeopleOnto#",
5     fex _"http://ambienttask/ontologies/FenixOnto#" }
6
7   ontology _"http://ambienttask/ontologies/LibraryOnto"
8
9   importsOntology
10    { _"http://ambienttask/ontologies/LocationOnto",
11      _"http://ambienttask/ontologies/PeopleOnto",
12      _"http://ambienttask/ontologies/FenixOnto"}
13
14  axiom listaxiom
15     definedBy
16        ?somebook[hasISBN hasValue ?somISBN] memberOf Book
17           implies   ?somelist memberOf CourseBooks.
18
19  concept Book
20     nonFunctionalProperties
21        typeWSDL hasValue "Book"
22     endNonFunctionalProperties
23     hasName impliesType _string
24     hasAuthor impliesType _string
25     hasISBN impliesType _string
26     hasTheme impliesType BookTheme
27
28  concept BookPlace subConceptOf loc#Region
29     nonFunctionalProperties
30        typeWSDL hasValue "BookPosition"
31     endNonFunctionalProperties
32     inArea impliesType _string
33     inStand impliesType _string
34     inPanel impliesType _string
```

Figura 4.3: Exemplo de especificação de uma ontologia WSMO

conceito e feito através da propriedade `hasAttribute` que representa uma lista de atributos segundo a classe `Attribute`. Para cada atributo é definido o nome do atributo e o tipo de dado associado a ele. Os tipos dos atributos podem ser outros conceitos ou algum tipo primitivo definido segundo o espaço de nomes dado pelo WSMO em `http://www.wsmo.org/wsml/ wsml-syntax#`. Alguns desses tipos são: `string`, `integer`, `date`, `boolean`, etc. O superconceito ao qual poderia pertencer o conceito é definido pela propriedade `hasSuperConcept`, como se mostra na linha 28 da Figura 4.3.

Além dos conceitos, podem ser definidos axiomas e relações sobre conceitos da ontologia. Nas linhas 14-17 da Figura 4.3 se define o axioma `list`, que define que um `Book` com a propriedade `hasISBN` implica o conceito `ListBooks`.

```
1  Class Concept
2      hasNonFunctionalProperties type
                          nonFunctionalProperties
3      hasSuperConcept type concept
4      hasAttribute type attribute
5      hasDefinition type logicalExpression
                         multiplicity = single-valued
6
7  Class Attribute
8      hasNonFunctionalProperties type
                               nonFunctionalProperties
9      hasRange type concept  multiplicity = single-valued
```

Figura 4.4: Classe `Concept` e `Attribute`.

Cada elemento definido no WSMO possui uma seção de `nonFunctionalProperties`, que é usado para a descrição de propriedades não funcionais do elemento, representadas por entradas chave-valor. Algumas chaves dessas propriedades são: `Contributor`, `Coverage`, `Creator`, `Date`, `Description`, `Format`, `Identifier`, `Language`, `Owner`, `Publisher`, `Relation`, `Rights`, `Source`, `Subject`, `Title`, `Type` e `Version` que indicam, intuitivamente, uma propriedade do elemento.

Na implementação do modelo proposto empregamos a chave `typeWSDL` (das propriedades não funcionais) para fazer o mapeamento entre conceitos definidos na ontologia e tipos definidos no documento WSDL do serviço web, formando pares do tipo `(typeWSDL,`*<tipo WSDL>*`)`. Nas linhas 21

e 30 da Figura 4.3 se mostra um exemplo desse mapeamento. Os valores `Book` e `BookPosition` da chave `typeWSDL` são os nomes dos tipos definidos no arquivo WSDL do serviço web. Assim, na execução de um serviço web os conceitos são convertidos nos tipos associados para serem passados como parâmetros aos métodos do serviço web definidos na sua interface WSDL.

4.2.2 Web Services

A especificação de serviços web envolve basicamente dois pontos: (1) a funcionalidade do serviço –*o que faz* e (2) como a funcionalidade do serviço web será atingida –*como trabalha*. Para o primeiro ponto é empregada a classe `Capability` associada com a propriedade `hasCapability` da classe `Service`; o segundo ponto é descrito na classe `Interface` na propriedade `hasInterface` da mesma classe, conforme mostrado na Figura 4.5.

```
1  Class Service
2     hasNonFunctionalProperties type nonFunctionalProperties
3     importsOntology type ontology
4     usesMediator type {ooMediator, wwMediator}
5     hasCapability type capability multiplicity = single-valued
6     hasInterface type interface
```

Figura 4.5: Classe `Service`.

Na Figura 4.6 se mostra o cabeçalho da especificação do serviço `Library`. Para permitir a execução automática do serviço é necessário possuir as classes que representam os tipos definidos na interface WSDL do serviço web, por razões que serão explicadas no Capítulo 5. Fazemos essa associação através das propriedades não funcionais do serviço web, empregando a chave `urlClasses` cujo valor é uma URL do arquivo *jar* que contém as classes Java. Na linha 9 da Figura 4.6 se mostra o caso para o serviço `Library`.

Interface do serviço

A classe `Capability` descreve as capacidades do serviço web, que são as suas funcionalidades do serviço web com base em suas pré-condições, hipóteses, pós-condições e efeitos (IOPEs) (Figura 4.7). A especificação de cada uma dessas propriedades é feita através de instâncias da classe `Axiom` que permite a declaração de uma lista de expressões lógicas em WSML.

```
1  namespace { _"http://ambienttask/wsami/LibraryWService#",
2     lib _"http://ambienttask/ontologies/LibraryOnto#",
3     loc _"http://ambienttask/ontologies/LocationOnto#",
4     peo _"http://ambienttask/ontologies/PeopleOnto#",
5     fex _"http://ambienttask/ontologies/FenixOnto#" }
6
7  webService _"http://ambienttask/wsami/Library"
8    nonFunctionalProperties
9      urlClasses hasValue "http://ambienttask/code/library.jar"
10   endNonFunctionalProperties
```

Figura 4.6: Descrição WSMO do serviço `Library`.

Na Figura 4.8 se mostra a especificação das capacidades do serviço *Library*(`LibraryCapability`). Como exemplo, a hipótese definida para o serviço nas linhas 3-5 coloca a restrição que para poder executar o serviço o usuário deveria estar localizado no `ime`. A execução do serviço produz uma lista de livros relacionados (`?listbooks memberOf lib#RelatedBooks` linha 18) e uma lista de livros de disciplinas (`?listbooks memberOf lib#CourseBooks` linha 22).

Empregamos a WSMO para a especificação de serviços disponíveis no ambiente, portanto fazemos distinção entre os pares hipóteses/efeitos e pré-condições/pós-condições. Usamos as hipóteses e efeitos para a descrição do contexto do ambiente em que o serviço pode ser executado; e usamos as pré-condições e pós-condições para especificar os parâmetros de entrada e saída, respectivamente, como conseqüências da execução do serviço.

Comportamento do serviço web

O segundo ponto na especificação de serviços web semânticos (o *como trabalha* o serviço web) está dividido por sua vez em duas partes: (1) a especificação de como trabalha do ponto de vista das aplicações cliente e (2) a especificação de como ele trabalha do ponto de vista de outros serviços web. A classe `Choreography` é usada para o primeiro ponto e a classe `Orchestration` para o segundo ponto. Ambas classes são parte da classe `Interface` (linhas 5 e 6 da Figura 4.9) que está associada à classe `Service` (linha 6 da Figura 4.5).

A descrição de ambos comportamentos se baseia em estados que mudam

```
1  Class capability
2      hasNonFunctionalProperties type nonFunctionalProperties
3      importsOntology type ontology
4      usesMediator type {ooMediator, wgMediator}
5      hasSharedVariables type sharedVariables
6      hasPrecondition type axiom
7      hasAssumption type axiom
8      hasPostcondition type axiom
9      hasEffect type axiom
10
11  Class axiom
12     hasNonFunctionalProperties type nonFunctionalProperties
13     hasDefinition type logicalExpression
```

Figura 4.7: Classe `Capability` e classe `Axiom`.

segundo uma máquina de estados finitos. Cada estado é definido com base em conceitos de uma ontologia e as mudanças sobre esses estados se cumprem segundo *regras de transição*. O agente que executa os serviços web carrega um modelo do mundo que resulta da interação com o ambiente e a execução dos serviços web muda esses estados. Assim, as mudanças de estado que os serviços web expressam nas suas regras de transição são incorporadas ao modelo do mundo do agente.

As regras de transição acrescentam, eliminam ou atualizam sentenças no novo estado se determinadas condições forem cumpridas. As regras de transição têm a forma `if <condições> then <atualizações>`. A parte `<atualizações>` são sentenças que indicam a ação a ser efetuada sobre o mundo representado pelo agente. Existem três ações: (1) `add`, adiciona um fato, (2) `update`, atualiza o fato implicado com a nova informação e (3) `delete`, remove um fato. A descrição dos serviços no ambiente emprega somente atualizações do tipo `add` que adiciona fatos ao estado do cliente. É por isso que chamamos de *efeito* a esta parte de uma regra de transição.

As linha 18-21 da Figura 4.10 mostra a regra de transição que indica que será adicionado o fato `?related books memberOf lib#RelatedBooks` ao modelo do agente se a condição `?someStudent[hasLocation hasValue ?someLocation] memberOf fex# StudentFenix` for cumprida. No ambiente inteligente esta regra de transição é interpretada como dizendo que uma lista de livros relacionados será obtida (`?relatedbooks memberOf lib#Related-`

```
1  capability _"LibraryCapability"
2
3  assumption assumption1
4     definedBy
5        ?user[hasLocation hasValue loc#ime] memberOf peo#Ademic.
6
7  precondition precond1
8     definedBy
9        ?book[hasISBN hasValue ?isbn] memberOf lib#Book.
10
11 precondition precond2
12    definedBy
13       ?book[hasTheme hasValue ?theme] memberOf lib#Book
14          and  ?theme memberOf lib#BookTheme.
15
16 postcondition postcond1
17     definedBy
18       ?listbooks memberOf lib#RelatedBooks.
19
20 postcondition postcond2
21     definedBy
22       ?listbooks memberOf lib#CourseBooks.
```

Figura 4.8: `Capability` do serviço `Library`.

```
1  Class interface
2      hasNonFunctionalProperties type nonFunctionalProperties
3      importsOntology type ontology
4      usesMediator type ooMediator
5      hasChoreography type choreography
6      hasOrchestration type orchestration
```

Figura 4.9: Classe `Interface`.

`Books`) a partir da localização do estudante (`?someStudent[hasLocation hasValue ?someLocation] memberOf fex#StudentFenix`).

A Figura 4.11 representa a regra de transição como estados no modelo do agente. Se as condições da regra foram satisfeitas no Estado 1 as atualizações da regra serão efetivadas levando o modelo do agente ao Estado 2. O uso de regras de transição implica que diferentes estados podem ser alcançados dependendo da ordem em que as regras de transição forem aplicadas. O mecanismo para a execução e descoberta de serviços deve levar em consideração essa possibilidade.

Para permitir a execução automática do serviço web é feito o mapeamento das regras de transição a operações concretas na interface WSDL do serviço web. Estas associações são feitas nas propriedades não funcionais da `Choreography` do serviço web (linhas 3-8 da Figura 4.10). Para as operações definidas no WSDL: `booksForPosition`, `findBookByTheme`, `takeBook` e `findBookByIsbn` são definidas chaves que representam as regras de transição associadas aos métodos. Assim, quando o sistema achar que para chegar a um estado deve ser aplicada uma certa regra de transição será executado o método concreto correspondente do serviço web.

4.2.3 Metas

As metas são especificações de pós-condições e efeitos (objetivos) que o usuário deseja conseguir quando executa um serviço web; a especificação desses objetivos é comparada com a descrição das capacidades do serviço web. Se o serviço web satisfaz o objetivo a interação com o web service é iniciada segundo sua `Choreography`.

Estes elementos não foram usados diretamente no desenvolvimento do modelo proposto, mas sua funcionalidade foi mantida. Na Figura 4.12 se apresenta como exemplo da especificação da meta `LibraryGoal` que con-

```
1  interface LibraryInterface
2    choreography _"urn:LibraryChoreography"
3    nonFunctionalProperties
4       someLocation\-relatedbooks  hasValue "booksForPosition"
5       someTheme\-disciplinebooks hasValue "findBookByTheme"
6       someBook\-someBook hasValue "takeBook"
7       someISBN\-relatedbooks hasValue "findBookByIsbn"
8    endNonFunctionalProperties
9    transitionRules courseBooks
10      if (?someTheme memberOf lib#BookTheme) then
11         add(?coursebooks memberOf lib#AvailableBooks)
12      endIf
13
14      if (?someBook[isAvailable hasValue true] memberOf lib#Book) then
15         add(?someBook [isAvailable hasValue false] memberOf lib#Book)
16      endIf
17
18      if (?someStudent[hasLocation hasValue ?someLocation]
19             memberOf fex#StudentFenix) then
20         add(?relatedbooks memberOf lib#RelatedBooks)
21      endIf
```

Figura 4.10: Exemplo da `Choreography` do `Library` serviço web.

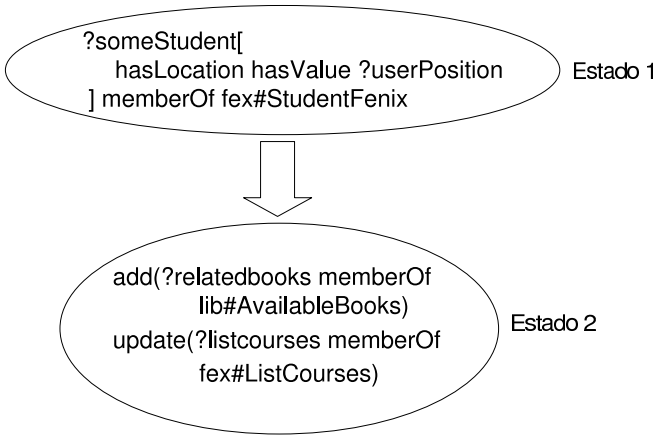

Figura 4.11: Regra de transição do serviço `Library`.

sulta uma lista de livros disponíveis na biblioteca (`?listbooks memberOf lib#AvailableBooks`). Na meta é especificada a capacidade do serviço (linha 13) usado para tentar satisfazer a meta.

Basicamente, em uma meta se faz uma descrição do estado desejado que resultaria da execução de um ou vários serviços web. Nesse ponto se faz necessário um mecanismo para poder avaliar se um determinado serviço web semântico pode atingir uma meta. Esse processo é conhecido como *matching*. Em [33] se apresenta um mecanismo para avaliar numericamente o grau de calçamento entre um serviço web e um objetivo. Baseado nisso, é possível escolher o serviço web mais adequado dentre todos aqueles que podem atingir o objetivo.

4.2.4 Mediadores

Os mediadores são entidades cuja principal função é permitir a interoperabilidade entre diferentes serviços web. Para isso, eles mapeiam os elementos WSMO entre serviços web diferentes. Assim, eles deveriam permitir que dois serviços web pudessem ter a mesma interpretação da informação, apesar de empregarem ontologias diferentes. Uma meta de um serviço web pode ser vinculada completa ou parcialmente à meta de outro serviço web, podendo-se conseguir refinamentos no tratamento de consultas.

```
1   namespace { _"http://ambienttask/wsami/LibraryWService#",
2      lib _"http://ambienttask/ontologies/LibraryOnto#",
3      loc _"http://ambienttask/ontologies/LocationOnto#",
4      peo _"http://ambienttask/ontologies/PeopleOnto#",
5      fex _"http://ambienttask/ontologies/FenixOnto#" }
6
7   goal  _"http://LibraryGoal"
8      importsOntology { _"http://ambienttask/ontologies/LibraryOnto",
9         _"http://ambienttask/ontologies/LocationOnto",
10        _"http://ambienttask/ontologies/PeopleOnto",
11        _"http://ambienttask/ontologies/FenixOnto"}
12
13  capability _"http://LibraryCapability"
14
15  postcondition postcond1
16     definedBy
17        ?listbooks memberOf lib#AvailableBooks.
```

Figura 4.12: Exemplo da especificação de Metas

Existem quatro tipos de mediadores, dependendo dos elementos da WSMO que pretendem relacionar. Entre duas ontologias, um mediador ontologia-ontologia (definido na classe `ooMediator`) definiria o mapeamento entre conceitos semelhantes em ambas ontologias; o mediador serviço-serviço (definido na classe `wwMediator`), pode ser usado para relacionar dois serviços web, definindo como um serviço web pode ser usado no lugar de outro para atender um mesmo tipo de consultas; o mediador serviço-meta (classe `wgMediator`) serve para ligar a meta que pode ser atingida à execução do web service. Assim, várias metas poderiam ser atendidas por vários serviços web; o mediador meta-meta (definido na classe `ggMediator`) define uma meta em termos de outra. Os mediadores ontologia-ontologia poderiam ser empregados para converter ontologias em outra linguagem como OWL-S ou RDF-Schema, embora ainda não seja detalhado esse processo.

Até a última versão da WSMO, os mediadores não foram claramente especificados, e existem diferentes abordagens para sua implementação. Alguns deles consideram os mediadores como serviços web [32] que têm como entrada uma ontologia, meta ou serviço web e fornecem a saída equivalente em outra ontologia, meta ou serviço web respectivamente. Uma outra abordagem é construir mediadores como *middleware* entre clientes e servidores.

4.3 WSML e F-Logic

4.3.1 Web Services Modeling Language - WSML

A WSML (*Web Service Modeling Language*) é a linguagem que fornece sintaxe e semântica formal para a WSMO. A WSML está baseada em lógicas descritivas e lógica de primeira ordem, e incorpora o paradigma orientado a objetos à programação em lógica. A incorporação destes formalismos permite a implementação de mecanismos de inferência formais para a descoberta e composição automática de serviços web semânticos, como a abordagem mostrada em [25].

A WSML fornece dois níveis de especificação: o sintático e o semântico. A sintaxe é facilmente interpretada por humanos e tem bastante similaridade com linguagens orientadas a objetos, embora só sejam definidos os atributos das classes e não seus métodos. A semântica é fornecida através da declaração de cláusulas de Horn que permitem a declaração de axiomas para os diferentes elementos da WSMO. Assim, a especificação de axiomas na WSMO se baseia num formalismo robusto, contrastando com a OWL-S que emprega diferentes linguagens de regras em XML.

A WSML define quatro níveis de abstração [55], que vão desde a descrição de dados e conceitos até o nível de informação:

- Meta-meta-modelo: Descrição da estrutura e semântica do meta-meta-modelo. Em WSMF esse nível está definido pela própria WSML;
- Meta-modelo: Descrição da estrutura que define a estrutura e semântica do nível do meta-modelo. A WSMO constitui esse nível de linguagem, as classes para ontologias, serviços web, metas, mediadores, capacidades, choreography, etc. são entidades desse nível;
- Modelo: Descrição dos dados no nível de informação-especificação de ontologias, metas, mediadores e serviços web segundo o domínio de aplicação;
- Informação: São os dados a serem descritos. Na WSMO se refere a mensagens trocados entre serviços web.

Existem diferentes variantes da WSML segundo o formalismo a ser suportado e o nível de raciocínio requerido. Cada uma dessas variantes possui uma sintaxe própria. Em [55] se apresentam as regras completas em que se baseia a gramática da WSML:

- WSML-Core: Permite a especificação mediante cláusulas de Horn e lógica descritiva, permitindo a declaração de novos tipos. Pode se relacionar perfeitamente como a OWL. Esta variante possui a menor expressividade da WSML, permitindo basicamente a declaração de classes, atributos, relações binárias, e herança de classes. Não possui símbolos para funções nem igualdade;

- WSML-DL: Acrescenta à WSML-Core a especificação completa das lógicas descritivas *SHIQ*;

- WSML-Flight: Extende a WSML-Core permitindo a declaração de restrições, meta-modelos e negação não monotônica. Baseia-se em F-Logic ativando uma linguagem baseada em regras;

- WSML-Rule: Estende a capacidade de programação em lógica fornecida pela WSML-Flight e permite o emprego de funções, símbolos e regras não seguras;

- WSML-Full: Unifica WSML-DL e WSML-Rule, segundo uma declaração de primeira ordem, incrementando a funcionalidade fornecida pela WSML-Rule.

Na próxima seção apresentamos a F-Logic(Frame-Logic), uma lógica que estende a programação lógica com sintaxe orientada a objetos e lógica descriptiva. Por estas características é possível converter especificações em WSML em um programa lógico em F-Logic.

4.3.2 F-Logic

A F-Logic foi desenvolvida como uma linguagem para bases de dados orientados a objetos. Como se sabe, as linguagens orientadas a objetos tentam modelar o mundo com a definição de classes a que pertencem os objetos, os quais representam entidades do mundo real, e relações entre essas classes (herança, especialização, composição, pertinências, etc.). Por outro lado a programação em lógica permite modelar problemas difíceis de modelar por linguagens não-declarativas, baseando-se em uma semântica bem definida de alguma lógica formal.

A F-Logic faz a união das facilidades de modelagem das linguagens orientadas a objetos e a semântica bem definida das lógicas. Isso permite a descrição de ontologias num estilo orientado a objetos [2] A união dessas linguagens é conseguida de maneira que a declaratividade da programação em lógica seja mantida. À semelhança da programação declarativa, se define

o *que* deve ser resolvido (e não *como*), mas seguindo o paradigma orientado a objetos.

A seguir se apresenta uma parte de um programa em F-Logic. O programa mostrado foi desenvolvido em *Flora-2* [50] que é uma implementação da F-Logic baseada em *Prolog* e *HiLog* que implementa a semântica da F-Logic.

```
1   book[hasname *=> string, hasisbn *=> string,
2        hastheme *=> booktheme, inplace *=> bookplace].
3   person[hasid *=> integer, hasname *=> string,
4          haslocation *=> location].
5   listcourses[hascourse *=>> course].
6   bookplace[inarea *=> string, instand *=> string,
7             isavailable *=> boolean].
10  coursefenix[hascode *=> string, hasname *=> string,
11              hasteacher *=> teacherfenix].
13  listbooks[haselements *=>> book].
14  location[posx *=> integer, posy *=> integer, posz *=> integer].
15  booktheme[hasname *=> string, hasdiscipline *=> string].
16
17  coursebooks::listbooks.
18  relatedbooks::listbooks.
19  student::person.
20  bookplace::region.
21  academic::person.
22  studentfenix::student.
23
24  ime:place[hasid->"ime01", hasname->"IME-USP"].
25
28  unique_position1(X) :- X:people, X[haslocation -> Y],
29                                   X[haslocation -> Z],
30                                   Y\=Z.
31  unique_position(X) :- not unique_position1(X).
```

Figura 4.13: Um programa em *F-Logic*

A sintaxe de Flora-2 é semelhante à sintaxe do Prolog com respeito à declaração de predicados, átomos e variáveis. Esses três elementos são chamados de *Id-terms* e são a base dos programas em Flora-2. Um átomo definido como em Prolog poderia representar uma classe, objeto ou atributo

de uma classe em Flora-2.

Átomos de assinatura

A definição de conceitos ou classes é feita usando a sintaxe *nome_classe* $[atributo_1* => tipo_1, atributo_2* =>> tipo_2, ...]$, onde cada $tipo_i$ pode ser uma classe ou algum tipo primitivo (`integer`, `string`, `float` , etc). Estes tipos de termos são chamados de *átomos de assinatura* (*signature atoms*). Os tokens => e =>> são usados para indicar que está sendo definido o tipo de dado do atributo. O token => indica que o atributo está associado a um único valor do tipo especificado, entanto que o token =>> indica que o atributo está associado a um conjunto de valores. A seguinte sentença da linha 13 da Figura 4.13

```
listbooks[haselements *=>> book]
```

indica que a classe `listbooks` no seu atributo `haselements` está relacionada a um conjunto de valores do tipo `book`.

Conforme sugerido em [2], usando o paradigma orientado a objetos são apresentados os nomes dos atributos como métodos associados aos objetos. Assim, ao aplicar o método `hasname` do objeto `ime` (linha 24) se obtém o nome associado ao objeto (`Instituto de Matemática e Estatistica`). A noção de métodos surgiu da possibilidade de empregar predicados para os nomes de atributos. Com isto temos átomos do tipo

```
john[son(bob) ->> {mary, jack}]
```

e uma consulta do tipo

```
john[son(bob) ->> X]
```

instanciaria `X` com `{mary, jack}`, o que intuitivamente significa executar o método `son` com o parâmetro `bob`.

Átomos ISA

Nas linha 17-22 da Figura 4.13 é definida a hierarquia *IS-A* (*"é um"*), que estabelece as relações de sub-classe (sub-conceito em lógicas descritivas) e pertinência. Este tipo de termos é chamado de *átomos ISA* (*ISA-atoms*). A sintaxe `A::B` indica que a classe `A` é uma sub-classe da classe `B` e a sintaxe `O:C` indica que o objeto `O` pertence à classe `C`.

Flora-2 permite a herança de atributos entre classes através dos tokens $*$ => e $*$ =>>. Por exemplo, a classe `student` herda todos os atributos da

classe `person` (linha 19). Assim, é possível consultar pelo atributo `hasname` a partir da classe `student`. Entretanto, os tokens => e =>> não permitem a herança de atributos.

F-moléculas

Objetos são definidos da forma $classe : nome_objeto[atributo_1 -> obj_1 : classe_1, atributo_2 ->> obj_2 : classe_2, ...]$, onde cada obj_i pode ser definido como um novo objeto. Este tipo de termos é chamado de *F-moléculas* (*F-molecules*). Na linha 24 se define o objeto (instância de classe) `ime` como membro da classe `place`. Da mesma maneira que os átomos de assinatura o token $->$ associa o atributo a um valor único e $->>$ associa o atributo a um conjunto de valores. Portanto o tipo de associação do atributo do objeto deve coincidir como o tipo de associação do atributo declarado na classe à qual pertence o objeto.

São chamados moléculas devido a que podem ser divididas em átomos. Por exemplo, a F-molécula da linha 24 é equivalente a:

```
ime:place[hasid->"ime01"] and
ime:place[hasname->"Instituto de Matematica e Estatistica"]
```

Regras

A declaração de regras é feita como em Prolog, ou seja mediante cláusulas de Horn com a forma (`cabeçalho :- corpo`) e conectivos lógicos `,` (conjunção) e `;` (disjunção). Nas linhas 28-30 se mostra uma regra para garantir que qualquer objeto da classe `person` deveria ter somente uma localização associada a seu atributo `haslocation`.

Reificação em Flora-2

A implementação da *reificação* por Flora-2 é importante para a manutenção de crenças, já que permite adicionar novos fatos à base de conhecimentos com uma semântica mais natural. Por exemplo para dizer que `tom` acredita que `alice` acha que Flora-2 é legal (`flora2:coolThing`) poderia ser utilizada a sentença:

```
tom[believes->>alice[thinks->>flora2:coolThing].
```

Mas essa representação não está correta, já que a sentença poderia assumir a forma

```
tom[believes->>alice] and
alice[thinks->>flora2:coolThing].
```

que indica que `tom` acredita somente em `alice`. Para conseguir o efeito desejado é preciso transformar a fórmula

```
alice[thinks->>flora2:coolThing].
```

em um objeto a mais na base de conhecimento. Isso é conhecido como *reificação* de fórmulas. A reificação de fórmulas em Flora-2 é declarada mediante a construção `${...}`. Assim, a sentença anterior pode ser re-escrita como:

```
tom[believes->>${alice[thinks->>flora2:coolThing]}].
```

Pode-se empregar uma construção como a apresentada a seguir para inserir essa nova formula (crença) na base de conhecimentos.

```
tom[believes->>Think] , insert{ Think }.
```

A função primitiva `insert{...}` insere o conjunto de fórmulas especificada no parâmetro à base de conhecimento. Ou seja, as fórmulas inseridas passam a fazer parte do programa. No exemplo, a variável `Think` seria unificada com a fórmula reificada e inserida na base de conhecimento. É possível fazer a mesma coisa com regras empregando a função `insertrule{...}`.

Capítulo 5

Modelo Proposto

Neste capítulo descrevemos o modelo para ambientes inteligentes proposto nessa dissertação. A descrição considera: (1) a arquitetura do modelo, (2) o mecanismo de descoberta de serviços, (3) o mecanismo de execução de serviços e (4) os serviços que pertencem à arquitetura.

A arquitetura do modelo proposto se baseia em uma arquitetura orientada a serviços que utiliza em sua implementação os serviços web semânticos, conforme apresentado na Seção 4.1. No Capítulo 3 foram apresentados os projetos Aura e GatorTech que propõem uma arquitetura para ambientes inteligentes também baseada em serviços.

A descrição semântica dos serviços disponíveis no ambiente é feita empregando a ontologia WSMO. Na Seção 4.2 apresentamos como é feita esta especificação e as considerações para permitir a execução automática dos serviços web e a modelagem do contexto em que os serviços podem ser executados.

O modelo de computação proposto se baseia no modelo *computação dirigida por tarefas*, apresentado na Seção 2.4. Nesse modelo, o usuário se preocupa somente com a tarefa que desenvolve e não com o gerenciamento dos serviços que lhe permitem executar essa tarefa. O gerenciamento de serviços diz respeito à necessidade de descoberta e execução dos serviços mais adequados para executar uma tarefa. A interação com o ambiente deve ocorrer de maneira transparente, conforme apresentado na Seção 2.3: o usuário não percebe que está interagindo com um sistema computacional o qual deve ser configurado passando os parâmetros necessário para a execução dos métodos.

O mecanismo para a descoberta e execução de serviços no ambiente se baseia no formalismo de F-Logic, apresentado na Seção 4.3. No presente

capítulo apresentamos a conversão dos serviços e o mecanismo de descoberta em uma teoria lógica em F-Logic, a partir da qual podemos fazer a descoberta dos serviços para uma tarefa especificada.

No presente capítulo apresentamos também o sistema de manutenção da verdade baseado em hipóteses (*Assumption-based Truth Maintenance System* - ATMS) empregado para manter as relações de maneira dinâmica entre hipóteses/pré-condições e efeitos/pós-condições que surgiriam como conseqüência da execução de serviços. O ATMS nos permite achar os conjuntos de pré-condições minimais que produzem certo efeito. Como apresentaremos neste capítulo o ATMS permite resolver consultas mais dinamicamente.

Foram empregados dois sistemas de middleware para suporte à arquitetura proposta: o WSAmI (Seção 3.7) e o MoCA (Seção 3.5). Empregamos o middleware de-senvolvido pelo projeto WSAmI para suporte à descoberta de serviços web. Conforme foi apresentado na Seção 3.6, a descoberta de serviços web pelo middleware não é feita num nível semântico nem de maneira automática, pois não se estabelece um protocolo para a descoberta de serviços baseado em SLP. Assim, usamos o protocolo de descoberta implementado por este middleware para a busca de instâncias de serviços web numa rede de dispositivos móveis e fixos. O WSAmI nos dá a infra-estrutura necessária para construir nosso mecanismo de descoberta e execução automática de serviços no ambiente. Já o MoCA foi empregado como middleware para o gerenciamento da informação contextual dos dispositivos no ambiente.

5.1 Arquitetura do modelo

A arquitetura pre-supõe uma arquitetura de infra-estrutura para redes sem fio, pois ela se baseia numa entidade centralizada em um nó fixo, encarregada de gerenciar as bases de conhecimento dos dispositivos no ambiente, e contém os mecanismos de descoberta e execução automática de serviços. O mecanismo proposto de descoberta automática precisa de grande capacidade computacional para as inferências lógicas.

A arquitetura proposta se apóia sobre o middleware do WSAmI. Assim, uma característica importante da arquitetura é que os serviços podem ser fornecidos por dispositivos móveis, desde que tenham a capacidade de executar um contêiner de serviços web. Especificamente, deveria poder executar o middleware do projeto WSAmI. Conforme apresentado na Seção 3.6 o WSAmI emprega Jetty e Tomcat como servidores de serviços web para dispositivos móveis e fixos, respectivamente.

Na Figura 5.1 apresentamos os componentes principais da arquitetura proposta. Uma característica da arquitetura proposta, é que toda a comunicação entre os componentes é através da Internet.

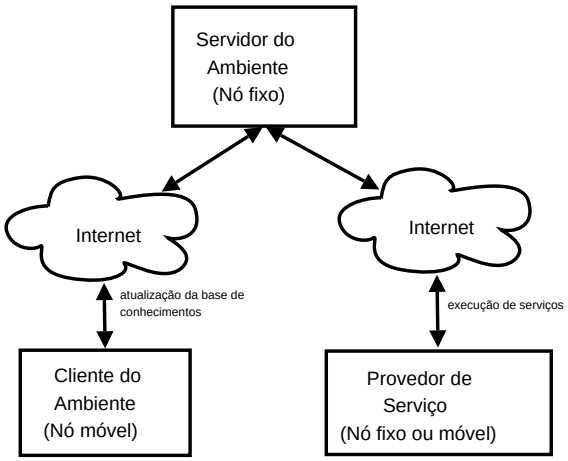

Figura 5.1: Componentes da arquitetura proposta.

A diferença desta arquitetura com a arquitetura para serviços web [51] é que a execução dos serviços é feita pelo servidor do ambiente e não pelos clientes diretamente. Assim, os clientes interagem somente com o servidor do ambiente. Isto permite ver o papel do servidor do ambiente como mediador entre os clientes e os fornecedores de serviços que permite a descoberta e execução automática de serviços.

A seguir fazemos uma breve descrição dos três componentes da arquitetura:

1. **Servidor do ambiente:** É a entidade central encarregada de manter os componentes associados aos dispositivos e fazer a descoberta e execução de serviços no ambiente. A interação dos dispositivos com o ambiente é feita através deste módulo que fornece suas funcionalidades através de um serviço web.

   Este componente se encarrega de gerenciar os dispositivos que se encontram no ambiente e é executado em um nó fixo da rede. Mantém para cada dispositivo no ambiente uma *base de conhecimento* e um *ATMS*. Os mecanismos de descoberta e execução de serviços estão contidos também neste componente.

2. **Fornecedores de serviços:** São os componentes da arquitetura que são executados em dispositivos móveis ou fixos e disponibilizam algum serviço no ambiente.

   Um serviço é fornecido através de serviços web. Para que o serviço esteja disponibilizado no ambiente uma descrição semântica em WSMO dele deve ser registrado no *repositório de serviços* do servidor do ambiente.

3. **Clientes do ambiente:** São executados em dispositivos móveis que são carregados pelos usuário que desejam interagir com o ambiente.

   Um cliente do ambiente é um cliente do serviço web fornecido pelo servidor do ambiente.

A seguir fazemos uma descrição dos principais módulos do servidor do ambiente, como se mostra na Figura 5.2.

5.1.1 Repositórios

A arquitetura possui três repositórios:

1. **Repositório de serviços:** Contém as especificações semânticas em WSMO dos serviços disponíveis no ambiente. O mecanismo de descoberta de serviços busca no repositório as especificações de serviços disponíveis para um determinado contexto. O módulo de descoberta baixa e interpreta um arquivo *wsml* com essa especificação. Este repositório é acessível através do *serviço UR* do WSAmI (Seção 3.6.1).

2. **Repositório de ontologias:** Mantém um repositório das ontologias empregadas na descrição dos serviços no ambiente. Na descoberta de serviços as ontologias necessárias para a inferência são procuradas nesse repositório. Este repositório contém arquivos *wsml* com a especificação das ontologias em WSMO.

3. **Repositório de código:** Mantém um repositório de classes Java geradas a partir da interface WSDL do serviço. Estas classes são carregadas em tempo de execução e são necessárias para permitir a execução automática dos serviços. O sistema emprega essas classes para gerar objetos Java que possam ser enviados aos serviços web. O sistema não emprega *stubs*[1] pré-gerados para a execução de serviços web,

[1] Um *stub* é um objeto proxy que pega uma chamada de método para um objeto normal e a converte a uma mensagem *SOAP* (no caso dos serviços web) que é enviada ao serviço web.

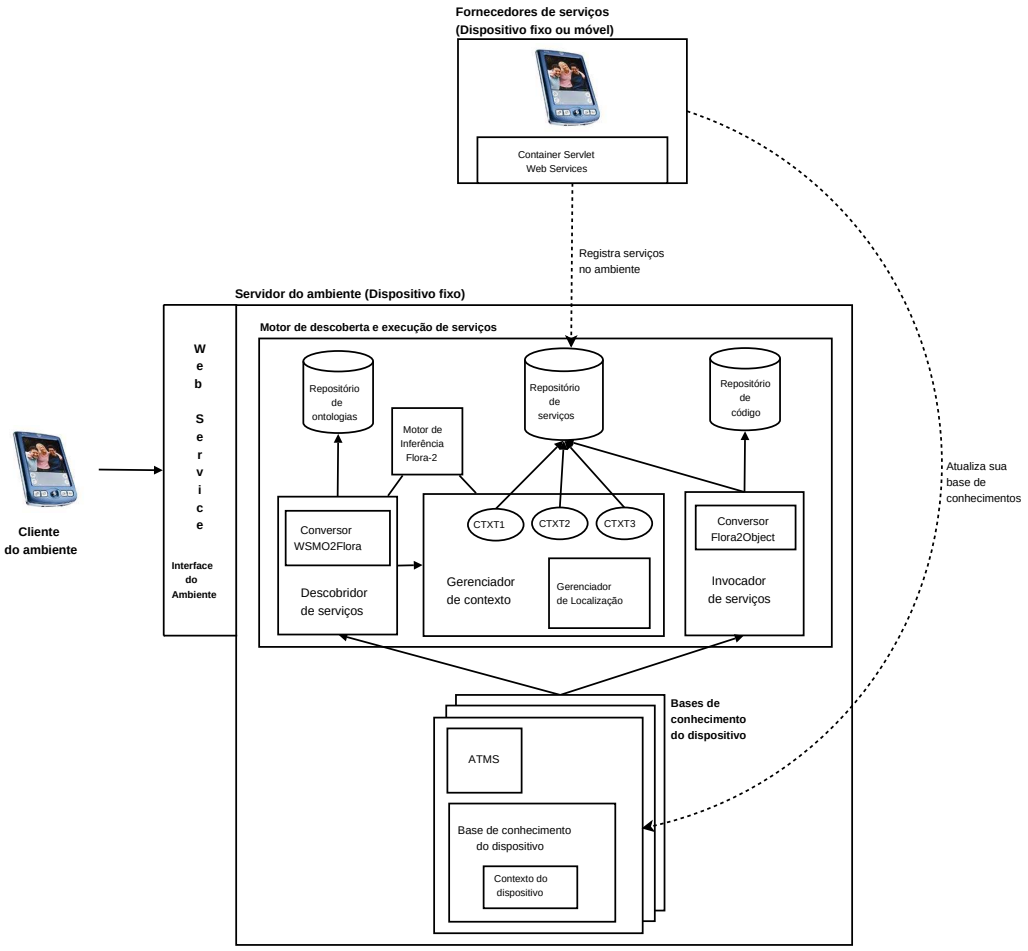

Figura 5.2: Arquitetura do modelo proposto.

mas precissa das classes que definem os tipos (seção `types`) da interface WSDL.

5.1.2 Gerenciador de contexto

Este módulo é o encarregado de fazer o gerenciamento dos serviços disponíveis no ambiente. Neste módulo são registradas associações entre descrições de contextos e serviços, formando-se pares do tipo:

$$< \{contex_1, contex_2, ...\}, \{serv_1, serv_2, ...\} >$$

Onde $contex_i$ é uma expressão em WSMO que define um contexto e $serv_i$ é um serviço no ambiente. Dessa maneira são determinados os serviços disponíveis para um dado contexto. Isto nos permite reduzir a amplitude de busca de serviços e também estabelecer restrições sobre a disponibilidade de serviços no ambiente. Por exemplo, para determinar que o serviço `Library` se encontra disponível somente para alunos do IME, registramos o seguinte par no *gerenciador de contexto*:

```
< { ?someStudent[hasDepartment hasValue loc#ime]
        memberOf fex#StudentFenix },
  {Library} >
```

Serviços que não estão associados a nenhum contexto são considerados que estão disponíveis para qualquer contexto. Com base nesses registros o gerenciador de contexto (através do módulo *inferência em Flora-2*) acha os serviços disponíveis para o contexto do dispositivo. Todos estes serviços, e também os serviços não associados a nenhum contexto, servem de entrada ao módulo de descoberta de serviços.

O gerenciador de contexto também é o encarregado de monitorar o contexto do dispositivo e atualizar a sua base de conhecimentos. Para isto, emprega a API do MoCA para atualizar a informação de contexto proveniente dos dispositivos direto na sua base de conhecimentos.

5.1.3 Descobridor de serviços

Este módulo é o encarregado da descoberta de serviços mais adequados para a tarefa determinada pelo usuário. Na Seção 5.2 se faz uma descrição mais detalhada do mecanismo de descoberta proposto.

Este módulo está relacionado diretamente com o *gerenciador de contexto* e a *base de conhecimento do dispositivo*. Mais especificamente, a descoberta

de serviços emprega o *contexto do dispositivo* (que faz parte da base de conhecimentos do dispositivo) para achar os serviços acessíveis para o contexto do dispositivo através do *gerenciado de contextos*. E emprega o *ATMS* para manter as relações entre hipóteses e efeitos e poder achar os métodos de serviços que podem ser executados para conseguir uma tarefa determinada.

5.1.4 Invocador de serviços

Este módulo é o encarregado da execução automática dos serviços descobertos pelo módulo de *descoberta de serviços*. A execução de serviços implementada usa a invocação por interface dinâmica (*Dynamic Invocation Interface* - DII) para *Jax-RPC*, permitindo a execução de serviços web sem empregar *stubs* pré-compilados, desde que novos serviços possam ser incorporados ao ambiente. Para este tipo de execução é necessário ter: (1) o ponto de execução do serviço (*end-point*), (2) o nome do método e (3) um arranjo dos objetos que são os parâmetros do método.

Por outro lado, o *invocador de serviços* precisa das classes que representam os tipos declarados na interface WSDL do serviço, as quais são passados como parâmetros aos serviços web. Esses objetos são obtidos a partir da *base de conhecimentos do dispositivo*.

O invocador de serviços também está encarregado de converter os resultados da execução de um serviço e colocá-los na *base de conhecimento* do dispositivo. Nas próximas seções é descrito detalhadamente o mecanismo para a execução automática de serviços.

5.1.5 Gerenciador de localização

Este componente é parte do *gerenciador de contextos* e emprega a API do MoCA para acessar os serviços fornecidos pelo MoCA. Assim, este componente permite consultar o serviço de localização (LIS) do MoCA para obter a região simbólica em que se encontra um dispositivo, conforme apresentado na Seção 3.5. Fazendo consultas a este serviço este módulo atualiza a localização do usuário direto na base de conhecimento do dispositivo.

5.1.6 Base de conhecimento

Este componente gerencia toda informação que um dispositivo contém . Nesta base de conhecimento é armazenado também o contexto do dispositivo, que é conhecimento que muda constantemente como a localização do usuário.

A base de conhecimento do dispositivo foi implementada para ter entradas do tipo:

$$< (conceito), \{objeto_1, objeto_2, ...\} >$$

onde *conceito* é uma expressão em WSMO e $objeto_i$ são objetos Java que representam (reificam) esse conceito. Por exemplo:

```
< (?someStudent[hasDepartment hasValue loc#ime]
                        memberOf fex#StudentFenix),
                        {Student@1232fasf, Student@3g3122} >
```

onde `Student@1232fasf` e `Student@3g3122` são instâncias Java da classe `Student`. A conversão de objetos Java em sentenças F-Logic é feita empregando reflexão sobre as classes e objetos através do componente *Flora2Object*. Esta conversão é necessária para permitir ao descobridor de serviços montar a teoria lógica e fazer a inferência.

Desta maneira se torna fácil manter relações na base de conhecimentos entre conceitos definidos nas ontologias e objetos Java para esses conceitos.

5.1.7 Processo de descoberta e execução

A seguir apresentamos a seqüência normal de passos no processo de descoberta e execução de serviços:

Passo 1 - Registro: O dispositivo é registrado no ambiente. Neste primeiro passo o ambiente cria uma base de conhecimento, um ATMS e um contexto para o dispositivo.

Passo 2 - Especificação da tarefa: O usuário envia a descrição da tarefa que quer executar. A especificação desta tarefa pode ser feita de maneira automática ou manual. No primeiro caso, o encarregado pode ser um agente inteligente instalado no dispositivo do usuário que reconhece as necessidades do usuário e, baseado em toda a informação que puder coletar, determina a tarefa pelo usuário. Já no segundo caso o usuário define explicitamente a tarefa que quer desenvolver no ambiente. Isto implica que o usuário conheça os conceitos das ontologias envolvidas na especificação dos serviços no ambiente.

Passo 3 - Disponibilidade de serviços: O gerenciador de contexto acha o conjunto de serviços disponíveis para o contexto do dispositivo.

Passo 4 - Descoberta e montagem do ATMS: O componente encarregado da descoberta de serviços localiza os serviços e os métodos destes

serviços que atendem melhor a tarefa do usuário (este procedimento é detalhado na seção seguinte). Os serviços descobertos são empregados para montar o ATMS do dispositivo. A partir de consultas no ATMS pode ser determinado os métodos de serviços a serem executados.

Passo 5a - Execução dos serviços: Os métodos descobertos são executados pelo *invocador de serviços*. Os objetos que são necessários como parâmetros para os métodos são obtidos a partir da base de conhecimento do dispositivo. Os resultados da execução dos métodos são adicionados à base de conhecimento do dispositivo.

Passo 5b - Ajuda pró-ativa: Se não existir nenhum método de serviço a ser executado, são enviados ao usuário os serviços mais próximos como ajuda pró-ativa e ser completadas pelos usuários.

Passo 5c - Múltiplos objetos: Pode acontecer que se tenha vários objetos que podem ser passados como parâmetro a um método. O usuário é o encarregado de escolher qual objeto será utilizado para a execução.

Existem dois processos executados constantemente, que mantêm atualizados (1) o contexto e (2) o ATMS do dispositivo.

1. **Atualização do contexto:** O contexto de cada dispositivo é atualizado pelos gerenciadores de contexto e de localização, que após intervalos determinados capturam informação do dispositivo (através dos serviços CIS e LIS do MoCA, Seção 3.5) e atualizam a sua base de conhecimentos.

2. **Atualização do ATMS:** O ATMS do dispositivo é atualizado segundo a disponibilidade dos serviços. É necessário detectar se os serviços introduzidos anteriormente no ATMS ainda estão disponíveis no ambiente. A descoberta de serviços é feita constantemente, com base na tarefa inicial definida pelo usuário. Serviços já adicionados ao ATMS são descartados, e serviços que deixaram de funcionar são removidos.

5.2 Descoberta de serviços

O processo de descoberta de serviços é feito em duas fases: (1) a descoberta de serviços propriamente dita, quando é empregada a descrição das IOPEs do serviço (Seção 4.2.2) e (2) busca de *environments* válidos do ATMS. O ATMS é montado a partir das regras de transição (Seção 4.2.2) dos serviços descobertos pela primeira fase.

5.2.1 Mecanismo de descoberta

A descoberta de serviços é feita com base na descrição das capacidades dos serviços especificadas na classe `Capability` definida pela ontologia WSMO. Este elemento descreve as pré-condições, hipóteses, pós-condições e efeitos (IOPEs) do serviço. Para determinar se um serviço pode satisfazer uma meta especificada, descrevemos a especificação do serviço e as ontologias envolvidas como uma teoria em F-Logic. Se a teoria puder ser satisfeita, então o serviço é selecionado para a fase seguinte.

Em [25] é apresentado um arcabouço lógico para a descoberta de serviços web em F-Logic. Nesse artigo são propostos e avaliados um conjunto de regras lógicas para a descoberta de serviços para uma meta dada.

A partir das idéias apresentadas nesse artigo é empregada a seguinte condição para a descoberta de serviços:

$$\exists Ser \bullet O, Ser_{pos}, KB_{disp} \models G \wedge Ser_{hip} \tag{5.1}$$

Nesta expressão Ser representa um serviço, O é o conjunto de ontologias empregadas na descrição do serviço, Ser_{pos} são as pós-condições do serviço, KB_{disp} é o conjunto de fatos na base de conhecimento do dispositivo, G é a meta a ser atingida e Ser_{hip} são as hipóteses do serviço.

Esta condição indica que para uma dada meta G existe um serviço Ser tal que, a partir do conjunto de ontologias (O), as pós-condições do serviço (Ser_{pos}) e a base de conhecimento do dispositivo (KB_{disp}) é possível ter como conseqüência lógica a meta desejada (G) e as hipóteses do serviço (Ser_{hip}).

A Figura 5.3 mostra a teoria criada para o caso do serviço `Library` (`librarywservice:webservice`). As pós-condições do serviço são representados como regras a serem inseridas (notação `${...}`) na teoria em tempo de execução através da sentença `insertrule{ Post }`.

A representação das pós-condições em F-Logic é feita como cláusulas de Horn, onde as hipóteses do serviço implicam as pós-condições do serviço. Assim, uma meta pode disparar alguma regra que para ser satisfeita requer que as hipóteses sejam satisfeitas também. E para se conseguir isto, todas as hipóteses (corpo da regra) devem estar na base de conhecimento do dispositivo. Para isso, os fatos da base de conhecimento do dispositivo são adicionados também à teoria (linha 26 da Figura 5.3).

Toda a inferência para a descoberta é feita com base nas ontologias empregadas. No apêndice A se mostra a teoria completa em Flora-2 para o serviço `Library` empregado no exemplo.

```
1   librarywservice:webservice[
2   postcond ->> ${_listbooks:relatedbooks :-
3                  ((_student[haslocation->ime] and
4                    _student:studentfenix))},
5   postcond ->> ${_listbooks:coursebooks :-
6                  ((_student[haslocation->ime] and
7                    _student:studentfenix))},
8   assumptions ->> ${(_student[haslocation->ime] and
9                      _student:studentfenix)},
10  result  ->> ${results:result[service ->> librarywservice]}
11  ].
12
13  postcond_main(X) :- (_listbooks:relatedbooks);
14                      ((_listbooks:coursebooks)).
15  goalRule(Goal) :- Goal:goal[query -> Q],Q.
16
17  /* discovery rules */
18  discovery(Service:webservice,Goal:goal) :-
19     Service[postcond ->> Post, result ->> Result,
20             assumptions ->> Assumptions],
21     insertrule{ Post },
22     if goalRule(Goal) then
22        insert{ _:result[service ->> Service,goals ->> Goal]},
23     deleterule{ Effects }.
24
25  /* knowledge base facts */
26  crhistian:studentfenix[haslocation -> ime].
```

Figura 5.3: Regra de descoberta de serviços para o serviço `Library`.

O processo de inferência para a descoberta de serviços é iniciada com `discovery (Services, Goals)` (linha 18) instanciando as variáveis `Services` e `Goals` com todos os serviços (elementos `webservice`) e metas (elementos `goal`) respectivamente, contidas na teoria.

Na linha 21 inserimos as regras que representam as pós-condições do serviço, logo, na linha 22 e avaliada a sentença `goalRule(Goal)` que consulta se a meta pode ser atingida para as pós-condições inseridas do serviço. Se a meta pode ser satisfeita é inserida a molécula

```
_:result[service->>Service,goals->>Goal]
```

que associa a um serviço as metas que pode satisfazer. Para obter todas as relações fazemos a consulta

```
?-_:result[service->>Service,goals->>Goal]
```

instanciando as variáveis `Service` e `Goal`.

Na linha 15 `Q` representa a expressão que se quer conseguir. O conceito `goal` tem a forma `goal[query => $]` e as consultas por tarefas são feitas empregando este conceito. Por exemplo, em `goal1:goal[query => ${Relatedbooks:listbooks}]`, `Q` representaria a tarefa `Relatedbooks:listbooks` e é feita a consulta nesta molécula.

5.2.2 Montagem do ATMS

Na Seção 2.5 foram apresentados os princípios e funcionamento do ATMS. Nesta seção apresentamos como são introduzidas no ATMS as regras de transição dos serviços descobertos pelo mecanismo apresentado na seção anterior.

O objetivo principal do ATMS é manter as relações entre o conjunto de *fatos consistentes* que produzem certo efeito. Permitindo consultar pelos fatos necessários (condições minimais) para obter um determinado efeito. Para montar o ATMS são empregadas as regras de transição que definem o comportamento do serviço conforme apresentado na Seção 4.2.2. Assim, uma vez descoberto um serviço na fase anterior por cada regra de transição no serviço é inserido dois tipos de nós: (1) justificativa, que representa a regra de transição e (2) hipóteses, que representam as condições da regra de transição. A justificativa propagada tem a forma:

$$< efeito_i, \{\{cond_1, cond_2, ...\}, ...\}, \{\{cond_1, cond_2, ...\} \Rightarrow efeito_i, ...\} >$$

Onde $efeito_i$ e $cond_i$ são os efeitos e condições que se encontram em uma regra de transição.

Note-se que vários conjuntos de condições podem produzir um efeito, uma vez que podem existir várias regras de transição que possuem esse efeito como conseqüência. Por exemplo, considere a regra de transição apresentada na Figura 4.11:

```
if(?someStudent[hasLocation hasValue ?someLocation]
      memberOf fex#StudentFenix) then
add(?relatedbooks memberOf lib#RelatedBooks)
```

para essa regra teríamos a seguinte justificativa propagada no ATMS:

```
< ?relatedbooks memberOf lib#RelatedBooks,
  { {?someStudent[hasLocation hasValue ?someLocation]
        memberOf fex#StudentFenix}},
  { ?someStudent[hasLocation hasValue ?someLocation]
        memberOf fex#StudentFenix}
    => ?relatedbooks memberOf lib#RelatedBooks >
```

É necessário também inserir as condições das regras de transição como hipóteses ao ATMS, para que consiga achar o conjunto de *environments* para os nós do ATMS baseado nessas hipóteses. Essas hipóteses tem a forma:

$$< cond_i, \{cond_i\}, \{\{cond_i\} \Rightarrow cond_i\} >$$

onde $cond_i$ são as condições das regras de transição.

O ATMS etiqueta cada efeito $efeito_i$, que foi propagado nas justificativas, com um rótulo L que indica o conjunto de *environments* do efeito. Para cada *environment* E_i que pertence a L, $efeito_i$ é conseqüência lógica de E_i no contexto do conjunto de ontologias envolvidas (O) e o conjunto de justificativas inseridas por algum serviço (RT_{Ser_i}):

$$O, E_i, RT_{Ser_i} \models efeito_i \tag{5.2}$$

Cada hipótese em E_i é associada a uma justificativa, que por sua vez está associada a uma regra de transição e portanto a um método concreto na interface WSDL do serviço. Conforme foi descrito na Seção 4.2.2, cada regra de transição está associada a um método da interface WSDL. Com esta associação um nó do ATMS tem a forma:

$$< efeito, \{\{(h_1^i, metodo_1^i, prop_i), (h_2^j, metodo_2^j, prop_j)\}\} >$$

onde todo h_j^i é uma hipóteses que pertence ao *environment* E_i. $metodo_j^i$ é o nome do método associado e $prop_j$ é um inteiro que indica a profundidade da *expansão*. A idéia de *expansão* e o uso do parâmetro $prop_j$ serão explicados na próxima seção.

Fundamentação com a base de conhecimento do dispositivo

O ATMS determina o conjunto de *enviroments* (*label* L) minimais que produzem um efeito $efeito_i$. Cada um desses conjuntos é o conjunto de pré-condições mínimas, consistentes e completas que suporta um efeito. Assim,

para determinar os métodos dos serviços a serem executados determinamos quais *environments* do efeito desejado são conseqüência lógica da base de conhecimentos do dispositivo ($KB_{dispositivo}$).

Representamos a base de conhecimentos do dispositivo como uma teoria lógica em F-Logic e conforme o conjunto de ontologias empregadas no serviço (O) satisfazemos a seguinte prova para o *environment* E_i:

$$\begin{aligned} &O, KB_{dispositivo} \models h_i \\ &\forall h_i \in E_i \end{aligned} \tag{5.3}$$

Onde h_i representa uma hipóteses que pertence a E_i. Os métodos associados às hipóteses dos *environments* satisfeitos pela base de conhecimentos serão executados. Aqueles métodos associados a *environments* que não possam ser completamente satisfeitos pela base de conhecimentos são fornecidos ao usuário como *ajuda pró-ativa*. Neste último caso o usuário tentara satisfazer ele mesmo aquele *environment* para conseguir o efeito desejado fornecendo os objetos faltantes ou modificando os existentes. Assim, para um dado efeito se acha o conjunto de *environments* dos quais pode-se derivar essa meta.

Pode acontecer que um *environment* satisfeito possua mais de um método associado, um por cada hipótese contida. Neste caso emprega-se o valor *prop* associado a cada hipótese e é devolvido o método com o maior número *prop*. Isto indica que é necessário primeiro executar este método já que é necessário ter seus efeitos para satisfazer o *environment*. Na Figura 5.4, isto indicaria primeiro executar o método da hipótese A, antes dos métodos das hipóteses X ou Y.

Expansão para sub-metas

Pode ocorrer o problema de não se achar um conjunto de *environments* que possam ser cobertos pela base de conhecimentos. Para as hipóteses dos *environments* é aplicado o processo de descoberta de serviços e suas regras de transição são propagadas no ATMS. Assim, novos conjuntos de hipóteses são associados às sub-metas. Em cada expansão o valor *prop* de cada nova hipóteses propagada é incrementado.

Cada *environment* pode ser visto como o conjunto de *sub-metas* necessárias para conseguir a meta final. O processo de descoberta é repetido para descobrir um conjunto de *environments* para essas sub-metas. Pelos algoritmos de propagação e atualização do ATMS os conjuntos de hipóteses achados

para as sub-metas fazem parte do rótulo (conjunto de hipóteses) do efeito inicial.

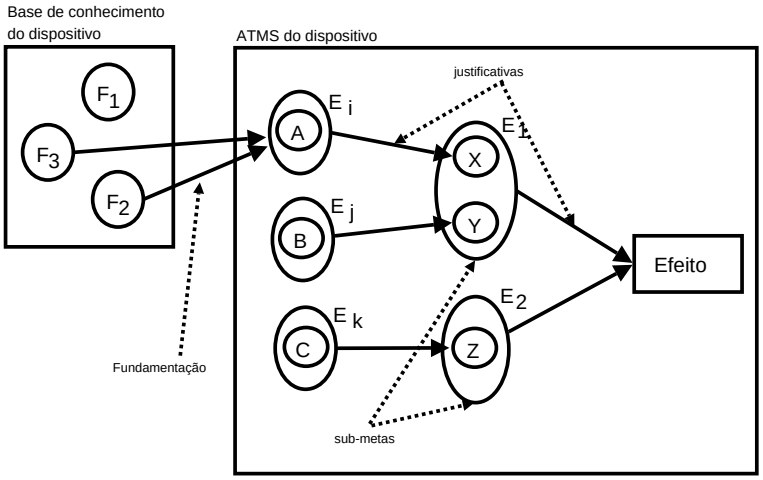

Figura 5.4: Fundamentação do ATMS pela base de conhecimentos do dispositivo.

Na Figura 5.4 se apresentam estas idéias. Os *environment* E_1 e E_2 foram achados para o efeito inicial. O estado do ATMS nesse momento é:

$$< efeito, \{\{X, Y\}, \{Z\}\} >$$

A seguir, são achadas justificativas para X, Y e Z dos *environments* E_1 e E_2. Para isto, X, Y e Z se convertem em sub-metas para as quais novas justificativas são propagadas. As novas justificativas acham os *environments* E_i para X, E_j para Y e E_k para Z. A propagação destas justificativas afeta também os *environments* do efeito inicial. Neste ponto o estado do ATMS é:

$$< efeito, \{\{X, Y\}, \{X, B\}, \{Y, A\}, \{A, B\}, \{Z\}, \{C\}\} >$$

A fundamentação com a base de conhecimento do dispositivo procura entre estes *environments*. Assim, o *environment* $\{A, B\}$ permite procurar por novos fatos que antes não podiam ser satisfeitos pela base de conhecimentos (X e Y).

Todas as hipóteses que pertencem a um *environment* deveriam ser satisfeitas por elementos da base de conhecimento, que são conjuntos *minimais*

que produzem certo efeito. Esses conjuntos são condições necessárias mínimas para a criação do efeito.

A Figura 5.5 mostra o exemplo de uma relação entre as hipóteses do ATMS e os efeitos com sentenças WSMO. Ao propagar as seguintes justificativas no ATMS,

```
?student[hasLocation hasValue ?somearea] memberOf StudentFenix
       =>  ?relatedbooks memberOf RelatedBooks}
```

e

```
?book[hasIsbn hasValue ?someisbn] memberOf Book
       =>  ?relatedbooks memberOf RelatedBooks}
```

o ATMS acha dois *environments* na etiqueta `{{?student[hasLocation hasValue ?somearea] memberOf StudentFenix}, {?book[hasIsbn hasValue ?someisbn] memberOf Book}}` para a molécula `?relatedbooks memberOf RelatedBooks`. Assim, o mecanismo de descoberta pode cobrir um dos dois *environments* com a base de conhecimentos do dispositivo.

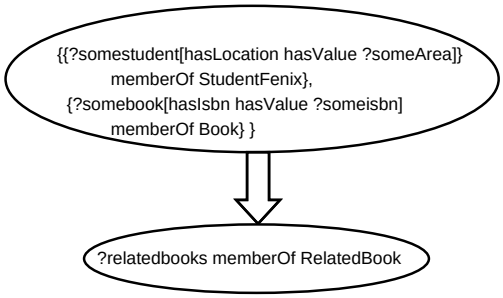

Figura 5.5: Relação causa-efeito do ATMS.

Orientação pró-ativa

Depois de ser feita a expansão de todas as possíveis regras de transição, e não for possível achar *environments* válidos. Os *environments* mais próximos que podem ser fundamentados são enviados ao usuário para completar as condições necessárias para a execução do método associado.

Atualização do ATMS

É necessário atualizar o ATMS perante mudanças na disponibilidade dos serviços. Se novos serviços ficam disponíveis ao dispositivo, o mecanismo de descoberta é iniciado. Se algum serviço deixa de estar disponível para o dispositivo, os seus efeitos deixam de ser válidos e são eliminados os efeitos e as hipóteses inseridas no ATMS das regras de transição do serviço.

5.2.3 Funções do ATMS

O uso do ATMS no mecanismo de descoberta nos permite implementar as funcionalidades requeridas pelo modelo computacional definido na Seção 2.4 que são apresentadas a seguir:

Manter o contexto computacional do dispositivo: O ATMS se converte na representação dos serviços que o usuário descobre e que o podem ajudar a atingir uma tarefa.

O ATMS e a base de conhecimento do dispositivo juntos têm as mesmas funcionalidades que o componente *Aura*(*Prism*) na arquitetura proposta pela arquitetura do projeto Aura (Seção 3.3).

A base de conhecimento do dispositivo mantém os fatos que são conseqüência da execução de serviços. Com esses fatos é possível achar novos serviços a executar, que podem por sua vez agregar novos fatos à base de conhecimentos.

Descoberta e configuração de serviços: Pelo mecanismo de descoberta é possível achar automaticamente serviços em novos ambientes e executá-los com base no conhecimento que possui o dispositivo do ambiente. O ATMS é atualizado freqüentemente segundo a disponibilidade dos serviços, mantendo o estado do ambiente para o dispositivo. A base de conhecimento do dispositivo também é atualizada com informação contextual do dispositivo e os resultados da execução de serviços. Com estes dois elementos atualizados é possível achar os parâmetros necessários para a execução dos métodos. Estas idéias se apresentam na Figura 5.6.

Orientação pró-ativa: A detecção dos *environments* mais próximos que podem ser cobertos pela base de conhecimento permite ao sistema achar os métodos mais próximos que podem ser executados. Com base nesses *environments* pode-se inferir algum "conselho" para o usuário. Por exemplo, supondo que temos o *environment* $E_1 = \{_ :$

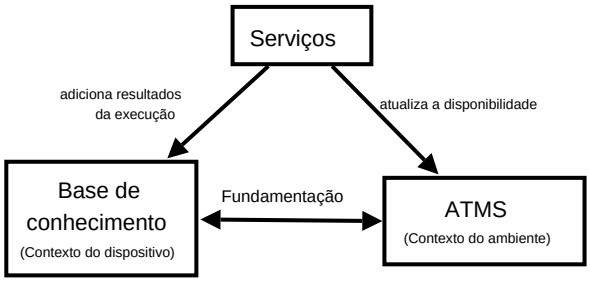

Figura 5.6: Atualização da base de conhecimentos e o ATMS

$student[haslocation- > area1]\}$ e na base de conhecimentos do dispositivo se tenha $joao : student[\ haslocation- > area2]$, para poder satisfazer o *environment* E_1 é recomendado ao usuário se movimentar para a $area1$.

5.3 Execução dos serviços

A execução dos métodos de serviços descobertos na fase anterior é feita pelo *ambiente de execução* que controla a execução de cada método. Além disso, este ambiente de execução encontra os parâmetros necessários para a exe-cução dos métodos na base de conhecimento e insere os resultados da execução na mesma base de conhecimento.

Este ambiente de execução executa os métodos segundo propriedades especificadas na interface semântica do serviço. Estas propriedades determinam a freqüência de execução e se os resultados obtidos serão ou não inseridos na base de conhecimento do dispositivo.

5.3.1 Propriedades de execução

A execução dos serviços é feita através do *invocador de serviços*, o qual fornece a implementação do ambiente para a execução de serviços. Para a automatização da execução dos serviços é associado a cada método do serviço o tipo de resultado que devolve e as vezes em que o método deve ser executado. São atribuídas a cada método do serviço estas propriedades de execução, o que é feito nas propriedades não funcionais da `Choreography` do serviço.

Assim, os resultados da execução de um método de serviço podem ser:

Persistentes (persistent): Os resultados são adicionados à base de conhecimento do dispositivo.

Transientes (transient): Os resultados não são adicionados à base de conhecimento do dispositivos e são disponibilizados temporariamente para o cliente.

O tipo de freqüência de execução pode ser:

Várias vezes (many): O método do serviço é executado repetidas vezes com a freqüência de tempo determinada no *Gerenciador de Execução*.

Uma vez (one): O método do serviço é executado uma vez só.

	Resultado Persistente	**Resultado Transiente**
Executar várias vezes	persistent-many	transient-many
Executar uma vez	persistent-one	transient-one

Tabela 5.1: Propriedades de execução para os métodos de serviços.

A Tabela 5.1 apresenta as possíveis configurações das propriedades associadas a um método. A Figura 5.7 apresenta um exemplo da associação destas propriedades aos métodos do serviço `Library`.

```
interface LibraryInterface
   choreography _"urn:LibraryChoreography"
   nonFunctionalProperties
      someTheme\-disciplinebooks hasValue
                                 "findBookByTheme:persistent-one"
      someISBN\-relatedbooks hasValue "relatedBooks:persistent-one"
      someBook\-someBook hasValue "takeBook:transient-one"
   endNonFunctionalProperties
```

Figura 5.7: Interface semântica do serviço `Library` com propriedades de execução

5.3.2 Criação de objetos Java

As condições das regras de transição do serviço indicam os conceitos requeridos pelo método do serviço para a execução. Objetos Java que representam esses conceitos são tomados da base de conhecimento do dispositivo.

Assim, pode acontecer que existam vários objetos possíveis para reificar um conceito. Por exemplo, o método que devolve uma lista de livros relacionados recebe um livro só, e o usuário deveria indicar um livro em particular se tiver vários livros na base de conhecimento. Neste caso é necessário que o usuário escolha o objeto com o qual executar o método.

A execução de um serviço web requer objetos Java para sua execução. Para a automatização da execução devemos achar esses objetos segundo a especificação semântica do método que queremos executar. Por exemplo, a partir da seguinte regra de transição.

```
if(?someStudent[hasLocation hasValue ?someLocation]
      memberOf fex#StudentFenix) then
add(?relatedbooks memberOf lib#RelatedBooks)
```

Procuramos na base de conhecimento do dispositivos por uma instância que represente a sentença `?someStudent[hasLocation hasValue ?someLocation] memberOf fexStudentFenix`. Esta sentença indica que precisamos somente do atributo `location` de alguma instância que represente o conceito `StudentFenix`. Uma vez que na base de conhecimento do dispositivo temos associadas sentenças WSMO e objetos Java, é fácil achar os objetos necessários e extrair o objeto que representa o atributo necessário, que no exemplo seria o atributo `location`.

Note-se que a partir da relações estabelecidas entre conceitos de ontologias e classes Java feitas na especificação das ontologias do serviço (como se mostrou na Seção 4.2.1) é possível achar a classe Java que representa o conceito da ontologia.

Os objetos obtidos da execução do método são inseridos na base de conhecimento. A partir da regra de transição é possível determinar a sentença WSMO associada ao objeto resultado da execução para poder ser inserida na base de conhecimento. No exemplo, os objetos da execução são associadas à sentença `?relatedbooks memberOf RelatedBooks`.

5.4 Serviços da arquitetura

A arquitetura proposta fornece serviços com funcionalidades específicas para o ambiente que são chamados de *serviços da arquitetura*. Estes serviços fornecem acesso a componentes da própria arquitetura como bases de conhecimento de outros dispositivos, o gerenciador de localização e o gerenciador de contexto.

Estes serviços podem ser descobertos e executados automaticamente como qualquer outro serviços que faz parte do ambiente. Os serviços da arquitetura implementados são:

Serviço de proximidade: Este serviço recebe o identificador de um dispositivo no ambiente e retorna uma lista de outros dispositivos que se encontram na sua proximidade.

A implementação deste serviço tem acesso direto às bases de conhecimento dos dispositivos registrados no ambiente para achar sua localização e devolver uma lista de dispositivos próximos ao dispositivo que enviou a consulta.

As especificações semânticas dos métodos deste serviço são:

- Método `getNeighborsDevice`:

```
if(?someDevice[hasMac hasValue ?someMac]
      memberOf Device)
   then
(?realtedDevice memberOf ListDevice)
```

- Método `getNeighborsUsers`:

```
if(?someDevice[hasMac hasValue ?someMac]
      memberOf Device)
   then
(?relatedusers memberOf ListUser)
```

- Método `getNeighborsUsers`:

```
if(?someuser[hasLocation hasValue ?somearea]
      memberOf User)
   then
(?relatedusers memberOf ListUser)
```

Serviço disparador de métodos: Este serviço mantém associações entre contextos e métodos de serviços a serem executados quando o contexto de um dispositivo calçar com o contexto associado.

Neste serviço são registrados contextos e um ou mais métodos de serviços que serão executados cada vez que o contexto de algum dispositivo estiver de acordo com o contexto registrado.

- Método `executeMethod`:

```
if(?context[hasElements hasValue ?somelist]
      memberOf Context)
   then (?someResult memberOf Result)
```

Por exemplo, a entrada registrada no serviço

```
< context1[hasElemens hasValue{?someDevice[
                              hasLocation hasValue ime]
                                   memberOf Device}]
    memberOf Context] ,

  welcomeService[hasId hasValue Welcome,
                 hasMethods hasValue {greetings}]
    memberOf Service >
```

executaria o método `greetings` do serviço `Welcome` que mostra saudações ao todos os dispositivos que se encontram no `ime`.

Serviço de introspecção do ambiente: Este serviço devolve informação contextual de um dispositivo no ambiente.

Este serviço acessa diretamente as bases de conhecimento dos usuários registrados no ambiente e retorna toda a informação contextual desse dispositivo.

- Método `getContext`:

```
if(?someDevice memberOf Device)
   then (?somecontext memberOf Context)
```

Serviço de localização e orientação: Este serviço permite determinar a localização de qualquer objeto no ambiente físico e o endereço do objeto com relação ao dispositivo no ambiente que executa o serviço.

A implementação desde serviço teria que ter algum mecanismo para determinar a localização de um objeto a partir do seu identificador. Um mecanismo de RFID ou consultas simples a um banco de dados serviriam para a implementação deste serviço.

Este serviço tem os seguintes métodos e suas definições semânticas:

- Método `getLocation`:

```
if(?someThing[hasId hasValue ?someId] memberOf Thing)
   then
(?someposition memberOf Position)
```

- Método `getDirection`:

```
if(?someThing[hasId hasValue ?someId] memberOf Thing  and
    ?someDevice memberOf Device)
   then
(?somedirection memberOf Direction)
```

Serviço de armazenamento: Este serviço fornece um mecanismo de persistência para usuários no ambiente. Este serviço mantem um repositório de arquivos por cada dispositivo registrado no ambiente. Este serviço pode ser acessado remotamente através da web ativando a ubiqüidade da informação.

- Método `addFile`:

```
if(?someUser[hasId hasValue ?someId] memberOf User
   and ?someFile[hasName hasValue ?someName,
                 hasType hasValue ?someType,...]
       memberOf BlobFile
)
   then (?addFileConfirmation memberOf AddFileConfirmation)
```

- Método `getFile`:

```
if(?someUser[hasId hasValue ?someId] memberOf User
   and ?someFile[hasName hasValue ?someName]
       memberOf BlobFile
)
   then
(?someFile memberOf BlobFile)
```

Capítulo 6

Implementação do Modelo e Experimentos

Neste capítulo apresentamos a implementação do modelo proposto no Capítulo 5. É apresentada também a implementação de um ambiente inteligente para uma biblioteca como prova de conceito do modelo proposto. A ativação do ambiente inteligente para a biblioteca empregando o modelo proposto exige a definição de ontologias, descrição semântica e implementação de serviços que forneçam a informação e funcionalidade necessária sobre livros e sua localização.

Como parte do modelo proposto foram implementados: (1) o servidor do ambiente e (2) um cliente genérico do ambiente. O servidor do ambiente é a implementação do modelo descrito no Capítulo 5 e o cliente foi implementado para ser executado em um PDA, permitindo a mobilidade do usuário. Como a tecnologia de comunicação são os serviços web, é possível implementar o cliente e o servidor em diferentes plataformas. Assim, o cliente foi implementado com a API de serviços web da plataforma `.NET` e o servidor do ambiente com a API para serviços web da plataforma `J2EE`.

Neste capítulo se detalham aspectos da implementação como diagramas UML, interfaces WSDL, objetos CORBA definidos, servidores e arcabouços empregados.

6.1 Implementação do servidor do ambiente

Esta seção descreve a implementação do *servidor do ambiente* que é parte da arquitetura apresentada na Seção 5.2. Este componente é executado num nó fixo da rede onde todos os clientes no ambiente estabelecem uma conexão.

Assim, se o endereço IP deste nó muda, ele teria que ser atualizado em todos os clientes.

A comunicação com o servidor do ambiente é através de um serviço web. Foi empregado o servidor de aplicações `JBoss` como contêiner deste serviço web, e foi empregado CORBA para o gerenciamento de sessões com os clientes. Isto facilita o gerenciamento de sessões desde que um serviço web não mantém estado com os clientes.

6.1.1 Projeto do servidor do ambiente

A Figura 6.1 mostra as classes mais relevantes e as relações consideradas na implementação do servidor do ambiente. A seguir fazemos uma breve descrição das funcionalidades destas classes.

Motor do dispositivo: Esta classe é encarregada de encapsular todos os objetos e métodos necessários para o mecanismo de descoberta e execução de serviços. Cada instância desta classe contem um ATMS e uma base de conhecimentos associados a um dispositivo. Conforme se apresentará as seções seguintes esta classe é a implementação de um objeto CORBA encarregado de manter as sessões com os clientes.

Host: Mantém informação física de um dispositivo móvel no ambiente. A informação de contexto recolhida pelo MoCA é atualizada nesta classe.

Base de conhecimentos do dispositivo: Mantém o registro dos fatos do dispositivo, conforme descrito na Seção 5.1.6.

ATMS do dispositivo: Esta classe representa a implementação do ATMS do dispositivo; é composta por várias instâncias da classe `Nó` que por sua vez contêm instâncias de `Justificativa` e `Environment`.

Serviço: Esta classe armazena toda a informação de um serviço no ambiente. Esta classe contém: (1) o ponto de execução (*end-point*) do serviço web, (2) uma instância do modelo WSMO que descreve o serviço, (3) a URL onde se encontra o código das classes do serviços e (4) um identificador do serviço.

Motor Flora: Esta classe é encarregada de fazer inferências em F-Logic. Emprega a API do Flora2 para compilar, carregar e consultar teorias lógicas em F-Logic criadas ou convertidas por `WSMO2Flora` ou pelo mecanismo de descoberta. A classe `QuickInference` representa uma versão rápida de consultas de sentenças F-Logic.

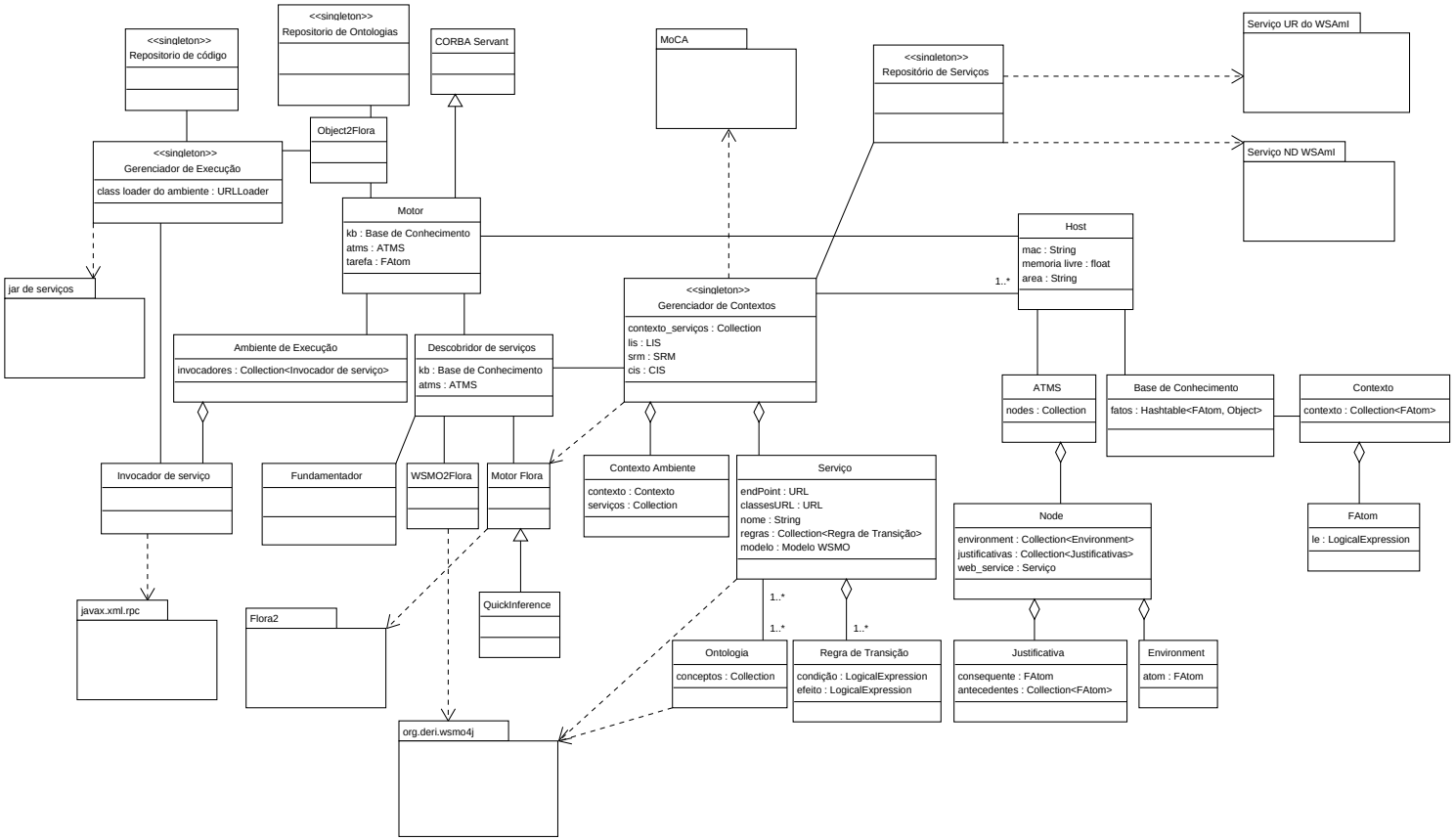

Figura 6.1: Diagrama de classes do servidor do ambiente.

WSMO2Flora: Converte uma especificação de serviço em WSMO para F-Logic que pode ser processada por Flora2.

Repositório de serviços: Esta classe se baseia nos serviços *UR* e *ND* do WSAmI. Esta classe faz consultas por serviços registrados no repositório universal do WSAmI (*UR*) e com isso consultar no serviço *ND* por instâncias de serviços e suas especificações semânticas em WSMO.

Gerenciador de contexto: Esta classe mantém uma tabela de serviços associados a contextos e emprega a base de conhecimentos e o motor Flora2 para achar os serviços disponíveis para o contexto do dispositivo. Esta classe mantém atualizadas as base de conhecimento dos dispositivos com informação contextual recoletada através do API do MoCA.

Descobridor de serviços: Abstrai todo o mecanismo para a descoberta de serviços apresentado na Seção 5.2. Para a primeira fase da descoberta se emprega a classe `WSMO2Flora` para converter as especificações WSMO dos serviços achados pelo gerenciador de contexto a Flora2. Para que logo o `Motor Flora` faça inferência sobre esta teoria para localizar o serviço mais adequado para o usuário. Depois, é montado o ATMS e a busca de *environments* satisfazíveis é delegada à classe `Fundamentador`.

Fundamentador: Esta classe se encarrega da fundamentação dos *environments* do ATMS pela base de conhecimento do dispositivo. Este componente obtém os *environments* da tarefa especificada encontrados pelo ATMS e devolve os métodos de serviços a serem executados associados aos *environments* satisfeitos.

Gerenciador de execução: Esta classe carrega em um só *Class Loader*[1] todas as classes Java que se encontram no repositório de código do ambiente. Nesta classe é mantida também, as relações entre conceitos de ontologias e classes Java. Desta maneira a classe `Object2Flora` pode criar objetos Java a partir de sentenças WSMO ou Flora.

Ambiente de execução: Esta classe se encarrega de criar invocadores de serviços para cada método de serviço a ser executado. A maneira em que serão executados os invocadores de serviços é segundo a descrição da Seção 5.3.

[1]Sucintamente, é a classe Java encarregada de carregar dinamicamente classes Java para disponibilizá-las à maquina virtual Java.

Invocador de serviços: Se encarrega de achar os parâmetros para a execução dos métodos do serviço. Faz consultas na base de conhecimentos do dispositivo por objetos Java que estejam associados às expressões da descrição semântica do método que será executado. Mais especificamente, procura objetos Java para as condições das regras de transição que descrevem o método.

Além disso, esta classe adiciona ou não (dependendo se o resultado foi declarado transiente ou persistente) os resultados da execução à base de conhecimento do dispositivo .

Conversor Object2Flora: Converte sentenças escritas em F-Logic a objetos Java. Esta classe recebe expressões escritas pelos usuários enviadas ao ambiente através dos dispositivos. Esta classe emprega o gerenciador de execução para criar instâncias de objetos segundo as classes associadas ao conceito.

A seguir apresentamos o diagrama de seqüência que mostra a interação que acontece entre todas as classes para a descoberta e execução de serviços *em um caso normal*. O diagrama não mostra a ajuda pró-ativa que acha o descobridor de serviços e nem o caso de múltiplos objetos do invocador de serviços. Estes dois casos são tratados como exceções. Estas exceções são enviadas ao cliente para ser resolvidas e dessa forma continuar o processo normal de execução. O diagrama da Figura 6.3 mostra o processo da descoberta de serviços.

O servidor do ambiente é formado por duas partes: (1) o serviço web do ambiente, implementado com a especificação *Jax-RPC* para serviços web em Java e (2) os motores e o gerenciador de motores de dispositivos, implementados em CORBA. Os motores são encarregados de manter as sessões com os clientes, uma para cada cliente conectado ao servidor do ambiente. Assim, o serviço web atua como fachada entre os dispositivos cliente e seu respectivo motor do ambiente.

6.1.2 Serviço web do servidor do ambiente

O acesso ao servidor do ambiente é feito através de um serviço web. Este serviço web serve como fachada que abstrai todos os processos do ambiente aos clientes.

Na Seção B do apêndice se mostra a interface WSDL acessível para os clientes. Toda interação entre os clientes e o servidor do ambiente é feita através somente destes métodos:

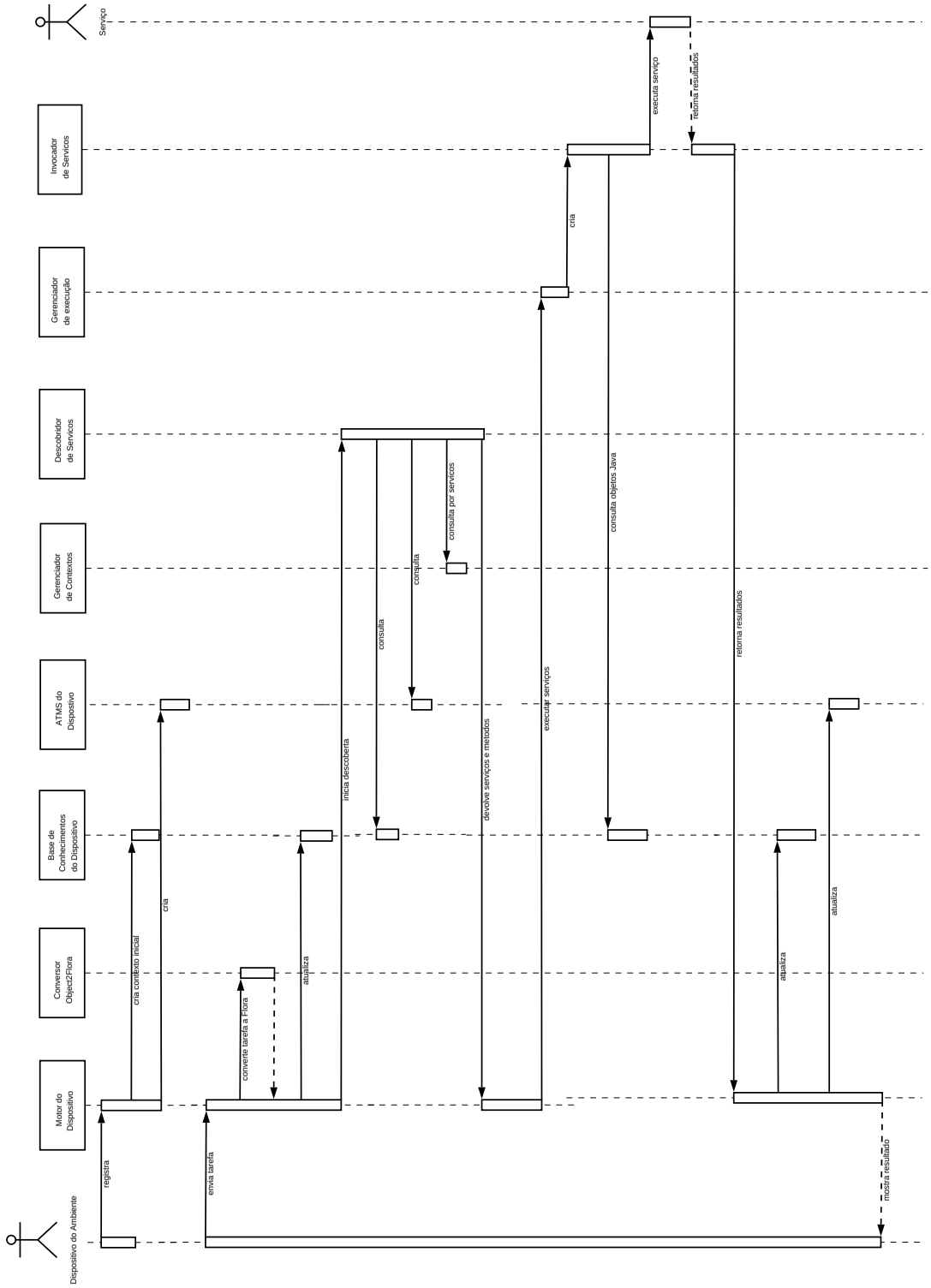

Figura 6.2: Interação entre os principais componentes do modelo.

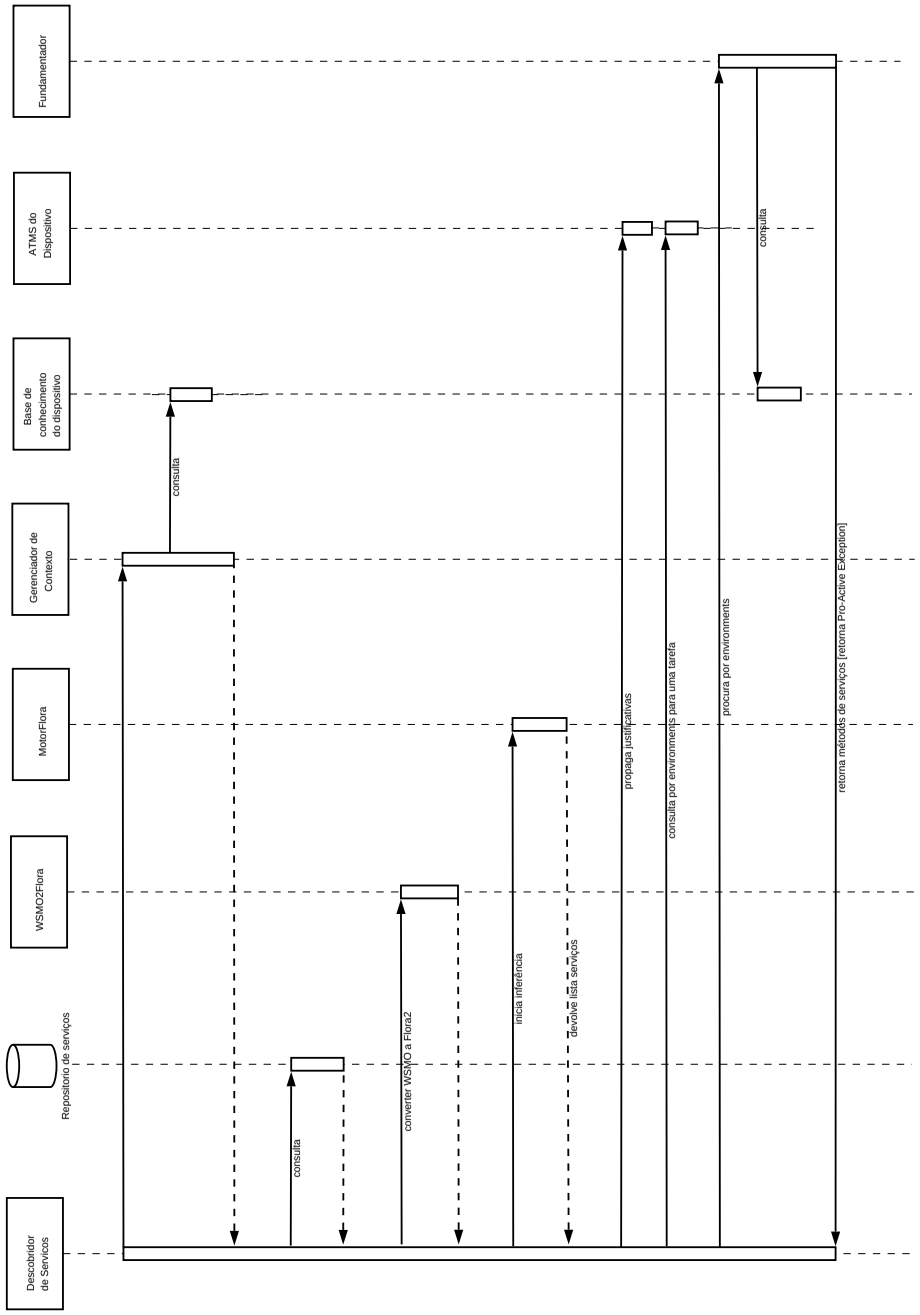

Figura 6.3: Interação de componentes para a descoberta de serviços.

`void register(HostData, ContextData):` Registra um dispositivo no ambiente. Recebe o endereço MAC do dispositivo encapsulado no parâmetro `HostData`. No registro do dispositivo também pode ser associado um contexto inicial ao dispositivo através do parâmetro `ContextData`. A informação contextual passada ao servidor são sentenças WSMO como `crhistian[hasNusp hasValue "5055668"] memberOf StudentFenix`.

`void setContextForHost(HostData, ContextData):` Atualiza o contexto de um dispositivo como o novo contexto passado no parâmetro `ContextData`.

`boolean hasException(HostData, GoalData):` Inicia todo o processo de descoberta e execução automática de serviços. Recebe a especificação da tarefa que se quer conseguir através do parâme-tro `GoalData`. Devolve `true` se todo o processo de descoberta e execução aconteceu sem problemas, e devolve `false` se foi produzida uma exceção `ProActive` ou `MultipleObject`. A primeira envia uma lista de serviços e métodos como ajuda pró-ativa ao usuário, e a segunda envia um lista de objetos que podem reificar/instanciar um conceito. É o usuário o encarregado de resolver estas exceções e continuar o processo através doe métodos `continueMultipleObject` ou `continueProActive`.

`WebServiceData[] getServiceForGoal(HostData):` Se o processo de descoberta e execução dos serviços foi normal (ou seja, o método anterior devolveu `true`) é possível obter o serviços e métodos que foram executados através deste método.

`ProActiveException getLastProActiveException(HostData):` Devolve, se existir, uma lista de métodos e serviços que servem de ajuda pró-ativa ao usuário.

`MultipleObjectException getLastMultipleObjectException(HostData):` Devolve uma lista de possíveis objetos para um conceito.

`void continueProActive(HostData, ProActiveException):` Quando o usuário completa a informação requerida na ajuda pró-ativa, o processo de descoberta e execução pode continuar através deste método.

`void continueMultipleObject(HostData, MultipleObjectException):` Os objetos escolhidos pelo usuário são enviados ao servidor do ambiente através deste método para continuar com a execução dos serviços.

`PositionData getPosition(HostData)`: Retorna a posição do dispositivo.

`String[] getKB(HostData)`: Retorna a base de conhecimentos do dispositivo como um arranjo de expressões WSMO.

`void addFactToKB(HostData, String)`: Adiciona um novo fato à base de conhecimento do dispositivo. Este método recebe uma cadeia de caracteres que é processada para criar o objeto Java a ser adicionado à base de conhecimento.

Note-se que todos os métodos precisam de um identificador do dispositivo passado através do parâmetro `HostData`. Isto porque a implementação do serviço web do ambiente não mantém estado de conexão com os clientes. A interação entre uma aplicação cliente e o servidor do ambiente se mostra na Figura 6.4.

As mensagens dentro do retângulo da figura podem ou não acontecer, dependendo de se alguma exceção foi lançada.

6.1.3 Motores para os dispositivos do ambiente

Um motor (*engine*) é um objeto CORBA encarregado de manter a sessão com um cliente (dispositivo) do ambiente. Este objeto CORBA acessa todos os outros componentes da arquitetura necessários para a descoberta e execução de serviços. Conforme foi apresentado, este componente representa a classe `Motor` do diagrama de classes da Figura 6.1. Desde que o serviço web do ambiente não matém estado com seus clientes, o gerenciamento dos motores criados por dispositivo é feito pelo gerenciador de motores (*engine home*) que é também um objeto CORBA.

Estes dois objetos CORBA foram configurados para ser transientes e com estado. Na Seção C do apêndice se mostra a interface IDL[2] do motor e gerenciador de motores (*engine home*). O serviço web do ambiente atua como uma fachada web ao ambiente, cada motor de dispositivo define os mesmos métodos na sua interface IDL que a interface WSDL do serviço web. Já a interface do gerenciador de motores define os métodos para a criação, eliminação e busca de motores baseado no identificador do dispositivo.

A Figura 6.5 mostra as duas partes do servidor do ambiente. Além disso, a figura mostra como a arquitetura proposta atua como um middleware entre os dispositivos do ambiente e os serviços, onde a comunicação nos dois lados é feita através da Internet.

[2] A interface IDL de um objeto CORBA define os métodos da interface desse objeto que são acessíveis remotamente.

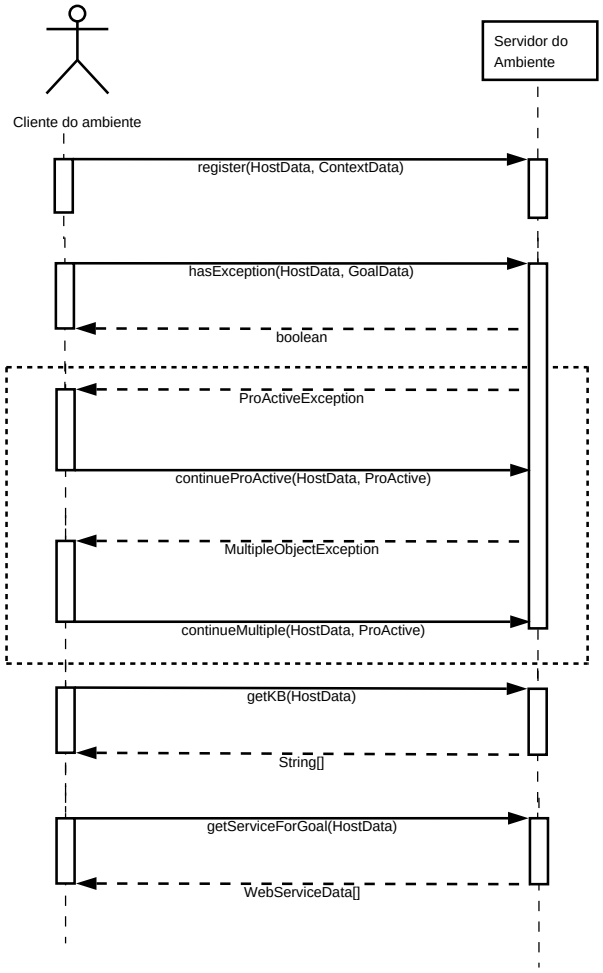

Figura 6.4: Interação entre uma aplicação cliente e o servidor do ambiente.

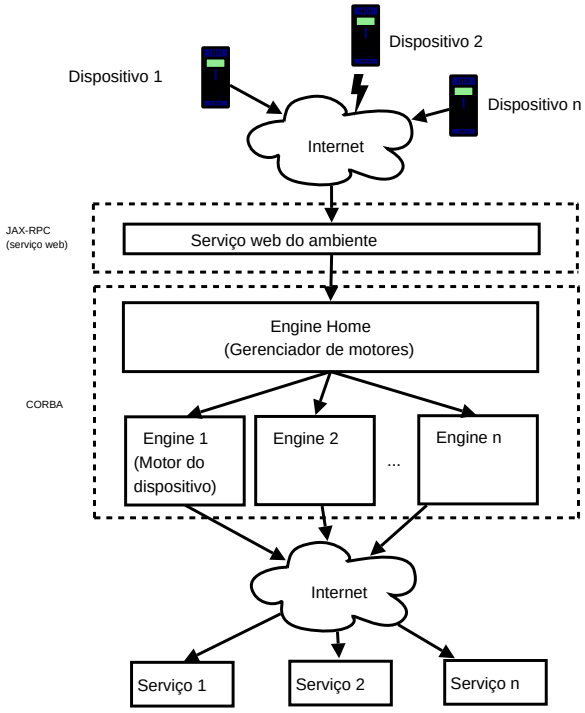

Figura 6.5: Servidor do ambiente.

6.2 Implementação do cliente do ambiente

O cliente foi implementado em C# do Microsoft Visual Studio .NET 2005. O cliente é executado em Windows CE 5.0 sobre um HP iPAQ Pocket PC. Um cliente do ambiente é um simples cliente do serviço web fornecido pelo servidor do ambiente. O uso de serviços web como meio de comunicação nos permite a interoperatividade entre as plataformas `.NET` e `J2EE`.

6.2.1 Interface genérica do usuário

Foi implementada uma *interface gráfica genérica* para o usuário. Através desta interface o usuário interage com o ambiente somente através de sentenças em WSMO. A partir disto é possível criar uma interface personalizada para um cenário específico. Por exemplo, para o caso do ambiente da biblioteca, que será apresentado com maior detalhe na seção seguinte, se im-

plementou uma interface que interpreta sentenças WSMO que representam livros e mostra ao usuário sua informação e posição no ambiente.

A Figura 6.6 mostra a tela de configuração do cliente. Nela é necessário configurar o endereço IP da máquina onde se está executando o servidor do ambiente. Opcionalmente, é possível configurar o ID do usuário e o endereço MAC do dispositivo.

Figura 6.6: Tela de registro.

A tela da Figura 6.7 permite ao usuário especificar uma tarefa ao ambiente e inicia o processo de descoberta e execução dos serviços apertando no botão *Init*. Para isto, o usuário digita a tarefa em WSMO o que implica que o usuário conheça as ontologias empregadas. Também, através desta tela o usuário pode adicionar fatos à sua base de conhecimentos.

A ajuda pró-ativa é apresentada ao usuário na tela da Figura 6.8. A interface mostra ao usuário os métodos mais próximos achados (como foi explicado na Seção 5.2.2) e a lista de expressões lógicas que faltaram para a execução de cada método. Assim, o usuário escreve uma instância dessa sentença na caixa de texto `New Object` que será enviada à base de conhecimento do dispositivo e poder continuar o processo.

Por exemplo, para a sentença `?somebook[hasIsbn hasValue ?someisbn] memberOf Book` o usuário declararia a instância `book1[hasIsbn hasValue`

Figura 6.7: Tela principal para o usuário.

Figura 6.8: Tela de ajuda pro-ativa.

`"12345623"] memberOf Book`. Uma vez que o usuário acaba com a ajuda pró-ativa pode continuar com o processo apertando o botão `Continue` da interface.

A Figura 6.9 apresenta o caso de ter múltiplos objetos para a execução de um método. Se mostra ao usuário uma lista de instâncias de objetos por método e o usuário deve escolher com qual desses objetos será efetuada a execução do método (como foi explicado na Seção 5.3.2). O usuário pode continuar a execução apertando o botão `Continue` da tela.

Figura 6.9: Tela de múltiplos objetos.

6.2.2 Sincronização com a base de conhecimentos

A aplicação cliente tem a sua base de conhecimento através do método `getKB` da interface WSDL do servidor do ambiente. Toda a informação de contexto e resultados de execução de serviços fica armazenado neste componente.

Dado que podem se executar serviços periodicamente, é possível que algum desses serviços adicione algum resultado à base de conhecimentos do dispositivo ou mude a informação contextual do dispositivo. Foi implementado no cliente um processo que é executado constantemente e que mantém

atualizada a interface do usuário com a base de conhecimento do dispositivo que fica no servidor do ambiente.

O usuário pode também adicionar novos fato à sua base de conhecimento através do método `addFactToKB` do servidor do ambiente.

6.3 O ambiente de teste

Nesta seção apresentamos o ambiente de teste implementado empregando o modelo proposto. Uma vez implementado o mecanismo de descoberta e execução de serviços, é fácil criar um ambiente inteligente.

Empregando o modelo proposto é necessário desenvolver as seguintes tarefas: (1) implementar os serviços web que implementaram as funcionalidades do serviço; (2) definir as ontologias empregadas na especificação semântica dos serviços; e (3) fazer a especificação semântica do comportamento dos serviços.

Na Figura 6.10 se mostra o PDA com que o usuário interage com o ambiente. Deve ser iniciado o monitor do MoCA e a aplicação cliente desenvolvida. Conforme foi explicado na Seção 6.1.1, através desta aplicação o usuário faz o registro do dispositivo no ambiente, estabelece um contexto inicial, uma meta que quer atingir e visualiza o estado da sua base de conhecimento com os resultados da execução de métodos de serviços.

Figura 6.10: Dispositivo com a aplicação cliente do ambiente.

6.3.1 Informação contextual

Foi necessário configurar o serviço de localização do MoCA para a área de biblioteca. A área foi dividida em 104 áreas como se mostra na Figura 6.11. Para cada área foi associado um ponto de referência onde foram coletadas sinais dos pontos de acesso para permitir ao MoCA inferir a área em que se encontra um dispositivo. A Figura 6.11 mostra o *Mapper Tool* do MoCA que serve para coletar potências dos sinais dos pontos de acesso que o dispositivo consegue receber no ponto de referência.

Figura 6.11: Mapa das áreas simbólicas na biblioteca no *Mapper* do MoCA.

Foram colocados três pontos de acesso e um roteador, ambos com o protocolo IEEE 802.11g (Wi-Fi), na área da biblioteca. Os pontos de acesso foram colocados nas áreas *area30*, *area49* e *area88*, e o roteador na área *area95*. Na Figura 6.12 se mostra o roteador instalado na biblioteca. Os três pontos de acesso foram conectados ao roteador através de uma rede fixa.

O Monitor do MoCA (Seção 3.5) pode ser configurada a freqüência com que enviará informação contextual ao serviço CIS e manter atualizada a localização do dispositivo. Nos experimentos efetuados esta freqüência foi configurada a 500 milisegundos. Assim, as respostas do sistema estão de

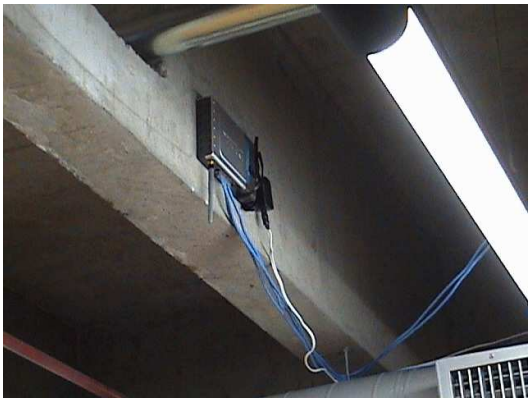

Figura 6.12: Roteador Wi-Fi da rede.

acordo com a localização do usuário no ambiente. A precisão do serviço de locali-zação do MoCA depende da quantidade de pontos de acesso no ambiente. Nos experimentos efetuados se apresentou uma margem de erro de 2 metros.

6.3.2 Ontologias implementadas

Foram definidas as seguintes ontologias para a especificação de serviços. Na Seção D do apêndice se mostra a definição completa em WSMO destas ontologias.

`Library:` Esta ontologia contém os conceitos referentes a livros na biblioteca como: `Book`, `ListBook`, `BookTheme`, `BookPlace` e `RelatedBooks`.

`Fenix:` Contém os conceitos referentes à informação acadêmica de um aluno, como estudantes, professores, disciplinas, departamentos, etc. Os conceitos definidos nesta ontologia foram: `StudentFenix`, `TeacherFenix`, `CourseFenix` e `Deparment`.

`Ambient:` Contém os conceitos: `Device`, `User` e `Context` que representam entidades do próprio ambiente. Estes conceitos são usados na especificação dos serviços da arquitetura (Seção 5.4).

`Location:` Define os conceitos para localização no ambiente. O conceito base de localização é `Location` que define como atributos x, y e z. Já o conceito `Region` faz referência a uma área simbólica no ambiente. Assim, é

possível associar uma localização a qualquer outro conceito. Por exemplo, a localização de um livro representado no conceito `BookPlace` é um sub-conceito do conceito `Region`, acrescido do número de estante e área temática do livro.

`People`: Define as propriedades e conceitos básicos para a definição de pessoas no ambiente. Qualquer conceito que represente uma pessoa no ambiente é um sub-conceito do conceito `People` definido nesta ontologia. O conceito `People` tem a propriedade `hasLocation` que associa a localização da pessoa no ambiente.

6.3.3 Serviços implementados

Foram implementados três serviços para o ambiente da biblioteca. Na Seção E do apêndice se apresenta a definição semântica completa em WSMO destes serviços.

1. Serviço `Fenix`

Este serviço permite consultar o estado acadêmico de um estudante. Foi simulado o acesso ao sistema Fênix da USP. Este serviço fornece as seguintes informações:

1. Listar livros cadastrados como bibliografia para uma disciplina.
2. Listar disciplinas matriculadas de um estudante.
3. Listar livros para um estudante (com base nas disciplinas de um estudante é possível extrair os livros relacionados).

Cada uma destas funcionalidades está associada às seguintes especificações de um método do serviço, respectivamente:

1.
```
if (?someCourse[hasCode hasValue ?someCode]
      memberOf fex#CourseFenix) then
   add(?listbooks memberOf lib#CourseBooks)
endIf
```

2.
```
if (?someStudent[hasNUSP hasValue ?someNusp]
      memberOf fex#StudentFenix) then
   add(?listcourses memberOf fex#ListCourses)
endIf
```

3.
```
if (?someStudent[hasNUSP hasValue ?someNusp]
        memberOf fex#StudentFenix) then
   add(?listbooks memberOf lib#CourseBooks)
endIf
```

2. Serviço `Library`

Este serviço encapsula as funcionalidades fornecidas pela biblioteca. Para os experimentos efetuados, este serviço acessa informação ao sistema Cólmeia para consultas sobre o acervo da biblioteca do Instituto de Matemática e Estatística da USP. Este serviço permite:

1. Reservar um livro.
2. Listar livros relacionados a um livro dado.
3. Listar livros relacionados a um tema dado.
4. Listar livros próximos ao usuário no ambiente.

Os métodos do serviço que implementam estas funcionalidades, respectivamente, são:

1.
```
if (?someBook[isAvailable hasValue true]
        memberOf lib#Book) then
   add(?someBook[isAvailable hasValue false]
           memberOf lib#Book)
endIf
```

2.
```
if (?someBook[hasISBN hasValue ?someISBN]
        memberOf lib#Book) then
   add(?relatedbooks memberOf lib#RelatedBooks)
endIf
```

3.
```
if (?someTheme memberOf lib#BookTheme) then
   add(?disciplinebooks memberOf lib#AvailableBooks)
endIf
```

```
4. if (?someStudent[hasLocation hasValue ?someLocation]
         memberOf fex#StudentFenix) then
      add(?relatedbooks memberOf lib#RelatedBooks)
   endIf
```

3. Serviço `Location`

Este serviço fornece orientação ao usuário para chegar a um livro dentro da área da biblioteca. O método deste serviço recebe um identificador de um livro e uma entidade estudante e devolve a localização e orientação para chegar ao livro.

Foi implementado este serviço fazendo consultas a um banco de dados que mantém as informações sobre livros que se encontram em cada uma das áreas segmentadas da biblioteca (Seção 6.3.1). Esta associação entre áreas e livros foi feita manualmente no banco de dados. A implementação deste serviço poderia ser melhorada empregando um sistema de etiquetas RFID, o que permitiria uma atualização automática da localização de um livro.

O método deste serviço tem a seguinte especificação:

```
1. if (?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book
         and ?someStudent[hasLocation hasValue ?somelocation]
               memberOf fex#StudentFenix) then
      add(?someDirection memberOf loc#Direction)
   endIf
```

6.3.4 Descoberta e execução dos serviços

O usuário estabelece como tarefa que deseja uma lista de livros relacionados e uma localização. Entende-se como livros relacionados aqueles livros que estão relacionados a algum conceito, como a posição do usuário ou algum outro livro. Assim, a expressão definida pelo usuário é:

```
?bookList memberOf RelatedBooks and ?someDirection
     memberOf Direction.
```

Além disto, a base de conhecimentos do dispositivo inicialmente contém:

```
crhistian[hasLocation hasValue ime, hasNUSP hasValue 5055668]
   memberOf StudentFenix
```

A partir da tarefa especificada e dos serviços `Library` e `Fenix`, o mecanismo de descoberta do ambiente monta o ATMS estabelecendo as relações entre hipóteses e efeitos. Na Seção F do apêndice se mostra o estado final do ATMS para a descoberta. Assim, o nó do ATMS correspondente à meta `?bookList memberOf RelatedBooks` é:

```
<?relatedbooks memberOf lib#RelatedBooks. ,

{  {?someBook[hasISBN hasValue ?someISBN]
        memberOf lib#Book. }
   {?someStudent[hasLocation hasValue ?someLocation]
        memberOf fex#StudentFenix. }
   {?someCourse[hasCode hasValue ?someCode]
        memberOf fex#CourseFenix. }
   {?someStudent[hasNUSP hasValue ?someNusp]
        memberOf fex#StudentFenix.  }
} >
```

O ATMS associou à meta `?relatedbooks memberOf lib#RelatedBooks` quatro *environments*: (1) `{?someBook[hasISBN hasValue ?someISBN] memberOf lib#Book}`, (2) `{?someStudent[hasLocation"hasValue ?someLocation] memberOf fex#StudentFenix}`, (3) `{?someCourse[hasCode hasValue ?someCode] memberOf fex#CourseFenix }` e (4) `{?someStudent[hasNUSP hasValue ?someNusp] memberOf fex#StudentFenix}`.

Estes *environment* são achados a partir da *"composição"* dos serviços `Library` e `Fenix`. Por exemplo, o serviço `Fenix` nos devolve a lista de livros que foram cadastrados nas disciplinas que o estudante está cursando. Isto na especificação:

```
if (?someStudent[hasNUSP hasValue ?someNusp]
       memberOf fex#StudentFenix)
then
   add(?listbooks memberOf lib#ListBooks)
endIf
```

já o serviço `Library` devolve uma lista de livros relacionados a partir de um livro. Isto na especificação:

```
if (?someBook[hasISBN hasValue ?someISBN] memberOf lib#Book)
then
   add(?relatedbooks memberOf lib#RelatedBooks)
endIf
```

A partir destas regras o ATMS acha o *environment* `{?someStudent[ hasNUSP hasValue ?someNusp] memberOf fex#StudentFenix}`.

O *environment* `{?someStudent[hasLocation hasValue ?someLocation] memberOf fexStudentFenix}` oferecido pelo serviço `Library` é obtido diretamente pela especificação:

```
if (?someStudent[hasLocation hasValue ?someLocation]
     memberOf fex#StudentFenix) then
   add(?relatedbooks memberOf lib#RelatedBooks)
endIf
```

Assim, os *environment* `{?someStudent[hasLocation hasValue ?someLocation] memberOf fex#StudentFenix}` e `{?someStudent[hasNUSP hasValue ?someNusp] memberOf fex#StudentFenix}` podem ser cobertos pela base de conhecimento do dispositivo, uma vez que temos `crhistian[hasLocation hasValue ime, hasNusp hasValue "5055668"] memberOf fex#StudentFenix`.

Na Seção F do apêndice se mostram os nomes dos métodos da interface WSDL associados às regras de transição dos serviços cujas pre-condições se encontram nos *environments* satisfeitos. Assim, os métodos executados são:

1. `someLocation-relatedbooks hasValue "booksForPosition:transient-many"`

   (Serviço `Library`)

2. `someStudentlistbooks hasValue "findBooksStudent:transient-one"`

   (Serviço `Fenix`)

O método `bookForPosition` é executado várias vezes e o resultado(`?relatedbooks memberOf lib#RelatedBooks`) não é adicionado à base de conhecimentos do dispositivo, desde que o resultado é temporal (propriedade *transient*). A configuração de executar este serviço várias vezes permite que o usuário receba a lista de livros relacionados segundo a sua localização na biblioteca. Já o segundo método mostra ao estudante os livros referenciados nas disciplinas que está cursando e executado só uma vez.

Já para a meta `?someDirection memberOf loc#Direction` temos a rotulação

```
<?someDirection memberOf loc#Direction. ,
```

```
{   { ?someStudent[hasLocation hasValue ?somelocation]
         memberOf fex#StudentFenix.
      ?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book.
    }

    { ?someStudent[hasLocation hasValue ?somelocation]
         memberOf fex#StudentFenix.
      ?someCourse[hasCode hasValue ?someCode]
         memberOf fex#CourseFenix.
    }

    { ?someStudent[hasLocation hasValue ?somelocation]
         memberOf fex#StudentFenix.
      ?someStudent[hasNUSP hasValue ?someNusp]
         memberOf fex#StudentFenix.
    }
}>
```

com os *environments*

1.
```
{ ?someStudent[hasLocation hasValue ?somelocation]
      memberOf fex#StudentFenix.
    ?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book.
}
```

2.
```
{  ?someStudent[hasLocation hasValue ?somelocation]
      memberOf fex#StudentFenix.
    ?someCourse[hasCode hasValue ?someCode]
      memberOf fex#CourseFenix.
}
```

3.
```
{ ?someStudent[hasLocation hasValue ?somelocation]
      memberOf fex#StudentFenix.
   ?someCourse[hasCode hasValue ?someCode]
      memberOf fex#CourseFenix.
}
```

O primeiro *environment* é achado diretamente a partir da seguinte regra de transição do serviço `Location`.

```
if (?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book
      and
   ?someStudent[hasLocation hasValue ?somelocation]
      memberOf fex#StudentFenix)
then
      add(?someDirection memberOf loc#Direction)
```

O segundo *environment* é formado a partir da propagação da seguinte regra de transição do serviço `Fenix` no primeiro *environment*.

```
if (?someCourse[hasCode hasValue ?someCode]
       memberOf fex#CourseFenix)
then
    add(?listbooks memberOf lib#ListBooks)
```

Isto é conseguido tornando o fato `?someBook[hasIsbn hasValue ?someIsbn] memberOf Book` do primeiro *environment* como sub-meta (conforme descrito na Seção 5.2). O processo de descoberta encontra e propaga esta regra de transição. Note-se que esta regra devolve uma lista de livros (`?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book`) que pode ser obtida pela execução do método associado.

O terceiro *environment* é achado da mesma maneira mas propagando a seguinte regra de transição que também pertence ao serviço `Fenix`.

```
if (?someStudent[hasNUSP hasValue ?someNusp]
       memberOf fex#StudentFenix)
then
   add(?listbooks memberOf lib#ListBooks)
```

A partir da base de conhecimento do dispositivo é possível satisfazer o terceiro *environment* que tem associados os métodos `findBooksStudent` e `goItBook`, um por cada elemento do *environment*. Pela profundidade da expansão é executado primeiro o método `findBooksStudent`. Conforme é explicado na próxima seção, quando o usuário adiciona explicitamente um livro na sua base de conhecimento, o mecanismo de descoberta pode satisfazer o primeiro *environment* que tem associado o método `goIt` que devolve finalmente uma localização para o usuário (`?someDirection memberOf`

`loc#Direction`) desse livro. Pelas propriedades de execução o método `goIt` é executado várias vezes, e se o método receber a posição corrente do usuário (`?someStudent[hasLocation hasValue ?somelocation] memberOf fex#-StudentFenix`), o método devolve a posição que o usuário deve seguir para achar o livro conforme se movimenta na biblioteca.

6.3.5 Interface cliente para o ambiente da biblioteca

A Figura 6.13 mostra a tela para o usuário. Nela se mostra a lista de livros próximos do usuário ou na sua base de conhecimentos. Selecionando um livro da lista e apertando o botão *Go it!* a seta indica ao usuário a direção que deve seguir para achar o livro selecionado. A direção devolvida considera que o usuário se encontra orientado a um lugar fixo.

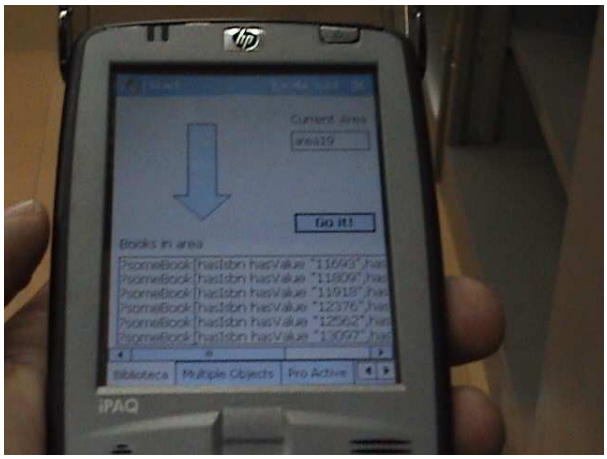

Figura 6.13: Tela do usuário para o ambiente da Biblioteca

Quando o usuário seleciona um livro e aperta o botão *Go it!* essa informação é introduzida na base de conhecimento do dispositivo. Com este fato na base de conhecimento é possível executar o método `GoIt` do serviço `Location` e o método `relatedBooks` do serviço `Library`. O primeiro retorna uma localização (`Direction`) para onde o usuário deve seguir para achar o livro selecionado. O segundo método devolve uma lista de livros que poderiam ser de interesse para o usuário associados ao livro na sua base de conhecimento.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Cenários para o modelo proposto

Em [16] se apresentam cenários onde os ambientes inteligentes podem ser satisfatoriamente aplicados e teriam grande utilidade. Estes cenários sugerem direcionamentos para pesquisas em ambientes inteligentes. Os cenários montados consideram fatores sociais, tecnológicos, econômicos e outros onde os ambientes inteligentes podem influenciar.

Com base nesses cenários, propomos três cenários para mostrar o potencial do modelo proposto. Os cenários analisados são:

- Um ambiente reativo à presença de pessoas;
- Um ambiente de recomendação numa rede metropolitana;
- Um ambiente com composição de serviços.

A implementação de um ambiente com o modelo proposto requer a definição de ontologias com que serão feitas as especificações semânticas dos serviços, os quais requerem a implementação dos serviços web que ofereçam a funcionalidade requerida.

A seguir se apresenta como seria implementado cada cenário.

7.1.1 Um ambiente reativo à presença de pessoas

Neste cenário se ressalta o funcionamento do mecanismo de descoberta e execução de serviços.

Neste cenário, um serviço é implementado para controlar um objeto físico no ambiente. Por exemplo, um serviço que controla a impressora do ambiente, que recebe arquivos binários, os converte no formato correto e os envia à impressora.

Além disso, este serviço de impressão é oferecido ao usuário segundo a proximidade do usuário como a impressora no ambiente. A descrição deste serviço é:

```
if(?somedocument[hasId hasValue ?someId, isPrinted hasValue false]
       memberOf BlobDocument
   and ?user[hasLocation hasValue "area1"]
       memberOf User
)
   then (?somedocument[isPrinted hasValue true]
            memberOf BlobDocument)
```

Assim, a base de conhecimentos do dispositivo teria:

```
crhistian[hasLocation hasValue "area1"] memberOf User.

excel_document[hasId hasValue 1342,
               hasName hasValue "my_sheet.xls",
               isPrinted hasValue false] memberOf BlobDocument.
```

A regra de transição do serviço propagada no ATMS do dispositivo cria o *environment* `{?somedocument[...isPrinted hasValue false] memberOf BlobDocument, ?user[hasLocation hasValue "area1"] memberOf User}` que pode ser coberto pela base de conhecimentos do dispositivo e pelos métodos executados. Neste ponto, podem acontecer duas coisas:

1. *O usuário muda de área:* Neste caso o ATMS ainda conservaria o *environment*, mas a base de conhecimentos do dispositivo teria agora `crhistian[hasLocation hasValue "area2"] memberOf User` e desde que o serviço requer algum `User` na área *"area1"*. O usuário teria que se aproximar da área da impressora para conseguir imprimir o documento.

2. *O serviço fica indisponível:* O processo de descoberta de serviços que é executado freqüentemente atualiza o ATMS do dispositivo. Neste caso o *environment* no ATMS é eliminado. O mecanismo de descoberta

procura por outros serviços que permitam imprimir os documentos. Estes outros serviços podem ser impressoras do prédio ou uma outra impressora na sala contígua.

Da mesma forma que o serviço de impressora, vários outros serviços podem ser colocados no ambiente para se ativar na presença de pessoas. Por exemplo, pode ser implementado um serviço que ajusta automaticamente a temperatura do ambiente às preferências do usuário.

A especificação semântica deste serviço seria:

```
if(?temperature[hasValueCelcius hasValue ?somevalue]
       memberOf Temperature
   and ?user[hasLocation hasValue "sala1"]
       memberOf User
)
   then (?temperature memberOf AmbientTemperature)
```

A preferência da temperatura do usuário é introduzida na sua base de conhecimentos com a sentença.

```
temperature[hasValueCelcius hasValue 24] memberOf Temperature.
crhistian[hasLocation hasValue "sala1"] memberOf User.
```

Modelo Sensor-Atuador

Mais genericamente, este tipo de cenário se baseia no modelo sensor-atuador. Isto é, os sensores são dispositivos no ambiente encarregados de capturar alguma informação específica do ambiente (como a temperatura do ambiente) e passam esta informação aos atuadores para efetuar alguma operação que podem modificar o ambiente.

Em nosso modelo, um sensor e um atuador são implementados cada um por um serviço. Assim, na definição do sensor é definido o tipo de informação que se coleta do ambiente e na definição do atuador são definidos os efeitos que se produz no ambiente.

Como exemplo, consideramos um sensor de temperatura (o sensor) e um aquecedor que regula a temperatura do ambiente (o atuador). O sensor teria a especificação:

```
if(?user[hasLocation hasValue "quarto1"]
       memberOf User
)
   then (?temperature memberOf Temperature)
```

Esta regra define que este serviço devolve a temperatura para qualquer usuário que se encontre em `sala1` que representa a área onde o sensor pode detectar adequadamente a temperatura do ambiente.

Já a especificação do aquecedor seria:

```
if(?temperature[hasValueCelcius hasValue ?somevalue]
)
   then (?temperature memberOf AmbientTemperature)
```

Note-se que o atuador requer como entrada a mesma informação que o sensor devolve. O mecanismo de descoberta de serviços acharia no ATMS:

```
< ?temperature memberOf AmbientTemperature,
  {{?user[hasLocation hasValue "quarto1"] memberOf User}}>
```

O serviço do atuador apenas sabe receber a temperatura do ambiente e regular a temperatura, com base nessa informação ou em alguma outra que não seja passada como parâmetro mas que seja parte de sua implementação. Por exemplo, o atuador pode regular a temperatura do ambiente também baseado na temperatura da região da cidade, fazendo consultas a um serviço web (ou algum outro mecanismo) que retorna a temperatura atualizada da cidade.

Note-se que a associação entre o sensor e atuador é feita pelo mecanismo de descoberta de serviços, o que permite que esta associação seja feita de maneira dinâmica, permitindo que um sensor possa ser associado a outros atuadores do ambiente. Como no caso da temperatura, o sensor de temperatura pode ser associado a um serviço que gerencia a energia do prédio e, baseado na temperatura do ambiente, pode decidir economizar energia.

7.1.2 Um ambiente de recomendação numa rede metropolitana

Neste cenário ressaltamos as características da arquitetura proposta, mais especificamente se mostra como é aproveitada a ubiqüidade da Internet.

A arquitetura emprega serviços web como sistema de comunicação distribuído, o que permite a ativação de um ambiente uma rede metropolitana, como o campus de uma universidade ou mesmo uma cidade. Este cenário requer um mecanismo de localização de área extensa como parte da arquitetura, como GPS ou Placelab [28].

O usuário insere na sua base de conhecimento o "desejo" de comprar algum produto; por exemplo uma geladeira. A base de conhecimentos do dispositivo teria:

```
   geladeira[hasModel hasValue "LG"] memberOf Product.
   preferencia[hasElements hasValue geladeira] memberOf Prefer.
```

Os serviços são implementados por lojas de eletrodomésticos que oferecem ofertas e informação sobre seus produtos. Os serviços são oferecidos aos usuários pelo mecanismo de descoberta segundo a localização do usuário. Os serviços são divididos por zonas da cidade os associando a contextos de localização através do *gerenciador de contexto*.

Assim, quando o usuário se movimenta pela cidade as informação de lojas próximas são apresentadas ao usuário. Note-se que no momento que o usuário insere uma preferência na sua base de conhecimentos pode não ser oferecido nenhum serviço é segundo ele se movimenta na cidade ficam novos serviços disponíveis.

A descrição do serviço implementado por alguma loja como esta funcionalidade seria:

```
if(?someprefer[hasElements hasValue ?someproduct]
       memberOf Prefer
  and ?user[hasLocation hasValue ?somearea]
       memberOf User
)
   then add(?someInfo memberOf ProductInformation)
```

O relevante deste cenário é que o usuário pode acessar aos componentes da arquitetura (base de conhecimentos, ATMS e mecanismo de descoberta e execução de serviços) através da Internet o que lhe permite se movimentar na cidade. Neste cenário os *fornecedores de serviços* da arquitetura são fornecidos pelas lojas e podem ser facilmente adicionados ao *repositório de serviços* do ambiente.

Os serviços oferecidos pelas lojas podem fornecer mais funcionalidades como permitir a compra dos produtos. Para esse caso teríamos a seguinte especificação:

```
if(?someprefer[hasElements hasValue ?someproduct]
       memberOf Prefer
  and ?creditcard[hasUser hasValue ?someuser,
                  hasType hasValue ?someType]
       memberOf CreditCard
)
   then
      add( ?somestatus[hasStatus hasValue buyOk]
                memberOf StatusPayment)
```

7.1.3 Um ambiente com composição de serviços

Neste cenário mostramos como pode ser empregada a arquitetura e os serviços da arquitetura apresentados no Capítulo 5 de maneira conjunta, para conseguir maior interatividade entre os dispositivos no ambiente.

Quando um estudante entra numa biblioteca ele recebe os temas de interesse das pessoas que se encontram nesse momento na biblioteca. Assim, o estudante poderia fazer consultas ou ter uma discussão de algum assunto particular.

Para isto disponibilizamos um serviço que contém informação dos interesses do usuário. Este serviço pode fazer consultas num banco de dados que contém as informações dos usuários; por exemplo, em uma universidade este serviço poderia fazer consultas a um sistema de matrículas para conhecer o grau acadêmico dos usuários.

Assim, este serviço só sabe responder os interesses associados a um usuário. A descrição deste serviço seria:

```
if(?someUser memberOf User) then
   add(?someprefers[hasElements hasValue ?someprefer]
         memberOf ListPrefer)
```

Note-se que este serviço poderia ser um serviço aberto acessível pela Internet. A meta estabelecida pelo usuário para este fim (o de obter informação dos usuários na biblioteca) seria:

```
?someprefers memberOf ListPrefer.
```

O conceito `ListPrefer` é uma lista que contém elementos do conceito `Prefer`. A definição destes dois conceitos é:

```
concept Prefer
     hasUser impliesType User
     hasPrefers impliesType (1 *) _string

concept ListPrefer
     hasElements impliesType (1 *) Prefer
```

A partir deste serviço e do serviço de proximidade do ambiente (Seção 5.4) a propagação destas regras no ATMS acharia:

```
< ?someprefers memberOf ListPrefer,
  {{?user[hasLocation hasValue ?somearea] memberOf User}}>
```

Isto a partir das justificativas:

1. `{?someuser[hasLocation hasValue ?somearea] memberOf User} => ?relatedusers memberOf ListUser)`

   do serviço de proximidade e

2. `{?someUser memberOf User} => ?someprefers[hasElements hasValue ?someprefer] memberOf ListPrefer`

   do serviço de interesses.

O *environment* `{?user[hasLocation hasValue ?somearea] memberOf User}` pode ser coberto pela base de conhecimentos do dispositivo e a execução freqüente dos métodos mostraria a informação atualizada segundo a proximidade do estudante com as pessoas na biblioteca.

Com base nos serviços apresentados nestes cenários é possível criar novos cenários que ofereçam maiores desafios aos ambientes inteligentes. Os cenários apresentados em [16] podem servir de inspiração para a implementação de ambientes empregando o modelo proposto.

7.2 Conclusões

Os experimentos realizados como prova de conceito do modelo para ambientes inteligentes proposto, foram satisfatórios.

É importante ressaltar do modelo proposto que a construção de um ambiente inteligente se baseia na implementação de serviços isolados e que através de uma especificação semântica do serviço, o modelo proposto permite ativar a descoberta e execução automática desses serviços para os usuários. Assim, a implementação de um ambiente inteligente se reduz à implementação das funcionalidades requeridas para o ambiente em um serviço web.

Em uma implementação real, a implementação de um serviço web pode ser feita sobre sistemas legados. Esse foi o caso na implementação do serviço `Library`, que faz consultas ao banco de dados do sistema *Colmeia*, o qual mantém o acervo de revistas e periódicos científicos da biblioteca. Já a implementação do serviço `Location` se baseia em consultas em um banco de dados que associa revistas do acervo a áreas da biblioteca. Pelos princípios de arquiteturas baseadas em serviços, a implementação deste serviço poderia ser modificada e o ambiente inteligente poderia operar normalmente. Por exemplo, a implementação deste serviço pode empregar sistemas RFID para manter automaticamente a localização de uma revista. Além disso, para

poderem ser ativadas novas funcionalidades dos serviços no ambiente, bastaria com atualizar a descrição semântica do serviço com as regras de transição que representem essas novas funcionalidades. Por exemplo, o serviço `Library` poderia permitir a reserva de livros acrescentando a regra:

```
if( ?someBook memberOf Book and
    ?someStudent[hasNusp hasValue ?someNusp]
   memberOf StudentFenix
  )
then
     add(?someBook[isReserved hasValue true] memberOf Book)
endIf
```

Existem limitações no modelo. Podem ser encontrados métodos que podem ser executados sem que o usuário tenha a intenção de os executar, por exemplo, no caso de reservas de livros, quando o usuário solicita reserva de um livro (estabelecendo a meta `?someBook[isReserved hasValue true] memberOf Book`). Se a base de conhecimento do dispositivo tiver exatamente um livro, este método seria executado passando como parâmetro esse livro (se houvesse mais de um livro o ambiente consultaria o usuário com qual livro deve ser executado o método), e pode acontecer que o usuário não tinha desejado reservar aquele livro em específico, senão algum outro obtido pela execução de algum serviço.

Nesta situação é necessário desenvolver um cliente específico para evitar essa situação. Por exemplo, bastaria colocar um botão que inicie a reserva de um livro. Além de estabelecer a nova meta `?someBook[isReserved hasValue true] memberOf Book`, ter-se-ia que remover todos os livros da base de conhecimento do dispositivo (através do método `removeFactToKB` do servidor do ambiente). Ao apertar esse botão o usuário deveria ter selecionado um livro para ser adicionado à base de conhecimento e executar o método para esse livro. O livro pode ser obtido a partir de uma lista de livros da base de conhecimento ou da informação transiente da execução de serviços. Nos experimentos realizados, uma situação semelhante foi resolvida através da interface da biblioteca (Seção 6.3.5) para guiar o usuário até um livro dentro da biblioteca. Desta maneira uma aplicação cliente projetada para um domínio específico inter-age com a interface WSDL do serviço web do ambiente. A aplicação cliente pode processar, interpretar ou apresentar ao usuário o conteúdo da sua base de conhecimento de maneira mais adequada.

Uma necessidade dos ambientes inteligentes é uma interface intuitiva para os usuários [41]. No modelo proposto o usuário trata somente de instâncias

dos conceitos definidos nas ontologias. As ontologias se mostram mais intuitivas para os usuários, pois uma ontologia se refere a conceitos do *domínio do usuário* e não a conceitos relacionados à *implementação dos serviços*. Embora na maioria dos casos os conceitos empregados sejam os mesmos nessas duas visões, isto permite mudar os conceitos do domínio sem mudar as implementações, ou seja, mudar a semântica do serviço sem mudar sua implementação.

Conforme foi apresentado na seção anterior, que mostra cenários para o modelo proposto, a descrição semântica dos serviços através de regras de transição (próprias da WSMO) permite a interoperabilidade entre serviços no ambiente. O cenário que mostra isto para nosso modelo se apresenta na Sub-seção 7.1.1, que permite associar dinamicamente um sensor e um atuador (implementados como serviços). Isto permite criar um ambiente com tolerância a falhas mediante a replicação de serviços. Ou seja, se um sensor ficasse indisponível para fornecer um tipo de informação, o mecanismo de descoberta pode encontrar serviços alternativos que forneçam o mesmo tipo de informação.

O uso de um ATMS para a descoberta de serviços mostrou ser um mecanismo rápido e que permite lidar com a variabilidade no contexto do dispositivo e a disponibilidade de serviços no ambiente. Existem abordagens que tratam a descoberta de serviços como um problema de planejamento em inteligência artificial. Isto é, dado um conjunto de ações e um estado inicial, achar a seqüência de ações (chamada de plano) que podem levar o estado inicial ao estado meta determinado. Em [46] são apresentados os métodos existentes para o tratamento desde problema. Nesta abordagem, diante de mudanças nos estados (contexto do dispositivo), é necessário fazer replanejamentos constantes, o que é uma tarefa que consome tempo e poder computacional.

No ATMS não foi estudado o uso de *no-goods* do ATMS (Seção 2.5). Isto permitiria adicionar restrições sobre os *environmets* a serem achados para um efeito, pois nenhum *environmet* possui como sub-conjunto um *no-good*. Por exemplo, se quisermos evitar executar métodos para clientes que estão na área `area10` e que tenham memória livre abaixo de 50%, podemos adicionar o seguinte *no-good* no ATMS:

```
<{?somestudent[hasLocation hasValue "area10"],
  ?somestudent[freeMemory hasValue 50]}>
```

A respeito da implementação do ambiente na biblioteca. Era esperado que a precisão do MoCA para a localização do dispositivo não seja exata. A

exactitude na localização de um dispositivo vária por metros o que faz que os livros de uma área próxima sejam apresentados ao usuário e não os livros da área exata em que se encontra.

7.3 Trabalhos futuros

Nesta seção descrevemos propostas de pesquisas que pretendem ser desenvolvidas com base nesta dissertação.

7.3.1 Um middleware inteligente

Na Seção 5.2 foi sugerido que o servidor do ambiente se converte em um mediador inteligente para a descoberta e execução de serviços entre os clientes e os fornecedores de serviços. Sucintamente, este mediador tem as seguintes funcionalidades:

1. Faz a descoberta e execução de serviços automaticamente para o usuário, baseado na sua tarefa especificada.

2. Coleta informação contextual do dispositivo e do usuário para necessários para a descoberta e execução de serviços.

3. Realiza a comunicação com os clientes e os fornecedores de serviços é feita através da Internet.

4. Devido aos processos de descoberta e execução que demandam altos recursos computacionais, o mediador é executado num nó fixo da rede.

A principal desvantagem da arquitetura proposta é se basear em uma entidade centralizada, o servidor do ambiente. Um sistema centralizado tem várias desvantagens quando comparado com um sistema distribuído.

Como trabalho futuro se pretende tornar o servidor do ambiente em um *middleware adaptativo* com as mesmas funcionalidades do servidor do ambiente. Especificamente, o middleware fornece a adaptabilidade baseado no contexto do dispositivo para a execução dos serviços contidos nele. O middleware pretende ser executado em um dispositivo fixo ou móvel com as capacidades computacionais necessárias e pretende ativar automaticamente a interação com novos dispositivos que se incorporem ao ambiente.

Numa primeira visão este middleware teria as seguintes características:

1. Deve poder ser executado em dispositivos móveis e fixos.

2. Ativa a descoberta e execução automática de serviços para o cliente. Além disso, converte o cliente em um provedor de serviços.

3. Permite adaptabilidade na execução dos serviços baseado no contexto do dispositivo.

4. Coletar e processar informação contextual dos dispositivos para os processos anteriores.

Nesta proposta se pretende seguir a abordagem do middleware *MobiPADS* [7] sobre a cadeia de serviços para o controle da adaptabilidade. Mas a modelagem e processamento da informação contextual estará baseada em ontologias. Para isto é necessário definir uma ontologia para definir a adaptabilidade do middleware. *O uso de ontologias nos permite maior expressividade na definição de contextos e maior flexibilidade na definição de regras de adaptabilidade.*

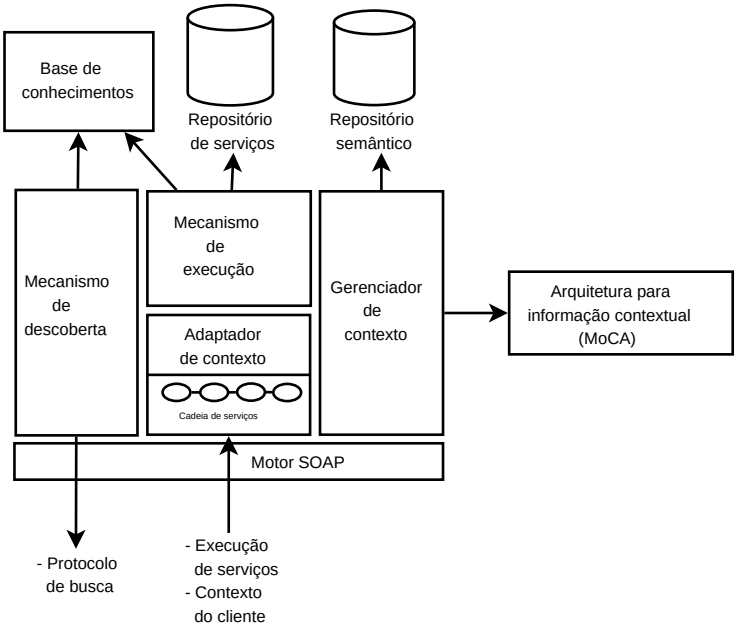

Figura 7.1: Arquitetura do middleware sugerido.

Na Figura 7.1 se apresenta uma visão geral prévia dos componentes do middleware. O módulo *Adaptador de contexto* é o encarregado de processar a informação contextual proveniente de algum cliente e adaptar a cadeia de

serviços às condições do contexto. Por exemplo, se a informação contextual do cliente indica que há uma queda na sua largura de banda mas com capacidade de processamento ainda suficiente, então um serviço de compressão de dados seria colocado na cadeia de serviços.

Em cada chamada a um serviço é passada a informação contextual do cliente que fez a chamada. Note-se que o middleware pode receber diversas informações contextuais por cada cliente que chame um serviço. Neste ponto é necessário ter um mecanismo de licitação que permita uma única configuração da cadeia de serviços. Um mecanismo como o adotado pelo middleware *CARISMA* [6] pode ser usado nesta tarefa.

O módulo de *Descoberta de serviços* seria basicamente o mesmo apresentado nesta dissertação. Este módulo implementaria um protocolo que permita a localização de serviços no ambiente. O módulo *Gerenciador de contexto* se encarregaria de coletar informação contextual do dispositivo, de forma autônoma ou acessando uma arquitetura de contexto como o MoCA. Já o *mecanismo de execução* se encarregaria de executar um serviço da mesma maneira como foi apresentado nesta dissertação. Este módulo se encontra sobre o adaptador de contexto fornecendo os resultados da execução à cadeia de serviços.

7.3.2 Filtragem colaborativa para ambientes inteligentes

Nesta proposta pretendemos agregar ao nosso modelo de ambiente inteligente a técnica de *filtragem colaborativa*(*Collaborative Filtering*), permitindo ao ambiente fazer predições automáticas baseadas nos dispositivos e usuários que se encontrem no ambiente.

A filtragem colaborativa é o processo em que são feitas predições para um individuo baseado nas mesmas preferências de um grupo de usuários. A filtragem colaborativa permite automatizar esta tarefa e permite trabalhar com uma grande quantidade de indivíduos.

A implementação desta proposta se basearia no arcabouço *SPREAD* [10] para a implementação de aplicações de computação ubíqua. O *SPREAD* se baseia em representar o mundo físico em um espaço de tuplas. Especificamente, faz a representação do mundo físico associando à tuplas áreas físicas do ambiente. Adicionando a filtragem colaborativa ao modelo computacional do *SPREAD*, podemos desenvolver o que chamamos de *filtragem colaborativa sensível à localização*.

Empregando nosso modelo, um serviço pode implementar um *gerenciador de tuplas* para uma área específica do ambiente. Assim, a descoberta de serviços se reduz a achar o serviço de gerenciador de tuplas adequado

para o dispositivo. Com base na filtragem colaborativa, o gerenciador de tuplas forneceria as tuplas relevantes aplicando esta técnica às tuplas que está gerenciando.

Referências Bibliográficas

[1] Alcañiz M. e Rey B., *New Technologies For Ambient Intelligence*, Ambient Intelligence, IOS Press 2005.

[2] Angele J. e Lausen G., *Ontologies in F-Logic*, Handbook on Ontologies in Information Systems. International Handbooks on Information Systems 29-50, Springer Verlag, 2004.

[3] Burbey I., *Ubiquitous Internet Computing*, WWW Beyond the Basics (`http://ei.cs.vt.edu/book/index.html`), Prentice Hall, 1998.

[4] Bussler C., Maedche A. e Fensel D., *A Conceptual Architecture for Semantic Web Enabled Web Services*, ACM Special Interest Group on Management of Data: Volume 31, Number 4, 2002.

[5] Bussler C., Maedche A. e Fensel D., *Web Services: Quo Vadis*, IEEE Intelligent Systems, 2003.

[6] Capra L., Emmerich W. e Mascolo C.. *CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications*, IEEE Transactions on Software Engineering, 29(10), 929-945, 2003.

[7] Chan A. T. e Chuang S. N., *Mobipads: A reflective middleware for context-aware mobile computing*, IEEE Trans. Software Eng., 29(12), 1072-1085, 2003.

[8] Chen G. e Kotz D., *A Survey of Context-Aware Mobile Computing Research. Dartmouth Computer*, Science Technical Report TR2000-381. Department of Computer Science - Dartmouth College, 2000.

[9] Chen H., Finin T. e Joshi A., *An Intelligent Broker for Context-Aware Systems*, Adjunct Proceedings of Ubicomp 2003, USA, October, 2003.

[10] Couderc P. e Banâatre M., *Ambient Computing Applications: an Experience with the SPREAD Approach*, Procs. of the 36th Hawaii IntŠl Conf. on System Sciences (HICSSŠ03), IEEE Comp Soc, 2003.

[11] Cortese G., Lunghi M. e Davide F., *Context-Awareness for Physical Service Environments Ambient Intelligence*, IOS Press, 2005.

[12] de Kleer J., *An Assumption-Based TMS*, Artifificial Intelligence, Volume 28, Issue 2, 127-162, 1986.

[13] de Kleer J., *Extending the ATMS*, Artificial Intelligence, Volume 28, Issue 2, 163-196, 1986.

[14] Dey A. K. e Abowd G. D., *Toward a better understanding of context and context-awareness*, GVU Technical Report GIT-GVU-99-22, College of Computing, Georgia Institute of Technology, 1999.

[15] Davies J., Fensel D., e Van Harmelen F., *Towards the Semantic Web: Ontology-Driven Knowledge Management*, John Wiley Sons, 2003.

[16] Ducatel K., Bogdanowicz M., Scapolo F., Leijten J. e Burgelman J-C. *Scenarios for Ambient Intelligence in 2010*, *ISTAG*, Febrary 2001.

[17] Fensel D. e Bussler C., *The Web Service Modeling Framework WSMF*, Electronic Commerce: Research and Applications, 113-137, 2002.

[18] Gagglioli A., *Optimal Experience in Ambient Intelligence*, Ambient Intelligence, IOS Press 2005.

[19] Helal A., Mann W., Elzabadani H., King J., Kaddourah Y. e Jansen E., *Gator Tech Smart House: A Programmable Pervasive Space*, IEEE Computer magazine, March 2005.

[20] Hansmann U., Merk L., Nicklous M. e Stober T., *Pervasing Computing*, Springer Second Edition, 2003.

[21] Issarny V., Sacchetti D., Tartanoglu F., Sailhan F., Chibout R., Levy N. e Talamona A. *Developing Ambient Intelligence Systems: A Solution based on Web Services*, In Journal of Automated Software Engineering. Vol 12. 2005.

[22] IST Advisory Group (ISTAG), *Ambient Intelligence: from Vision to Reality*, Ambient Intelligence, IOS Press 2005.

[23] Jorge T. e Correa da Silva F. S., *SATMS - The Simplified Assumption-based Truth Maintenance System*, Workshop, DCC, IME-USP, 2005.

[24] Kenneth D. F. e de Kleer J., *Building Problem Solvers*, MIT Press, 1993.

[25] Kifer M., Lara R., Polleres A. e Zhao C., *A Logical Framework for Web Services Discovery*, ICWS 2004.

[26] Kifer M. e Lausen G., *F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance, and Scheme*, International Conference on Management of Data archive ACM SIGMOD, 1989.

[27] Kleiner A., *Game AI: The Possible Bridge between Ambient and Artificial Intelligence*, *Ambient Intelligence*, IOS Press, 2005.

[28] LaMarca A., Chawathe Y., Consolvo S., Hightower J., Smith I., Scott J., Sohn T., Howard J., Hughes J., Potter F., Tabert J., Powledge P., Borriello G. e Schilit B., *Place Lab: Device Positioning Using Radio Beacons in the Wild*, In proceedings of Pervasive 2005, Munich, Germany.

[29] Masuoka R., Parsia B. e Labrou Y., *Task Computing - The Semantic Web meets Pervasive Computing*, `http://www.flacp.fujitsulabs.com`, Fujitsu Laboratories of America, Inc., 2004.

[30] OASIS - Organization for the Advancement of Structured Information Standards, *OASIS Reference Model for Service Oriented Architecture V1.0*, Official Committee Specification approved Aug 2, 2006.

[31] OWL Services Coalition, *OWL-S: Semantic Markup for Web Services*, `http://www.daml.org/services/owl-s/1.0/`.

[32] Paolucci M., Srinivasan N. e Sycara K., *Expressing WSMO Mediators in OWL-S*, The Third International Semantic Web Conference, ISWC 2004.

[33] Paolicci M., Kawamura T., Payne T. R. e Sycara K., *Semantic Matching of Web Services Capabilities*, 2004.

[34] Picco G., Murphy A. e Roman G., *Lime: Linda Meets Mobility.* In Proceedings of the 21stInternational Conference on Software Engineering (ICSE'99), Los Angeles, USA, 1999.

[35] Project Oxygen, `http://www.oxygen.lcs.mit.edu/`, site acessado pela última vez 01/06/2007.

[36] Project One.world, `http://www.cs.nyu.edu/rgrimm/one.world/`, site acessado pela última vez 01/06/2007.

[37] Rica G., *The Psychology of Ambient Intelligence: Activity, Situation and Presence*, Ambient Intelligence, IOS Press 2005.

[38] Robertson D., Agusti J., Correa da Silva F. S., Vasconcelos W. W. M e Melo A. C. V., *A Lightweight Capability Communication Mechanism*, 13th. International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, USA, 2000.

[39] Roman D., Lausen H. e Keller U., *Web Service Modeling Ontology (WSMO)*, `http://www.wsmo.org/TR/d2/v1.2/`, Working Draft D2v1.2, April 2005.

[40] Sacramento V., Endler M., Rubinsztejn H. K., Lima L.S., Gonçalves K., do N ascimento F.N. e Bueno G., *MoCA: A Middleware for Developing Collaborative Applications for Mobile Users*, ACM/IFIP/USENIX International Middleware Conference, Toronto, October, 2004.

[41] Schmidt A., *Interactive Context-Aware Systems Interacting with Ambient Intelligence*, Ambient Intelligence, IOS Press, 2005.

[42] Schilit N., *A System Architecture for Context-Aware Mobile Computing*, Phd Teses, Columbia University, 1995.

[43] Schilit N., Adams N. e Want R., *Context-Aware Computing Applications*, IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[44] Singh M. e Huhns M., *Service-Oriented Computing: Semantics, processes and agents*, John Wiley Sons, 2005.

[45] Sousa J. P. e Garlan D., *Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments*, Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, August 25-31, 2002.

[46] Srivastava B. e Koehler J., *Web Service Composition: Current Solutions and Open Problems*, ICAPS 2003 Workshop on Planning for Web Services, 2003.

[47] Strang T. e Linnhoff-Popien C., *A Context Modeling Survey*, Workshop on Advanced Context Modeling, Reasoning and Management as part of UbiComp 2004.

[48] Strang T., Linnhoff-Popien C. e Frank K., *CoOL: A Context Ontology Language to enable Contextual Interoperability* International Conference on Distributed Applications and Interoperable Systems, DAIS2003, 2003.

[49] Xavier E. e Correa da Silva. F. S., *Expressing Systems Capabilities for Knowledge Coordination*, AAMAS'2002.

[50] Yang G., Kifer M., Zhao C. e Chowdhary V., *Flora-2: User's Manual*, Version 0.94, Project Flora-2.

[51] W3C Working Group, *Web Services Architecture*, `http://www.w3.org/TR/ws-arch/`, W3C Working Group Note 11 February 2004.

[52] Wang Z. e Garlan D., *Task-Driven Computing*, Technical Report, School of Computer Science, Carnegie Mellon University, May 2000.

[53] Weiser M., *The Computer for the Twenty-First Century*, Scientific American, pp. 94-10, September 1991.

[54] WSMO Group, *A Conceptual Comparison between WSMO and OWL-S*, `http://www.wsmo.org/TR/d4/d4.1/v0.1/`, Working Draft D4.1v0.1, April 2005.

[55] WSMO Group, *The Web Service Modeling Language WSML*, `http://www.wsmo.org/TR/d16/d16.1/v0.2/`, WSML Final Draft 20 March 2005.

Apêndice

A. Teoria em F-Logic para a descoberta do Serviço Library

```
/* is-a hierarchy*/
bookplace::region.
region::location.
relatedbooks::listbooks.
locationabsolute::location.
coursebooks::listbooks.
locationrelative::location.
student::academic.
teacher::academic.
place::location.
teacherfenix::academic.
academic::person.
studentfenix::student.
ime:place.


/* ontologies definition*/
bookplace[inarea *=> string, instand *=> string,
          inpanel *=> string, isavailable *=> boolean].
listbooks[haselements *=>> book].
department[hascode *=> string, hasname *=> string].
region[].
relatedbooks[].
confirm[hasresponse *=> string].
person[hasid *=> integer, hasname *=> string,
       hasage *=> integer, haslocation *=> location].
```

```
locationabsolute[].
listcourses[hascourse *=>> coursefenix].
coursefenix[hascode *=> string, hasname *=> string,
            hasteacher *=> teacherfenix,
            indepartment *=> department].
coursebooks[].
location[posx *=> integer, posy *=> integer,
         posz *=> integer].
locationrelative[inreference *=> location].
student[].
teacher[].
place[hasid *=> integer, hasname *=> string].
teacherfenix[hascoursefenix *=>> coursefenix].
book[hasname *=> string, hasauthor *=> string,
     hasisbn *=> string, hastheme *=> booktheme,
     inplace *=> bookplace].
booktheme[hasname *=> string, hasdiscipline *=> string].
academic[hasschedule *=> schedule, hasgrade *=> string].
studentfenix[hasnusp *=> string,
             hascoursefenix *=>> coursefenix].


/* instances definition*/
ime:place[hasid->"ime01", hasname->"IME"].


/* axioms ontologies defition*/
(_somelist:listbooks):-(_somebook[hasisbn->_somISBN]
                        and _somebook:book).
(_somebook[hasisbn->_somISBN] and _somebook:book):-
                        (_somelist:listbooks).


/* webservice definition*/
webservice[effects =>> $, assumptions =>> $,
           postcond =>> $,result =>> $, assumptions =>> $ ].
result[service =>> webservice, goals =>> goal].
goal[query => $].
```

```
librarywservice:webservice[
postcond ->> ${_listbooks:relatedbooks :-
              ((_student[haslocation->ime] and
                _student:studentfenix))},
postcond ->> ${_listbooks:coursebooks :-
              ((_student[haslocation->ime] and
                _student:studentfenix))},
assumptions ->> ${(_student[haslocation->ime] and
                   _student:studentfenix)},
result  ->> ${results:result[service ->> librarywservice]}
].

postcond_main(X) :- (_listbooks:relatedbooks) ;
                    ((_listbooks:coursebooks)).
goalRule(Goal) :- Goal:goal[query -> Q],Q.

/* discovery rules */
discovery(Service:webservice,Goal:goal) :-
Service[postcond ->> Post, result ->> Result,
            assumptions ->> Assumptions],
insertrule{ Post },
if goalRule(Goal) then
       insert{ _:result[service ->> Service, goals ->> Goal] },
deleterule{ Effects }.

/* knowledge base facts */
   crhistian:studentfenix[haslocation -> ime].
```

B. Interface WSDL do Servidor do Ambiente

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="AmbientTask" targetNamespace=
                   "http://ambienttask/AmbientTask/"
             xmlns:tns="http://ambienttask/AmbientTask/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:ns2="http://ambienttask/AmbientTask/" >
```

```
<types>
  <schema targetNamespace="http://ambienttask/AmbientTask/"
          xmlns:tns="http://ambienttask/AmbientTask/"
          xmlns="http://www.w3.org/2001/XMLSchema"
                 elementFormDefault="qualified"
          attributeFormDefault="unqualified">

     <complexType name="HostData">
  <sequence>
   <element name="mac" type="xsd:string"/>
  </sequence>
 </complexType>
 <complexType name="LogicalExpression">
  <sequence>
   <element name="le" type="xsd:string"/>
  </sequence>
 </complexType>
 <complexType name="MyIntArray">
  <sequence>
   <element maxOccurs="unbounded" minOccurs="0"
            name="Entry" type="xsd:int"/>
  </sequence>
 </complexType>
 <complexType name="MultipleObjectException">
  <sequence>
   <element name="objects" type="tns:ArrayOfEntryMultiple"/>
  </sequence>
 </complexType>
 <complexType name="ArrayOfEntryMultiple">
  <sequence>
   <element maxOccurs="unbounded" minOccurs="0"
            name="Entry" nillable="true" type="tns:EntryMultiple"/>
  </sequence>
 </complexType>
 <complexType name="EntryMultiple">
  <sequence>
   <element name="params" type="tns:ArrayOfParamInfo"/>
   <element name="ws" type="tns:WebServiceData"/>
  </sequence>
 </complexType>
```

```
<complexType name="ArrayOfParamInfo">
 <sequence>
  <element maxOccurs="unbounded" minOccurs="0"
           name="Entry" nillable="true" type="tns:ParamInfo"/>
 </sequence>
</complexType>
<complexType name="ParamInfo">
 <sequence>
  <element name="method" type="string"/>
  <element name="objs" type="tns:ArrayOfstring"/>
  <element name="oreder" type="int"/>
  <element name="taken" type="int"/>
  <element name="values" type="tns:ArrayOfstring"/>
 </sequence>
</complexType>
<complexType name="ArrayOfstring">
 <sequence>
  <element maxOccurs="unbounded" minOccurs="0"
           name="Entry" type="xsd:string"/>
 </sequence>
</complexType>
<complexType name="WebServiceData">
 <sequence>
  <element name="methods" type="tns:ArrayOfstring"/>
  <element name="urlService" type="string"/>
 </sequence>
</complexType>
<complexType name="ProActiveException">
 <sequence>
  <element name="services" type="tns:ArrayOfEntryPro"/>
 </sequence>
</complexType>
<complexType name="ArrayOfEntryPro">
 <sequence>
  <element maxOccurs="unbounded" minOccurs="0"
           name="Entry" nillable="true" type="tns:EntryPro"/>
 </sequence>
</complexType>
<complexType name="EntryPro">
 <sequence>
```

```
  <element name="methods" type="tns:ArrayOfProActiveInfo"/>
  <element name="ws" type="tns:WebServiceData"/>
 </sequence>
</complexType>
<complexType name="ArrayOfProActiveInfo">
 <sequence>
  <element maxOccurs="unbounded" minOccurs="0"
           name="Entry" nillable="true" type="tns:ProActiveInfo"/>
 </sequence>
</complexType>
<complexType name="ProActiveInfo">
 <sequence>
  <element name="les" type="tns:ArrayOfstring"/>
  <element name="methodName" type="string"/>
  <element name="obj" type="tns:ArrayOfstring"/>
 </sequence>
</complexType>
<complexType name="PositionData">
 <sequence>
  <element name="areaName" type="string"/>
  <element name="x" type="double"/>
  <element name="y" type="double"/>
  <element name="z" type="double"/>
 </sequence>
</complexType>
<complexType name="ArrayOfWebServiceData">
 <sequence>
  <element maxOccurs="unbounded" minOccurs="0" name="Entry"
           nillable="true" type="tns:WebServiceData"/>
 </sequence>
</complexType>
<complexType name="GoalData">
 <sequence>
  <element name="goals" type="tns:ArrayOfstring"/>
 </sequence>
</complexType>
<complexType name="ContextData">
 <sequence>
  <element name="contextinfo" type="tns:ArrayOfstring"/>
 </sequence>
```

```
  </complexType>
 </schema>
</types>
<message name="AmbientInterface_removeKBResponse">
 <part name="le" type="xsd:int"/>
</message>
<message name="AmbientInterface_continueMultipleObject">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="arrayOfint_2" type="tns:MyIntArray"/>
</message>
<message name="AmbientInterface_hasExceptionResponse">
 <part name="result" type="xsd:boolean"/>
</message>
<message name="AmbientInterface_continueProActiveResponse"/>
<message name="AmbientInterface_setContextForHostResponse"/>
<message name="AmbientInterface_continueProActive2">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="ProActiveException_2" type="tns:ProActiveException"/>
</message>
<message name="AmbientInterface_hasException">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="GoalData_2" type="tns:GoalData"/>
</message>
<message name="AmbientInterface_continueMultipleObject2">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="MultipleObjectException_2"
       type="tns:MultipleObjectException"/>
</message>
<message name="AmbientInterface_getPositionResponse">
 <part name="result" type="tns:PositionData"/>
</message>
<message name="AmbientInterface_addFactToKB">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="LogicalExpression_2" type="tns:LogicalExpression"/>
</message>
<message name="AmbientInterface_setContextForHost">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="ContextData_2" type="tns:ContextData"/>
</message>
<message name="AmbientInterface_continueMultipleObject2Response"/>
```

```
<message name="AmbientInterface_continueMultipleObjectResponse"/>
<message name="AmbientInterface_getServiceForGoalResponse">
 <part name="result" type="tns:ArrayOfWebServiceData"/>
</message>
<message name="AmbientInterface_getLastMultipleObjectException">
 <part name="HostData_1" type="tns:HostData"/>
</message>
<message name="AmbientInterface_getLastProActiveExceptionResponse">
 <part name="result" type="tns:ProActiveException"/>
</message>
<message name="AmbientInterface_voidResponse"/>
<message name="AmbientInterface_register">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="ContextData_2" type="tns:ContextData"/>
</message>
<message name="AmbientInterface_getServiceForGoal">
 <part name="HostData_1" type="tns:HostData"/>
</message>
<message name="AmbientInterface_getPosition">
 <part name="HostData_1" type="tns:HostData"/>
</message>
<message name="AmbientInterface_getLastProActiveException">
 <part name="HostData_1" type="tns:HostData"/>
</message>
<message name="AmbientInterface_getKB">
 <part name="result" type="tns:ArrayOfstring"/>
</message>
<message name="AmbientInterface_removeFactToKB">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="LogicalExpression_2" type="tns:LogicalExpression"/>
</message>
<message name="AmbientInterface_registerResponse"/>
<message name="AmbientInterface_continueProActive">
 <part name="HostData_1" type="tns:HostData"/>
 <part name="ProActiveException_2" type="tns:ArrayOfstring"/>
</message>
<message name="AmbientInterface_getLastMultipleObjectExceptionResponse">
 <part name="result" type="tns:MultipleObjectException"/>
</message>
<portType name="AmbientInterface">
```

```
<operation name="continueMultipleObject"
           parameterOrder="HostData_1 arrayOfint_2">
 <input message="tns:AmbientInterface_continueMultipleObject"/>
 <output message="tns:AmbientInterface_continueMultipleObjectResponse"/>
</operation>
<operation name="continueMultipleObject2"
           parameterOrder="HostData_1 MultipleObjectException_2">
 <input message="tns:AmbientInterface_continueMultipleObject2"/>
 <output message="tns:AmbientInterface_continueMultipleObject2Response"/>
</operation>
<operation name="continueProActive"
           parameterOrder="HostData_1 ProActiveException_2">
 <input message="tns:AmbientInterface_continueProActive"/>
 <output message="tns:AmbientInterface_continueProActiveResponse"/>
</operation>
<operation name="continueProActive2"
           parameterOrder="HostData_1 ProActiveException_2">
 <input message="tns:AmbientInterface_continueProActive2"/>
 <output message="tns:AmbientInterface_continueProActiveResponse"/>
</operation>
<operation name="getLastMultipleObjectException"
           parameterOrder="HostData_1">
 <input message="tns:AmbientInterface_getLastMultipleObjectException"/>
 <output message="tns:AmbientInterface_getLastMultipleObjectExceptionResponse"/>
</operation>
<operation name="getLastProActiveException" parameterOrder="HostData_1">
 <input message="tns:AmbientInterface_getLastProActiveException"/>
 <output message="tns:AmbientInterface_getLastProActiveExceptionResponse"/>
</operation>
<operation name="getPosition" parameterOrder="HostData_1">
 <input message="tns:AmbientInterface_getPosition"/>
 <output message="tns:AmbientInterface_getPositionResponse"/>
</operation>
<operation name="getKB" parameterOrder="HostData_1">
 <input message="tns:AmbientInterface_getPosition"/>
 <output message="tns:AmbientInterface_getKB"/>
</operation>
<operation name="addFactToKB"
           parameterOrder="HostData_1 LogicalExpression_2">
 <input message="tns:AmbientInterface_addFactToKB"/>
```

```
  <output message="tns:AmbientInterface_voidResponse"/>
 </operation>
 <operation name="removeFactToKB"
           parameterOrder="HostData_1 LogicalExpression_2">
  <input message="tns:AmbientInterface_removeFactToKB"/>
  <output message="tns:AmbientInterface_removeKBResponse"/>
 </operation>
 <operation name="getServiceForGoal"
           parameterOrder="HostData_1">
  <input message="tns:AmbientInterface_getServiceForGoal"/>
  <output message="tns:AmbientInterface_getServiceForGoalResponse"/>
 </operation>
 <operation name="hasException"
           parameterOrder="HostData_1 GoalData_2">
  <input message="tns:AmbientInterface_hasException"/>
  <output message="tns:AmbientInterface_hasExceptionResponse"/>
 </operation>
 <operation name="register"
           parameterOrder="HostData_1 ContextData_2">
  <input message="tns:AmbientInterface_register"/>
  <output message="tns:AmbientInterface_registerResponse"/>
 </operation>
 <operation name="setContextForHost"
           parameterOrder="HostData_1 ContextData_2">
  <input message="tns:AmbientInterface_setContextForHost"/>
  <output message="tns:AmbientInterface_setContextForHostResponse"/>
 </operation>
</portType>
<binding name="AmbientInterfaceBinding" type="tns:AmbientInterface">
 <soap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="continueMultipleObject">
  <soap:operation soapAction=""/>
  <input>
   <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
  </input>
  <output>
   <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
```

```
 </output>
</operation>
<operation name="continueMultipleObject2">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="continueProActive">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="continueProActive2">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="getLastMultipleObjectException">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
```

```
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="getLastProActiveException">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="getPosition">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="getKB">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="addFactToKB">
```

```
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="removeFactToKB">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="getServiceForGoal">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </output>
</operation>
<operation name="hasException">
 <soap:operation soapAction=""/>
 <input>
  <soap:body namespace="http://ambienttask/AmbientTask/"
             use="literal"/>
 </input>
 <output>
  <soap:body namespace="http://ambienttask/AmbientTask/"
```

```
              use="literal"/>
   </output>
  </operation>
  <operation name="register">
   <soap:operation soapAction=""/>
   <input>
    <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
   </input>
   <output>
    <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
   </output>
  </operation>
  <operation name="setContextForHost">
   <soap:operation soapAction=""/>
   <input>
    <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
   </input>
   <output>
    <soap:body namespace="http://ambienttask/AmbientTask/"
              use="literal"/>
   </output>
  </operation>
 </binding>
 <service name="AmbientTask">
  <port binding="tns:AmbientInterfaceBinding"
        name="AmbientInterfacePort">
   <soap:address location="http://CrhisN:8080/AmbientWS/AmbientTask"/>
  </port>
 </service>
</definitions>
```

C. Interface IDL do motor do ambiente

```
// Engine.idl

module enginelocal {
```

```
typedef sequence<string> methods;
typedef sequence<string> les;
typedef sequence<string> myObjArray;
typedef sequence<short> myIntArray;

enum ServiceType {WIRED, WIRELESS};
struct HostInfo { long id; string name; string ip; string mac;};
struct Position { string area; double x; double y; double z; };
struct WebServiceInfo {string name; string endPoint;
                long position; methods ms; ServiceType servType;};
struct ContextInfo {les les;};
struct Goal {les les;};


typedef sequence<WebServiceInfo> WebServiceSeq;
typedef sequence<string> Objects;


struct ProActiveInfo {string method;les ls; les objs;};
struct ParamInfo {string method; long order; Objects objs; long taken;};

typedef sequence<ProActiveInfo> proMethods;
typedef sequence<ParamInfo> multMethods;


struct HashProItem {WebServiceInfo ws; proMethods methods;};
struct HashMultItem {WebServiceInfo ws; multMethods methods;};

typedef sequence<HashProItem> HashProActive ;
typedef sequence<HashMultItem> HashMultiple ;

     // Exception definitions
exception CreateException { string detail; };
exception RemoveException { string detail; };
exception NotFoundException { };
exception NotPlaneException { string detail; };
exception InconsistentException { string details; };

exception ProActiveException{ HashProActive table;};
exception MultipleObjectsException{HashMultiple table;};
```

```
interface Engine {
        void setPosition(in Position pos);
        boolean setContext(in ContextInfo ctxt) raises(InconsistentException);
        boolean addContext(in ContextInfo ctxt) raises(InconsistentException);
        boolean isConsistent(in ContextInfo ctxt);

        // not change attributes
        HostInfo getHost();

        // web services plane
        WebServiceSeq getServices(in Goal goals)
                      raises(NotPlaneException,
                             ProActiveException,
                             MultipleObjectsException);
        WebServiceInfo nextService() raises(NotPlaneException);

        // web services exceptions
        ProActiveException getLastProActiveException();
        MultipleObjectsException getLastMultipleException();
        WebServiceSeq getLastServices();
        long isAmbientException();

        void continueProActive(in myObjArray objs);
        void continueMultiple(in myIntArray takens);

        //kb access
        myObjArray getKB();
        void addToKB(in string le);
        long removeFromKB(in string le);

        // estimate position
        Position getEstimatePosition();
    };

    interface EngineHome {
            Engine create(in HostInfo host) raises(CreateException);
            Engine createWithContext(in HostInfo host, in ContextInfo ctxt)
                         raises(CreateException);
            void remove(in string mac) raises(RemoveException);
```

```
            Engine findByMAC(in string mac) raises(NotFoundException);
    };
};
```

D. Ontologias definidas

a. Ontologia Library

```
namespace { _"http://ambienttask/ontologies/LibraryOnto#",
     lib _"http://ambienttask/ontologies/LibraryOnto#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     loc _"http://ambienttask/ontologies/LocationOnto#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     util _"http://ambienttask/ontologies/UtilOnto#",
     fex _"http://ambienttask/ontologies/FenixOnto#" }

ontology _"http://ambienttask/ontologies/LibraryOnto"

     importsOntology
          { _"http://ambienttask/ontologies/LocationOnto",
            _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/UtilOnto",
            _"http://ambienttask/ontologies/FenixOnto"}

axiom listaxiom
     definedBy
          ?somebook[hasISBN hasValue ?somISBN] memberOf Book
implies
?somelist memberOf ListBooks.

axiom listaxiom2
     definedBy
          ?somelist memberOf ListBooks
implies
?somebook[hasISBN hasValue ?somISBN] memberOf Book.

concept Book
     nonFunctionalProperties
```

```
          typeWSDL hasValue "Book"
     endNonFunctionalProperties
     hasName impliesType _string
     hasAuthor impliesType _string
     hasISBN impliesType _string
     hasTheme impliesType BookTheme
     inPlace impliesType BookPlace

concept BookPlace subConceptOf loc#Region
     nonFunctionalProperties
          typeWSDL hasValue "BookPosition"
     endNonFunctionalProperties
     inArea impliesType _string
     inStand impliesType _string
     inPanel impliesType _string
     isAvailable impliesType _boolean

concept BookTheme
     nonFunctionalProperties
          typeWSDL hasValue "BookTheme"
     endNonFunctionalProperties
     hasName impliesType _string
     hasDiscipline impliesType _string

concept ListBooks
     nonFunctionalProperties
          typeWSDL hasValue "ListBooks"
     endNonFunctionalProperties
     hasElements impliesType  (1 *) Book

concept RelatedBooks subConceptOf ListBooks
     nonFunctionalProperties
          typeWSDL hasValue "ListBooks"
     endNonFunctionalProperties

concept CourseBooks subConceptOf ListBooks
     nonFunctionalProperties
          typeWSDL hasValue "ListBooks"
     endNonFunctionalProperties
```

```
concept Confirm
     nonFunctionalProperties
          typeWSDL hasValue "boolean"
     endNonFunctionalProperties
     hasResponse impliesType _string
```

b. Ontologia Fenix

```
namespace { _"http://ambienttask/ontologies/FenixOnto#",
     fex _"http://ambienttask/ontologies/FenixOnto#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     util _"http://ambienttask/ontologies/UtilOnto#" }

ontology _"http://ambienttask/ontologies/FenixOnto"

     importsOntology
          { _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/UtilOnto"}

concept CourseFenix
     hasCode impliesType _string
     hasName impliesType _string
     hasTeacher impliesType TeacherFenix
     inDepartment impliesType Department

concept TeacherFenix subConceptOf peo#Academic
     hasCourseFenix impliesType  (1 *) CourseFenix

concept StudentFenix subConceptOf peo#Student
     hasNUSP impliesType _string
     hasCourseFenix impliesType  (1 *) CourseFenix

concept ListCourses
     hasCourse impliesType  (1 *) CourseFenix

concept Department
     hasCode impliesType _string
     hasName impliesType _string
```

c. Ontologia Ambient

```
namespace { _"http://ambienttask/ontologies/AmbientOnto#",
     fex _"http://ambienttask/ontologies/AmbientOnto#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     loc _"http://ambienttask/ontologies/LocationOnto#" }

ontology _"http://ambienttask/ontologies/AmbientOnto"

     importsOntology
          { _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/LocationOnto"}

concept Device
     nonFunctionalProperties
          typeWSDL hasValue "Device"
     endNonFunctionalProperties
     hasMAC impliesType _string
     hasEndPoint impliesType _string

concept User
     nonFunctionalProperties
          typeWSDL hasValue "User"
     endNonFunctionalProperties
     hasId impliesType _string
     hasName impliesType _string

concept Context
     nonFunctionalProperties
          typeWSDL hasValue "Context"
     endNonFunctionalProperties
     hasElements impliesType  (1 *) _string
```

d. Ontologia Location

```
namespace { _"http://ambienttask/ontologies/LocationOnto#",
```

```
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     loc _"http://ambienttask/ontologies/LocationOnto#" }

ontology _"http://ambienttask/ontologies/LocationOnto"
     nonFunctionalProperties
          title hasValue "WSML Location Ontology"
     endNonFunctionalProperties

concept Location
     posX impliesType _integer
     posY impliesType _integer
     posZ impliesType _integer

concept LocationAbsolute subConceptOf Location

concept LocationRelative subConceptOf Location
     inReference impliesType Location

concept Place subConceptOf Location
     hasId impliesType _integer
     hasName impliesType _string

concept Region subConceptOf Location

instance ime memberOf Place
     hasId hasValue "ime01"
     hasName hasValue "Instituto de Matematica e Estatistica"
```

e. Ontologia `Communication`**:**

```
namespace { _"http://ambienttask/ontologies/CommunicationOnto#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     default _"http://ambienttask/ontologies/CommunicationOnto#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     util _"http://ambienttask/ontologies/UtilOnto#" }

ontology _"http://ambienttask/ontologies/CommunicationOnto"
     nonFunctionalProperties
          title hasValue "WSML Communication Ontology"
```

```
     endNonFunctionalProperties

     importsOntology
          { _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/UtilOnto"}

axiom wired_contrain23
     definedBy
          ?somePeople[peo#hasAge hasValue 22, peo#hasName hasValue "Crhisn"]
             memberOf peo#People.

concept Message
     hasId impliesType _integer
     hasText impliesType _string
     To impliesType util#List
     From impliesType peo#Person
     inEnvironment impliesType Environment

concept Environment
     hasId impliesType _string

concept WirelessMessage subConceptOf Message

concept WiredMessage subConceptOf Message

instance wirelessEnvironment memberOf Environment
     hasId hasValue "Wireless01"

instance wiredEnvironment memberOf Environment
     hasId hasValue "Wired01"
```

E. Especificação do serviços implementados

a. Serviço Fenix

```
namespace { _"http://ambienttask/wsami/FenixWService#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     dc _"http://purl.org/dc/elements/1.1#",
     lib _"http://ambienttask/ontologies/LibraryOnto#",
```

```
     loc _"http://ambienttask/ontologies/LocationOnto#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     fex _"http://ambienttask/ontologies/FenixOnto#" }

webService _"http://ambienttask/wsami/FenixWService"
     nonFunctionalProperties
          urlClasses hasValue "http://ambienttask/code/fenix.jar"
     endNonFunctionalProperties

     importsOntology
          { _"http://ambienttask/ontologies/LibraryOnto",
            _"http://ambienttask/ontologies/LocationOnto",
            _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/FenixOnto"}

capability _"http://FenixCapability"

assumption assumption1
     definedBy
          ?user[hasLocation hasValue loc#ime] memberOf peo#Ademic.

precondition precond1
     definedBy
          ?someStudent[hasNUSP hasValue ?someNusp] memberOf fex#StudentFenix.

precondition precond2
     definedBy
          ?someCourse[hasCode hasValue ?someCode] memberOf fex#CourseFenix.

postcondition postcond1
     definedBy
          ?listbooks memberOf lib#CourseBooks.

postcondition postcond2
     definedBy
          ?listcourses memberOf fex#ListCourses.

interface FenixInterface
     choreography _"urn:FenixChoreography"
     nonFunctionalProperties
```

```
          someStudent\-listcourses hasValue "findCoursesStudent:persistent-one"
          someStudent\-listbooks hasValue "findBooksStudent:transient-one"
          someCourse\-listbooks hasValue "findBooksCourse:persistent-one"
     endNonFunctionalProperties
     transitionRules rulesFenix
          if (?someCourse[hasCode hasValue ?someCode]
                  memberOf fex#CourseFenix) then
               add(?listbooks memberOf lib#CourseBooks)
          endIf

          if (?someStudent[hasNUSP hasValue ?someNusp]
                  memberOf fex#StudentFenix) then
               add(?listcourses memberOf fex#ListCourses)
          endIf

          if (?someStudent[hasNUSP hasValue ?someNusp]
                  memberOf fex#StudentFenix) then
               add(?listbooks memberOf lib#CourseBooks)
          endIf
```

b. Serviço Library

```
namespace { _"http://ambienttask/wsami/LibraryWService#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     dc _"http://purl.org/dc/elements/1.1#",
     lib _"http://ambienttask/ontologies/LibraryOnto#",
     loc _"http://ambienttask/ontologies/LocationOnto#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     fex _"http://ambienttask/ontologies/FenixOnto#" }

webService _"http://ambienttask/wsami/LibraryWService"
     nonFunctionalProperties
          urlClasses hasValue "http://ambienttask/code/library.jar"
     endNonFunctionalProperties

     importsOntology
          { _"http://ambienttask/ontologies/LibraryOnto",
            _"http://ambienttask/ontologies/LocationOnto",
            _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/FenixOnto"}
```

```
capability _"http://LibraryCapability"

assumption assumption1
     definedBy
          ?user[hasLocation hasValue loc#ime] memberOf peo#Ademic.

precondition precond1
     definedBy
          ?book[hasISBN hasValue ?isbn] memberOf lib#Book.

precondition precond2
     definedBy
          ?book[hasTheme hasValue ?theme] memberOf lib#Book
  and ?theme memberOf lib#BookTheme.

postcondition postcond1
     definedBy
          ?listbooks memberOf lib#RelatedBooks.

postcondition postcond2
     definedBy
          ?listbooks memberOf lib#CourseBooks.

interface LibraryInterface
     choreography _"urn:LibraryChoreography"
     nonFunctionalProperties
          someTheme\-disciplinebooks hasValue "findBookByTheme:persistent-one"
          someISBN\-relatedbooks hasValue "relatedBooks:persistent-one"
          someBook\-someBook hasValue "takeBook:transient-one"
          someLocation\-relatedbooks  hasValue "booksForPosition:transient-many"
     endNonFunctionalProperties
     transitionRules courseBooks
          if (?someBook[isAvailable hasValue true] memberOf lib#Book) then
               add(?someBook [isAvailable hasValue false]
                      memberOf lib#Book)
          endIf

          if (?someBook[hasISBN hasValue ?someISBN] memberOf lib#Book) then
               add(?relatedbooks memberOf lib#RelatedBooks)
```

```
          endIf

          if (?someTheme memberOf lib#BookTheme) then
               add(?disciplinebooks memberOf lib#AvailableBooks)
          endIf

          if (?someStudent[hasLocation hasValue ?someLocation]
                  memberOf fex#StudentFenix) then
               add(?relatedbooks memberOf lib#RelatedBooks)
          endIf
```

b. Serviço `Location`

```
namespace { _"http://ambienttask/wsami/LocationWService#",
     rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     dc _"http://purl.org/dc/elements/1.1#",
     lib _"http://ambienttask/ontologies/LibraryOnto#",
     loc _"http://ambienttask/ontologies/LocationOnto#",
     peo _"http://ambienttask/ontologies/PeopleOnto#",
     fex _"http://ambienttask/ontologies/FenixOnto#" }

webService _"http://ambienttask/wsami/LocationWService"
     nonFunctionalProperties
          urlClasses hasValue "http://ambienttask/code/location.jar"
     endNonFunctionalProperties

     importsOntology
          { _"http://ambienttask/ontologies/LibraryOnto",
            _"http://ambienttask/ontologies/LocationOnto",
            _"http://ambienttask/ontologies/PeopleOnto",
            _"http://ambienttask/ontologies/FenixOnto"}

capability _"http://LocationCapability"

precondition precond1
     definedBy
          ?somebook memberOf lib#Book.

postcondition postcond1
     definedBy
```

```
          ?direction memberOf loc#Direction.

interface LocationInterface
     choreography _"urn:LocationChoreography"
     nonFunctionalProperties
          someBook\-someDirection hasValue "goItBook:transient-many"
     endNonFunctionalProperties
     transitionRules rulesL
          if (?someBook[hasIsbn hasValue ?someisbn] memberOf lib#Book
  and ?someStudent[hasLocation hasValue ?somelocation]
         memberOf fex#StudentFenix) then
           add(?someDirection memberOf loc#Direction)
      endIf
```

F. Estado do ATMS para o ambiente da biblioteca

```
<?disciplinebooks memberOf AvailableBooks,
{
{(?someTheme memberOf BookTheme,findBookByTheme,1)}
}>

<?relatedbooks memberOf RelatedBooks,
{
{(?someBook[hasISBN hasValue ?someISBN] memberOf Book,relatedBooks,1)},
{(?someStudent[hasLocation hasValue ?someLocation] memberOf StudentFenix,
                                                    booksForPosition,1)},
{(?someCourse[hasCode hasValue ?someCode] memberOf CourseFenix,
                                          findBooksCourse,2)},
{(?someStudent[hasNUSP hasValue ?someNusp] memberOf StudentFenix,
                                           findBooksStudent,2)}
}>

<?listbooks memberOf ListBooks,
{
{(?someCourse[hasCode hasValue ?someCode] memberOf CourseFenix,
                                          findBooksCourse,2)}
{(?someStudent[hasNUSP hasValue ?someNusp] memberOf StudentFenix,
                                           findBooksStudent,2)}
}>
```

```
<?someDirection memberOf Direction,
{
{(?someBook[hasIsbn hasValue ?someisbn] memberOf Book,goItBook,1),
 (?someStudent[hasLocation hasValue ?somelocation] memberOf StudentFenix,
                                                   goItBook,1) },
{(?someStudent[hasLocation hasValue ?somelocation] membeOf StudentFenix,
                                                   goItBook,1),
 (?someCourse[hasCode hasValue ?someCode] memberOf CourseFenix,
                                          findBooksCourse,2) },
{(?someStudent[hasLocation hasValue ?somelocation] memberOf StudentFenix,
                                                   goItBook,1),
 (?someStudent[hasNUSP hasValue ?someNusp] memberOf StudentFenix,
                                           findBooksStudent,2) }
}>

<?someBook[isAvailable hasValue false] memberOf Book ,
{
{(?someBook[isAvailable hasValue true] memberOf Book,takeBook,1) }
}>

<?listcourses memberOf ListCourses,
{
{(?someStudent[hasNUSP hasValue ?someNusp] memberOf StudentFenix,
                                           findCoursesStudent,2) }
}>
```