UNIVERSIDADE CATÓLICA DE PELOTAS
ESCOLA DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Sensibilidade ao Contexto na Computação Pervasiva: Avaliando o Uso de Ontologias

por
João Ladislau Barbará Lopes

Trabalho Individual -
TI-2006/2-06

Orientador: Prof. Dr. Adenauer Corrêa Yamin

Pelotas, dezembro de 2006

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CASS	Context-Awareness Sub-Structure
CoBrA	Context Broker Architecture
CORTEX	CO-operating Real-time senTient objects: architecture and EXperimental evaluation
DAML	DARPA Markup Language
EXEHDA	Execution Environment for Highly Distributed Applications
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HTML	Hiper Text Markup Language
ISAM	Infra-estrutura de Suporte às Aplicações Móveis Distribuídas
KIF	Knowledge Interchange Format
OCML	Operational Conceptual Modelling Language
OIL	Ontology Inference Layer
OML	Ontology Markup Language
OWL	Web Ontology Language
XOL	Ontology Exchange Language
PDA	Personal Digital Assistant
RACER	Renamed Abox and Concept Expression Reasoner
RDF	Resource Description Framework
RFID	Radio Frequency Identification
RICE	RACER Interactive Client Environment
SHOE	Simple HTML Ontology Extensions
SOCAM	Service-oriented Context-Aware Middleware
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
SQL	Structured Query Language

STEAM	Scalable Timed Events And Mobility
W3C	World Wide Web Consortium
XML	Extensible Markup Language

RESUMO

A computação pervasiva se caracteriza por propiciar ao usuário acesso a seu ambiente computacional independente de localização, tempo e equipamento. Pesquisas recentes apontam que esta proposta pode ser construída pela integração da computação móvel, computação em grade e computação sensível ao contexto. O escopo geral deste trabalho é a computação pervasiva, e em particular as temáticas pertinentes à sensibilidade ao contexto. Seu objetivo central é avaliar o uso de ontologias na qualificação dos mecanismos utilizados para expressar informações de contexto. Assim, pretende-se identificar as possibilidades de atingir melhores níveis de descrição nas informações que caracterizam o contexto do ambiente computacional pelo emprego de uma semântica de maior expressividade que a usualmente praticada na coleta e no tratamento dos dados obtidos.

Palavras-chave: Computação pervasiva, computação sensível ao contexto, ontologias.

TITLE: “CONTEXT-AWARENESS IN PERVASIVE COMPUTING: EVALUATING THE USE OF ONTOLOGIES”

ABSTRACT

The pervasive computing is characterized by providing to the user access his computational environment independent of location, time and equipment. Recent researches point that this proposal can be built by the integration of the mobile computing, grid computing and context-aware computing. The general target of this work is the pervasive computing, and in particular the pertinent themes to the context-awareness. The central objective is to evaluate the ontologies use in the qualification of the mechanisms used to express context information. Thus, it intends to identify the possibilities to reach better description levels in the information that characterize the context of the computacional environment by the use of a semantics of larger expressiveness that the usually practiced in the collection and in the treatment of the obtained data.

Keywords: pervasive computing, context-aware computing, ontologies.

1 INTRODUÇÃO

A computação pervasiva é a proposta de um novo paradigma que permite ao usuário o acesso a seu ambiente computacional independente de localização, tempo e equipamento. O termo computação pervasiva ficou associado à IBM em função de uma edição do IBM *System Journal* que recebeu o título de *Pervasive Computing*. Nesta edição, todos os artigos trataram de aspectos promissores da computação pervasiva, dentre eles um artigo de Mark Weiser que resgatava sua visão para o futuro da computação, na qual recursos de computação onipresentes se ajustariam, de forma autônoma, para atender os usuários.

A proposta original de Weiser (WEISER, 1991) relativa à computação ubíqua ainda está distante de uma aplicação prática baseada em produtos de mercado (SATYANARAYANAN, 2001). Porém, sua proposta vem gradativamente sendo concretizada através da disponibilização de dispositivos móveis (PDAs, *SmartPhones*) e da consolidação de padrões para redes sem fio (*Bluetooth*, *Wi-Fi*).

O recente trabalho de (YAMIN, 2004) afirma que a proposta da computação pervasiva pode ser construída pela integração da computação móvel, computação em grade e computação sensível ao contexto. Em um ambiente de computação pervasiva, os dispositivos, serviços e agentes devem ser conscientes de seus contextos e automaticamente adaptar-se às suas mudanças, isso caracteriza a sensibilidade ao contexto.

A computação sensível ao contexto vem despertando muita atenção dos pesquisadores desde sua proposta em (SCHILIT; THEIMER, 1994). Diversos sistemas de sensibilidade ao contexto foram desenvolvidos para demonstrar a utilidade desta tecnologia, entretanto serviços de sensibilidade ao contexto nunca foram amplamente disponibilizados para os usuários e a construção de sistemas sensíveis ao contexto é ainda uma tarefa complexa e altamente consumidora de tempo, devido à falta de uma apropriada infra-estrutura para suporte. Percebe-se, então, que a sensibilidade ao contexto e a conseqüente adaptação ao mesmo constitui uma área na qual as pesquisas ainda não se aprofundaram suficientemente.

Um mecanismo apropriado para sensibilidade ao contexto deve prover suporte para tarefas, tais como: obtenção do contexto de diversas fontes, como sensores físicos, bancos de dados e agentes; execução de interpretação do contexto e disseminação do contexto para partes interessadas de um modo distribuído e pontual. Para suporte a estas tarefas é necessário um modelo de mecanismo de sensibilidade ao contexto muito bem estabelecido.

Uma questão relevante na sensibilidade ao contexto é o grau de expressividade que se pode obter na descrição dos possíveis estados do mesmo. Neste aspecto, o uso

de ontologias contribui para qualificar os mecanismos de sensibilidade ao contexto, em função da elevada expressividade que o uso destas pode propiciar.

As ontologias vem sendo utilizadas por várias áreas da Ciência da Computação, principalmente com o intuito de dotar os sistemas de meta-conhecimento. A utilização de ontologias para descrição semântica de um determinado vocabulário proporciona um entendimento amplo das características e propriedades das classes pertencentes a um domínio, assim como seus relacionamentos (FLEISCHMANN, 2004).

Este capítulo apresenta o tema do trabalho individual, destacando as motivações e objetivos, bem como descreve a estrutura do texto como um todo.

1.1 Tema

Este trabalho individual tem como enfoque principal a avaliação do uso de ontologias na qualificação dos mecanismos de sensibilidade ao contexto da computação pervasiva.

A computação pervasiva tem como requisito a manipulação de diferentes contextos de execução. Um dos aspectos deste tema, a sensibilidade ao contexto, é considerada um dos grandes desafios desta área de pesquisa.

Assim, o tema deste trabalho abrange estudos que visam avaliar o emprego de ontologias na qualificação dos mecanismos utilizados para expressar informações de contexto, através do estabelecimento da relação existente entre computação pervasiva, sensível ao contexto e ontologias.

1.2 Motivação

As previsões observadas especialmente em (SATYANARAYANAN, 2001) e (SAHA; MUKHERJEE, 2003) apontam que os próximos anos serão caracterizados por uma significativa elevação dos níveis de mobilidade, heterogeneidade e interação entre dispositivos conectados a redes globais.

As primeiras pesquisas envolvendo sistemas distribuídos em redes de grande abrangência, responderam diversas questões pertinentes ao gerenciamento de recursos físicos. Por sua vez, trabalhos mais recentes abordam o tratamento da heterogeneidade, porém não se aprofundam em aspectos pertinentes à sensibilidade ao contexto e a decorrente adaptação ao mesmo (AUGUSTIN; YAMIN; GEYER, 2002).

A premissa central na computação pervasiva consiste em permitir ao usuário o acesso ao seu ambiente de trabalho a partir de qualquer lugar, a qualquer tempo, usando vários tipos de dispositivos (móveis ou não), contemplando funcionalidades que prevêem uma mobilidade física (equipamentos e/ou usuários) e de software. Nesta proposta, a aplicação ou o ambiente de execução pró-ativamente monitoram e controlam as condições do contexto e a aplicação reage às alterações no contexto através de um processo de adaptação (YAMIN, 2004).

A computação pervasiva é um paradigma computacional bastante novo. Embora sendo uma área recente de pesquisa, ela é considerada por muitos como o novo paradigma computacional do século XXI. Neste sentido, cabe salientar que a computação pervasiva foi apontada pela SBC (Sociedade Brasileira de Computação) com um dos cinco grandes desafios para a pesquisa na área de computação para os próximos 10 anos, conforme

consta no documento ”Grandes Desafios da Pesquisa em Computação no Brasil - 2006 - 2016”.

Deste cenário decorre a principal motivação para este trabalho individual, o qual pretende avaliar o emprego de ontologias na qualificação dos mecanismos de sensibilidade ao contexto da computação pervasiva.

1.3 Objetivos

O objetivo geral deste trabalho individual é explorar a correlação entre computação pervasiva, sensível ao contexto e ontologias, avaliando o emprego de ontologias na qualificação dos mecanismos utilizados para expressar informações de contexto.

Os objetivos específicos são:

- estudar os fundamentos teóricos sobre computação pervasiva, computação sensível ao contexto e ontologias;
- revisar os principais projetos em computação pervasiva e computação sensível ao contexto;
- caracterizar os mecanismos de sensibilidade ao contexto utilizados em ambientes de execução para computação pervasiva;
- compreender a correlação entre computação pervasiva, sensível ao contexto e ontologias;
- estudar métodos para manipulação de ontologias;
- identificar ferramentas para geração e manipulação de ontologias;
- avaliar o emprego de ontologias na qualificação dos mecanismos de sensibilidade ao contexto da computação pervasiva.

1.4 Estrutura do texto

O texto é composto por oito capítulos. O capítulo 2 apresenta os fundamentos teóricos e os principais projetos relacionados à computação pervasiva. No capítulo 3 são conceituados aspectos fundamentais da computação sensível ao contexto e descritos os principais projetos relacionados à essa área. O capítulo 4 apresenta os fundamentos teóricos relacionados com ontologias, bem como descreve aspectos relativos ao projeto de ontologias, destacando linguagens e ferramentas voltadas à construção de ontologias. Ainda, trata da representação do conhecimento e raciocínio baseados na lógica de descrições. A relação entre a computação pervasiva, sensível ao contexto e ontologias é explorada no capítulo 5. Por fim, são apresentadas as considerações finais.

2 COMPUTAÇÃO PERVASIVA

2.1 Fundamentos teóricos

A computação pervasiva é um paradigma computacional que visa fornecer uma computação ”onde se deseja, quando se deseja, o que se deseja e como se deseja”, através da virtualização de informações, serviços e aplicações. Este ambiente computacional é constituído de uma grande variedade de dispositivos de diversos tipos, móveis ou fixos, aplicações e serviços interconectados, refletindo uma computação altamente dinâmica e distribuída.

A combinação de recursos oferecidos pela computação em grade, computação móvel e computação sensível ao contexto, cria a infra-estrutura para aplicações da computação pervasiva (YAMIN, 2004). Com isso, vislumbra-se uma realidade onde a computação passa a ser parte integrante do espaço físico do usuário. A computação é tratada não apenas na perspectiva de dispositivos, mas também como ambiente computacional que envolve o usuário. Busca-se um cenário no qual os recursos computacionais (ambientes, dados, aplicações) estarão disponíveis em qualquer local e todo o tempo no âmbito do ambiente pervasivo (AUGUSTIN, 2003).

A construção da proposta da computação pervasiva fundamenta-se em cinco elementos básicos, representados na figura 2.1, são eles (YAMIN, 2004):

- rede: elemento responsável por estabelecer uma conexão física e lógica entre o usuário e recursos disponíveis. Interconecta recursos de redes estruturadas e sem fio, permitindo assim, que os demais requisitos sejam atendidos;

- elevada heterogeneidade: representa a diversidade de dispositivos computacionais disponíveis ao usuário, tais como: notebooks, PDAs, smartphones, desktops, estações de alto desempenho, clusters, sistemas operacionais diversos;

- mobilidade: o deslocamento do usuário e dos dispositivos presentes no cenário pervasivo deve ser previsto, bem como a mobilidade do software também deve ser contemplada;

- disponibilidade: semântica SIGA-ME, ou seja, independência de equipamento, lugar ou tempo. Isso significa que uma aplicação pode ser executada em diversos dispositivos, ou seja, que independente da plataforma utilizada, dados ou serviços solicitados pelo usuário devem se manter disponíveis;

- adaptação: adaptação ao contexto é condição necessária à computação pervasiva e deve envolver tanto as aplicações como o ambiente de execução. Isso significa que

as condições de contexto são pró-ativamente monitoradas e o suporte à execução deve permitir que tanto a aplicação como ele próprio utilizem essas informações na gerência da adaptação de seus aspectos funcionais e não-funcionais.

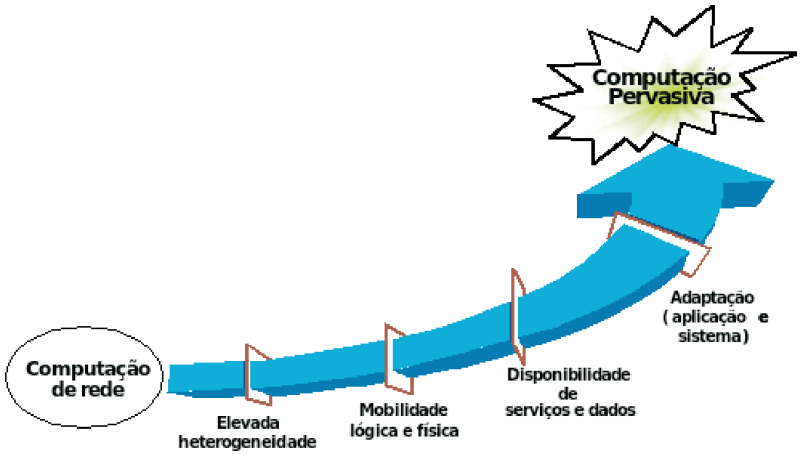

Figura 2.1: Elementos Básicos para Construção da Computação Pervasiva

Desde que Weiser concebeu sua visão de ubiqüidade, importantes evoluções no hardware tem sido obtidas, permitindo a criação de dispositivos menores e mais portáteis, sensores e dispositivos de controle com crescente poder de processamento e a padronização das tecnologias para comunicação sem fio. Com isso, estão sendo criadas as condições para permitir a premissa básica da computação pervasiva, ou seja, o acesso do usuário ao seu ambiente computacional a qualquer hora, em qualquer lugar, independente de dispositivo. Apesar dos significativos avanços observados, para que a computação pervasiva se torne uma realidade precisam ser vencidos alguns desafios:

- interfaces de usuário devem suportar diferentes modalidades, bem como prever e antecipar a intenção do usuário;
- serviços distribuídos devem ser adaptados aos usuários e suas tarefas e às trocas dinâmicas do estado do ambiente. Também devem contemplar a descoberta dinâmica de serviços e recursos;
- infra-estruturas devem ser dinamicamente configuradas, antecipando a ação/tarefa do usuário. Também, devem ser consideradas restrições impostas pelo ambiente, tais como: conexão à rede intermitente e imprevisível, baixa capacidade de armazenamento e processamentos dos dispositivos, alta possibilidade de perdas e furtos dos dispositivos, tarefas computacionais que consomem muita energia (bateria).

2.2 Projetos em computação pervasiva

A grande maioria dos projetos trata as questões necessárias à pervasividade de forma independente. Existem *middlewares* para atender diversas aspectos, tais como: sistemas distribuídos, comunicação, adaptação, reconhecimento do contexto, gerenciamento de recursos, computação em grade e outros. Nesta seção procura-se abordar *Middlewares* voltados para a execução de aplicações e criações de um ambiente pervasivo completo.

2.2.1 Projeto Gaia

O projeto Gaia, outra infra-estrutura baseada em middleware, estende os conceitos típicos de sistema operacional para incluir sensibilidade ao contexto. Ele visa o suporte ao desenvolvimento e à execução de aplicações portáveis para espaços ativos. Gaia exporta serviços para consultar e utilizar recursos existentes, para acessar e usar o contexto atual e provê um *framework* para desenvolver aplicações centradas no usuário, consciente de recursos, multi-dispositivos, sensível ao contexto e móvel. A figura 2.2 mostra o sistema, constituído basicamente pelo *kernel* do Gaia e pelo *framework* de aplicações (ROMAN et al., 2002).

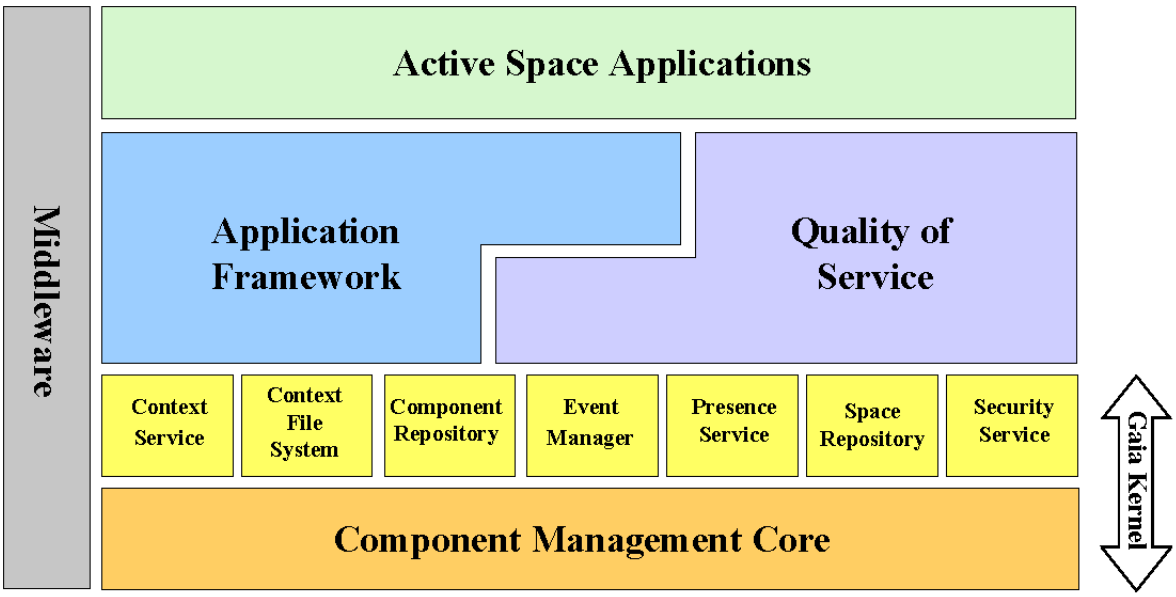

Figura 2.2: Arquitetura do Gaia

As partes do sistema Gaia que abrangem a sensibilidade ao contexto são: *Event Manager*, *Context Service* e *Context File System*. No Gaia o contexto é representado de uma maneira especial, são usados predicados da seguinte forma: *Context(*$< ContextType >, < Subject >, < Relater >, < Object >$*)* escritos em DAML+OIL. O *ContextType* corresponde ao tipo de contexto que o predicado está descrevendo, o *Subject* é a pessoa, lugar ou coisa com a qual o contexto está interessado e o *Object* é um valor associado com o *Subject*. O *Relater* relaciona o *Subject* e o *Object* através de operadores de comparação, um verbo ou preposição. Um exemplo para uma instância de contexto é: *Context(temperature, room 201, is, 20 C)*. Esta sintaxe é usada tanto para representação de contexto como para formação de regras de inferência.

O serviço de gerenciamento de eventos (*Event Manager*) é responsável pela distribuição dos eventos no espaço ativo e implementa um modelo de comunicação desacoplado baseado em produtores, consumidores e canais. Cada canal tem um ou mais produtores que provêem informação para o canal e um ou mais consumidores que recebem a informação. A confiabilidade é aumentada já que os produtores são permutáveis. Com a ajuda do *Context Service* aplicações podem pesquisar por informações de contexto específicas e elevar o nível de abstração com objetos de contexto. Por fim, o *Context File System* faz armazenamento pessoal automático, disponível na presente localização do usuário. Ele constrói uma hierarquia de diretórios virtuais para representar o contexto

como diretórios, onde o componente *path* representa tipos de contextos e valores.

O processamento de contexto no Gaia é ocultado no módulo de serviço de contexto (*Context Service)* permitindo a criação de objetos de contexto de alto nível pela execução de operações lógicas, tais como: quantificação, implicação, conjunção, disjunção e negação de predicados de contexto. Um exemplo de uma regra é: Context(Número de pessoas, Sala 501, >, 4) AND Context(Aplicação, PowerPoint, is, Running) ⇒ Context(Atividade Social, Sala 501, is, Apresentação).

2.2.2 Projeto Aura

O projeto Aura (figura 2.3)é uma proposta que visa projetar, implementar, empregar e avaliar sistemas de larga escala para demonstrarem o conceito de ”aura de informação pessoal”que se espalha pelas diversas infra-estruturas computacionais. É um grande projeto que investiga novas arquiteturas para o ambiente pervasivo. Seu foco é no usuário, suas tarefas e preferências (GARLAN; STEENKISTE; SCHMERL, 2002).

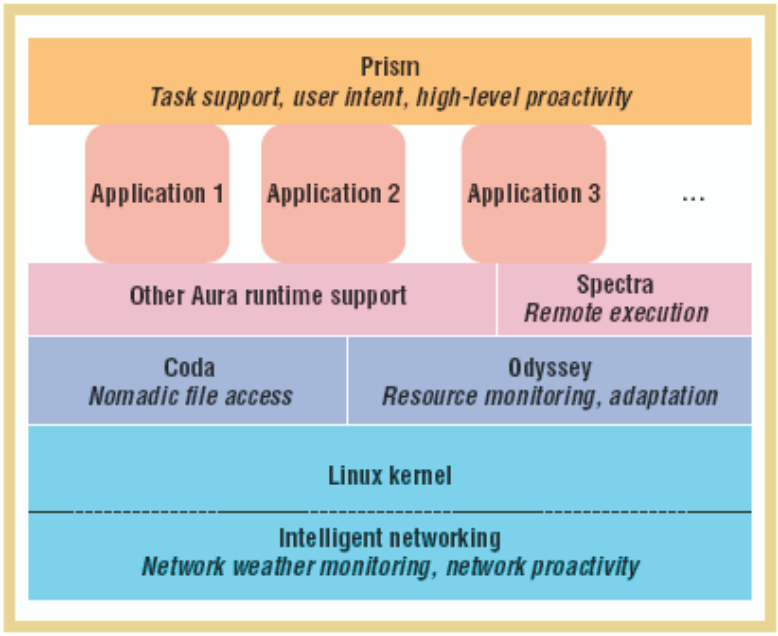

Figura 2.3: Arquitetura do Projeto Aura

Desta forma, o conceito de contexto embutido em Aura prevê a ênfase na dimensão pessoal. Aura visa dar suporte computacional a cenários de aplicações como: ”Fred está em seu escritório preparando um encontro no qual deve fazer uma apresentação e demonstração de software. A sala do encontro é a 10 minutos de onde se encontra. No horário do encontro, Fred ainda não está pronto. Ele pega seu Palm e caminha para a porta. Aura transfere o estado do seu trabalho do desktop para o Palm, e permite a ele terminar a apresentação usando comandos de voz enquanto se desloca. Aura infere onde Fred está indo com base em informações de seu calendário e seu deslocamento físico. Aura carrega a apresentação e o software de demonstração no computador de projeção, e inicializa o projetor”.

A arquitetura de software Aura, para atender este cenário, é composta de:

1. observador de contexto pessoal que interpreta o contexto físico do usuário e identifica localização, antecipa o movimento do usuário e identifica o foco da atenção

deste;

2. gerente de tarefas que mantém a representação da tarefa e mapeia entre contexto e preferências do usuário;

3. gerente do ambiente que conhece o ambiente computacional e pode descobrir e montar componentes para completar a tarefa, também reconhece os recursos disponíveis e avisa quando há troca no ambiente do usuário.

Estes componentes trabalham juntos para atender as tarefas de Fred. O observador de contexto reconhece que Fred está em seu escritório. O gerente de tarefas nota que sua preferência é entrada de dados via teclado. O gerente de ambiente é responsável por encontrar os componentes que fornecem tais serviços. Quando Fred começa a se deslocar (como notado pelo observador de contexto), o serviço que melhor preenche as necessidades de entrada de texto é o ditado (como notado pelo gerente de tarefas). Aura faz uma escolha: reconhece a entrada via fala no Palm, a qual tem capacidade limitada de vocabulário, e transmite a voz para um servidor remoto, que pode tornar-se indisponível conforme Fred se desloca. O gerente de ambiente escolhe outros componentes para atender Fred, caso isto ocorra.

Os desafios do projeto incluem (a) inferência de tarefas; (b) representação das tarefas; (c) alocação de recursos (GARLAN; STEENKISTE; SCHMERL, 2002). O foco num novo modelo de programação é a inovação do Aura.

2.2.3 Projeto Oxygen

O projeto Oxygen (OXYGEN, 2004) é desenvolvido pelo Laboratório de Ciência da Computação e Inteligência Artificial do MIT (Massachusetts Institute of Technology), e tenta atingir os objetivos da computação pervasiva, através da combinação de tecnologias de ”usuário”e tecnologias de ”sistema”. Tecnologias de ”usuário”atendem diretamente as necessidades humanas. De acordo com as premissas do projeto, tecnologias como Visão e Fala, permitem-nos interagir diretamente com Oxygen, como se estivéssemos interagindo, com uma outra pessoa qualquer, poupando muito tempo e esforço. A automatização, o acesso individualizado ao conhecimento, e as tecnologias da colaboração ajudam a executar uma grande variedade de tarefas.

Para suportar atividades humanas altamente dinâmicas e variadas, o Oxygen deve superar diversos desafios técnicos:

- pervasivo: precisa estar em todo lugar, com cada portal atingindo a mesma base de informações;

- embarcado: deve viver em nosso mundo, sentindo e afetando-o;

- mobilidade: precisa permitir ao usuário e seus ambientes computacionais mover-se livremente para qualquer lugar, de acordo com suas necessidades;

- adaptável: precisa prover flexibilidade e espontaneidade, em resposta às mudanças das necessidades dos usuários e das condições operacionais;

- poderoso e eficiente: precisa livrar-se das restrições impostas pelos limites de hardware, indo ao encontro somente das restrições de sistemas impostas pelas necessidades dos usuários e fonte de energia disponível ou largura de banda para comunicações;

- intencional: deve permitir às pessoas nomear serviços e objetos de software de acordo com a intenção, por exemplo, ”a impressora mais próxima”, ao invés de referir-se pelo endereço de um dispositivo;
- eterno: nunca deverá parar ou rebootar; componentes poderão ir e vir de acordo com a demanda de erros e upgrades, mas o sistema Oxygen como um todo deverá estar disponível e operacional todo o tempo.

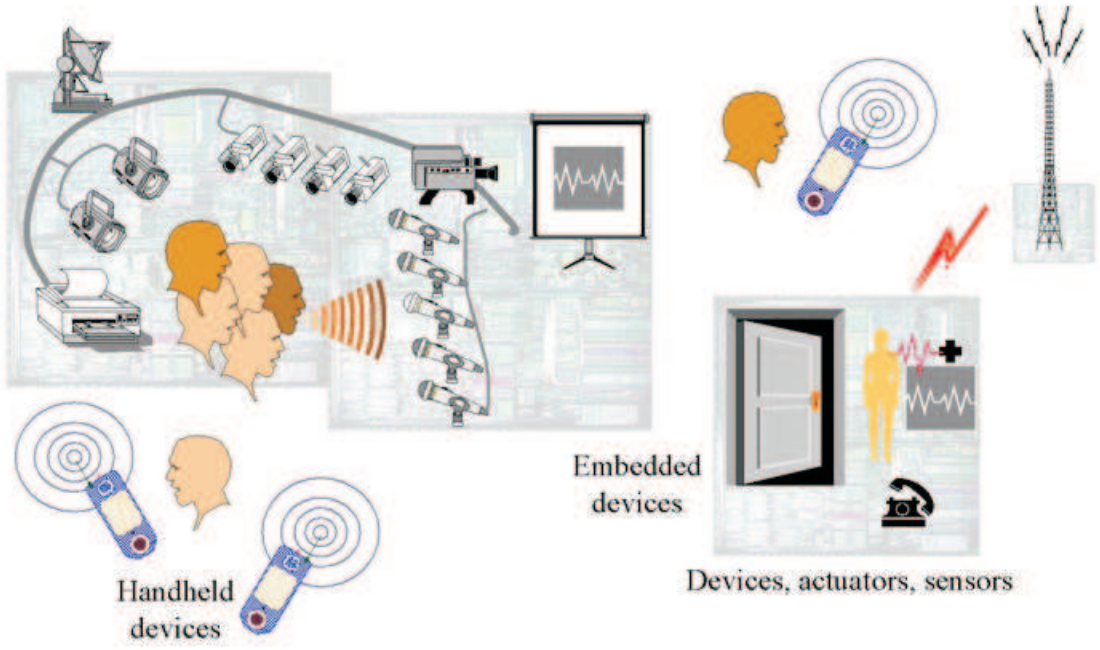

Figura 2.4: Ambientes do Oxygen

2.2.4 Projeto ISAM/EXEHDA

O foco do projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis Distribuídas) é a infra-estrutura de suporte necessária para a implementação das aplicações móveis distribuídas com comportamento adaptativo em um ambiente da computação pervasiva.

O EXEHDA (Execution Environment for Highly Distributed Applications) é um middleware que integra o projeto ISAM, sendo direcionado às aplicações distribuídas, móveis e conscientes do contexto da computação pervasiva (YAMIN, 2004). A figura 2.5 mostra os serviços do EXEHDA, organizados em quatro grandes subsistemas: execução distribuída, adaptação, comunicação e acesso pervasivo.

O suporte à adaptação no EXEHDA está associado à operação do subsistema de reconhecimento de contexto e adaptação, como mostra a figura 2.6. Este subsistema inclui serviços que tratam desde a extração da ”informação bruta”sobre as características dinâmicas e estáticas dos recursos que compõem o ambiente pervasivo, passando pela identificação em alto nível dos elementos de contexto, até o disparo das ações de adaptação em reação a modificações no estado de tais elementos de contexto.

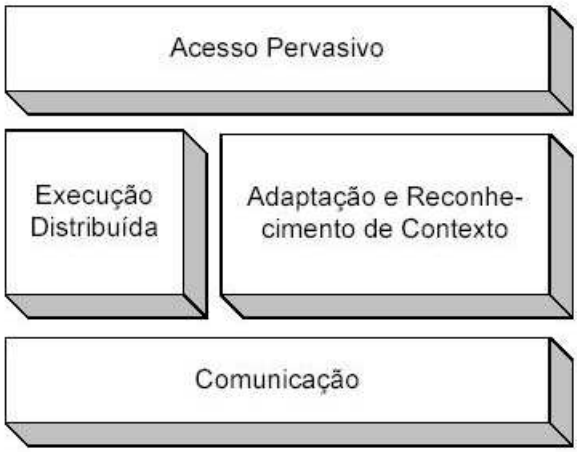

Figura 2.5: Subsistemas do EXEHDA

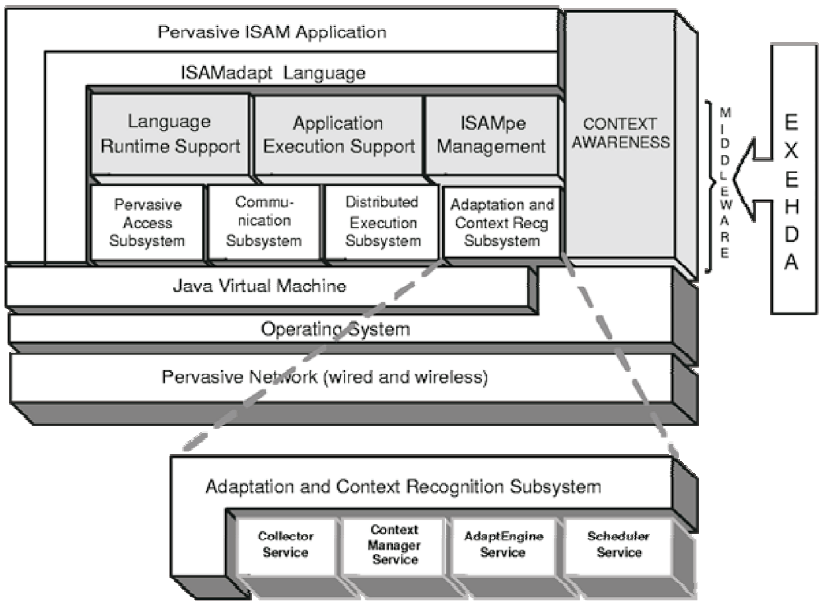

Figura 2.6: Subsistema de Reconhecimento de Contexto e Adaptação do EXEHDA

3 COMPUTAÇÃO SENSÍVEL AO CONTEXTO

A computação sensível ao contexto é um paradigma computacional que se propõe a permitir que as aplicações tenham acesso e tirem proveito de informações que digam respeito às computações que realizam, buscando otimizar seu processamento. Está relacionada com a habilidade dos sistemas computacionais obter vantagem das informações ou condições existentes em um ambiente dinâmico para adicionar valor aos serviços ou executar tarefas mais complexas. Assim, além de lidar com entradas explícitas, a computação sensível ao contexto também considera informação de contexto capturadas por meio de sensores, ou seja, entradas implícitas, tais como: localização, recursos e infra-estrutura disponíveis, preferências do usuário, atividade do usuário, número de dispositivos, tipo de dispositivo, carga computacional (CHEN; KOTZ, 2000).

A história dos sistemas computacionais sensíveis ao contexto começa em 1992 com a aplicação proposta por (WANT et al., 1992). O Active Badge Location System é considerado uma das primeiras aplicações sensíveis ao contexto. Esse sistema, baseado na tecnologia de infravermelho, conseguia determinar a localização atual do usuário, a qual era usada para encaminhar as ligações telefônicas para o telefone mais próximo. A figura 3.1 mostra exemplos de dispositivos Active Badges. Alguns trabalhos a respeito de sensibilidade à localização foram desenvolvidos ao longo dos anos 90, como se pode ver em (ABOWD et al., 1997), nesse período a localização do usuário era o atributo de contexto mais usado.

A localização é um aspecto-chave para os sistemas com mobilidade, pois a localidade influi significativamente no contexto disponibilizado para os mecanismos de adaptação. O contexto representa uma abstração peculiar da computação pervasiva, e inclui informações sobre recursos, serviços e outros componentes do meio físico de execução. Nesta ótica, devem ser obtidas outras informações de contexto que extrapolam a localização onde o componente móvel da aplicação se encontra (YAMIN, 2004).

Nos últimos anos, a computação sensível ao contexto tem recebido maior atenção, principalmente em função do desenvolvimento da computação móvel e do aparecimento de uma nova geração de dispositivos móveis. Porém, a construção do suporte à sensibilidade ao contexto para as aplicações apresenta inúmeros desafios, os quais se relacionam especialmente a obtenção, modelagem, armazenamento, distribuição e monitoramento do contexto.

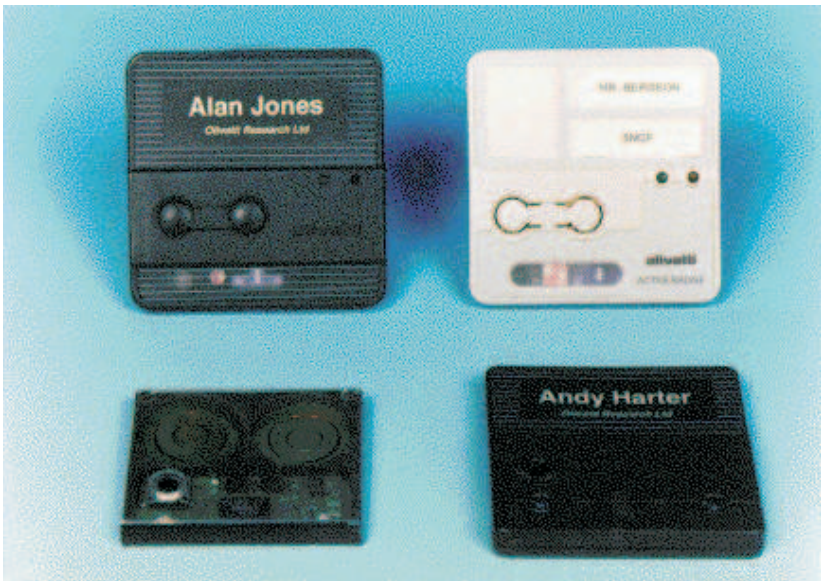

Figura 3.1: Exemplos de Dispositivos Active Badges

3.1 Definição de contexto

Na literatura o termo *context-aware* (sensibilidade ao contexto) apareceu pela primeira vez em (SCHILIT; THEIMER, 1994). Neste trabalho o contexto é descrito como o local, as identidades de pessoas próximas e os objetos e mudanças para esses objetos. (RYAN; PASCOE; MORSE, 1997) refere-se a contexto como a localização do usuário, o ambiente, a identidade e o tempo. (DEY, 1998) define contexto como o estado emocional do usuário, o foco de atenção, a localização e a orientação, a data e o tempo, os objetos e as pessoas no ambiente do usuário. (HULL; NEAVES; BEDFORD-ROBERTS, 1997) descreve contexto como os aspectos da situação corrente.

O contexto é definido em (YAMIN et al., 2003) como toda a informação relevante para a aplicação que pode ser obtida da infra-estrutura computacional, cuja alteração em seu estado dispara um processo de adaptação na aplicação. Nessa visão, o contexto permite enfocar os aspectos relevantes para uma situação particular e ignorar outros. A aplicação explicitamente identifica e define as entidades que caracterizam uma situação e essas passam a integrar o seu contexto.

Uma das mais citadas definições é a encontrada em (DEY; ABOWD, 2000), segundo o autor entende-se por contexto qualquer informação que pode ser usada para caracterizar a situação de uma entidade, entendendo-se por entidade uma pessoa, um lugar ou um objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo os próprios usuário e aplicação.

Segundo (CHEN; KOTZ, 2000) o contexto na computação móvel tem dois diferentes aspectos. Um dos aspectos inclui as características do ambiente circundante que determina o comportamento das aplicações móveis. O outro aspecto é relevante para as aplicações, porém não é crucial. Ele não é necessário para as aplicações se adaptarem a um segundo tipo de contexto, exceto para os exibir a usuários interessados. Baseado nessa consideração o autor define contexto como o conjunto de estados e configurações do ambiente que ou determina um comportamento da aplicação ou no qual um evento da aplicação ocorre e é interessante para o usuário.

3.2 Classificação do contexto

Uma forma de classificação é através da distinção entre as diferentes dimensões do contexto. Em (PREKOP; BURNETT, 2003) e (GUSTAVSEN, 2002) essas dimensões são denominadas de externa e interna, de forma similar, (HOFER et al., 2002) refere-se a contexto físico e lógico. A dimensão externa (ou física) significa que o contexto pode ser mensurado através de sensores de hardware, por exemplo, localização, luz, som, movimento, temperatura. Por sua vez, a dimensão interna (ou lógica) é especificada pelo usuário ou capturada monitorando a interação do usuário. Essa dimensão inclui objetivos do usuário, tarefas, contexto de trabalho, processos de negócio, estado emocional do usuário.

A maioria dos sistemas sensíveis ao contexto faz uso dos fatores de contexto externo, já que eles provêem dados úteis, tais como, informação de localização. Além disso, atributos externos são fáceis de serem monitorados devido a grande disponibilidade e facilidade de acesso a tecnologias de sensoriamento.

Exemplos de uso de atributos lógicos podem ser encontrados em (BUDZIK; HAMMOND, 2000), cujo projeto provê ao usuário informações relevantes obtidas a partir do acesso a páginas Web e documentos, trabalhando na perspectiva de descoberta de recursos e agentes de informação.

Quando lidamos com contexto podem ser identificadas entidades, tais como: lugares (salas, prédios), pessoas (indivíduos, grupos) e coisas (objetos físicos, recursos computacionais) (DEY; SALBER; ABOWD, 2001). Cada uma destas entidades pode ser descrita através de vários atributos, como: identidade (cada entidade tem um único identificador), localização (uma entidade possui uma localização e proximidades), status (corresponde às propriedades intrínsecas de cada entidade, por exemplo, temperatura e iluminação de uma sala, processos sendo executados em um dispositivo) e tempo (usado para precisamente definir a situação, ordenação de eventos).

3.3 Aplicações sensíveis ao contexto: arquitetura

Aplicações sensíveis ao contexto podem ser implementadas de diversas formas. A abordagem dependerá de requisitos e condições, tais como: a localização dos sensores, o número de usuários, a disponibilidade de recursos dos dispositivos utilizados, a facilidade para extensão do sistema. Com base nessas considerações três abordagens de arquiteturas para sistemas sensíveis ao contexto podem ser identificadas (CHEN, 2004):

- **Acesso Direto ao Sensor**: o software cliente obtêm a informação desejada diretamente do sensor, ou seja, não há qualquer camada adicional para obtenção e processamento dos dados do sensor. Esse aspecto dificulta a capacidade de expansão do sistema, por isso essa abordagem que tem sido pouco utilizada. Também, não é adequada para sistemas distribuídos, devido a natureza de seu acesso direto sem qualquer componente capaz de gerenciar múltiplos acessos concorrentes ao sensor.

- **Baseado em Middleware**: os modernos projetos de software usam métodos de encapsulação para separar a lógica de negócios da interface do usuário. A abordagem baseada em middleware introduz uma arquitetura em camadas para sistemas sensíveis ao contexto com a intenção de esconder os detalhes de baixo nível relativos à sensibilidade. Comparando com a abordagem de acesso direto ao sen-

sor, esta técnica facilita a expansão do sistema, já que não há necessidade de modificações no código do cliente, bem como há uma simplificação na reutilização do código dependente de hardware, devido à rígida encapsulação.

- **Servidor de Contexto**: esta abordagem distribuída especializa a arquitetura baseada em middleware introduzindo um componente para o gerenciamento remoto de acesso. Os dados obtidos dos sensores são movidos para um servidor de contexto com o objetivo de facilitar múltiplos acessos concorrentes. Além do reuso dos sensores, a utilização de um servidor de contexto tem a vantagem de retirar dos clientes operações que necessitam uso intensivo de recursos computacionais. Este é um aspecto importante, visto que a maioria dos dispositivos de borda usados em sistemas sensíveis ao contexto são dispositivos móveis com poder computacional limitado. Por outro lado, ao projetar um sistema sensível ao contexto baseado em arquitetura de cliente-servidor, é preciso levar em consideração o uso de protocolos apropriados, analisar o desempenho de rede e avaliar parâmetros de qualidade de serviço.

A separação entre a obtenção e o uso de um contexto é necessária para melhorar a capacidade de expansão e reutilização dos sistemas. Assim, uma arquitetura abstrata poderia incluir camadas para obtenção e uso do contexto através da adição de funcionalidades de interpretação e raciocínio. Na proposta de (AILISTO et al., 2002) tal arquitetura é constituída das seguintes camadas: sensores, recuperação de dados, pré-processamento, armazenamento-gerenciamento e aplicação.

Com relação à primeira camada, constituída pelos sensores, é importante destacar que esta não se refere somente a hardware, mas também a qualquer origem de dados que podem prover informações de contexto passíveis de utilização. Em função do modo como os dados são capturados, os sensores podem ser classificado em três grupos (INDULSKA; SUTTON, 2003):

- **Sensores Físicos**: é o tipo de sensor usado mais freqüentemente. Os sensores de hardware atualmente disponíveis são capazes de capturar praticamente qualquer dado físico. Alguns exemplos de sensores físicos são: sensores de calor, sensores de raios infra-vermelho e ultra-violeta, câmeras, microfones, sistema de posicionamento global (GPS), sistema global para comunicações móveis (GSM), sistema de identificação ativa, sensores de toque, termômetros.

- **Sensores Virtuais**: a origem das informações de contexto é um software. Isso significa que é possível determinar, por exemplo, a localização de um funcionário não somente através do uso de sistemas de localização com sensores físicos, mas também através de sensores virtuais que geram informações de localização, tais como: calendário eletrônico, sistema de reservas para viagens e e-mails. Outros atributos de contexto que podem ser obtidos por sensores virtuais incluem, por exemplo, a atividade do usuário através do movimento do mouse ou da digitação em um teclado.

- **Sensores Lógicos**: esses sensores fazem uso de um conjunto de fontes de informação, eles combinam sensores físicos e virtuais com informação adicional obtida em bancos de dados. Por exemplo, um sensor lógico pode ser construído para detectar a posição atual de um funcionário através da análise dos logins em microcomputadores e de um banco de dados mapeando os dispositivos fixos.

A segunda camada é responsável pela recuperação de dados brutos do contexto. Ela faz uso de drivers adequados para sensores físicos e APIs (Application Programming Interface) para sensores virtuais e lógicos. A funcionalidade de consulta é geralmente implementada através de componentes de software reutilizáveis que tornam transparente o acesso ao hardware, escondendo os detalhes de baixo nível através da disponibilização de métodos abstratos. Através do uso de interfaces para os componentes responsáveis por tipos equivalentes de contextos, estes componentes tornam-se intercambiáveis. Então é possível, por exemplo, substituir um sistema RFID (Radio Frequency Identification) por um sistema GPS sem maiores modificações.

A camada de pré-processamento é responsável pelo raciocínio e interpretação sobre o contexto. Os sensores consultados na segunda camada freqüentemente retornam dados técnicos que não são adequados para uso pelas aplicações, então esta camada eleva os resultados da segunda camada para um maior nível de abstração. As transformações incluem operações de extração e totalização. Por exemplo, a posição exata de uma pessoa dada pelas coordenadas de um GPS pode não ser valiosa para uma aplicação, mas o nome da rua e o número do prédio pelo qual a pessoa está passando é possivelmente uma informação mais importante.

Em sistemas sensíveis ao contexto constituídos por muitas e diferentes fontes de dados, um contexto único pode ser combinado nesta camada passando para um nível mais elevado de informação. Esse processo é chamado agregação ou composição. O valor de um sensor único geralmente não é importante para uma aplicação, a combinação de informações pode ser muito mais valiosa. Neste sentido, um sistema é capaz de determinar, por exemplo, se um cliente está situado dentro ou fora de um ambiente analisando diversos dados físicos, como temperatura e luz ou se a pessoa está em uma reunião capturando nível de ruído e localização. Para fazer essa análise corretamente diversos métodos estatísticos são envolvidos e geralmente algum tipo de fase de treinamento é requerida.

Certamente, essa funcionalidade de abstração poderia ser implementada diretamente pela aplicação. Porém, devido a diversas razões parece ser melhor encapsular essa tarefa e deixá-la sob responsabilidade do servidor de contexto. A encapsulação permite o reuso e facilita o desenvolvimento de aplicações clientes. Também, a performance da rede melhora, visto que os clientes tem que enviar somente uma requisição para obter os dados, ao invés de conectar-se a vários sensores. Ainda, é possível preservar os recursos computacionais, geralmente limitados, dos clientes.

Os problemas de conflitos relativos a obtenção da informação de contexto que podem ocorrer quando existem diferentes fontes de dados também são solucionadas nesta camada. Por exemplo, quando um sistema é notificado sobre a localização de uma pessoa através das coordenadas de seu telefone móvel e também pela filmagem por uma câmera dessa pessoa, pode ser difícil decidir qual informação usar. Geralmente esse conflito é solucionado pelo uso de dados adicionais, tais como, informações precisas de data e hora.

A quarta camada organiza os dados obtidos e os oferece através de uma interface para o cliente. O acesso dos clientes pode ocorrer de dois modos: síncrono a assíncrono. No modo síncrono o cliente faz requisições contínuas (*polling*) ao servidor buscando mudanças do contexto, através de métodos de chamadas remotas, ou seja, o cliente envia uma mensagem ao servidor de contexto requisitando algum tipo de informação e fica aguardando até receber a resposta. Por sua vez, o modo assíncrono trabalha através de subscrições, ou seja, no inicio do programa o cliente assina eventos específicos de seu

interesse. Quando um destes eventos ocorre o cliente é notificado pelo servidor.

Na maioria dos casos o modo assíncrono é mais satisfatório devido as rápidas mudanças no contexto. A técnica de *polling* do modo síncrono consome mais recursos, visto que a informação de contexto tem que ser requisitada muito freqüentemente e a aplicação tem que comprovar as mudanças usando algum tipo de histórico de contexto.

A quinta camada contempla o cliente, é a camada de aplicação. A reação aos diferentes eventos e as instâncias de contexto é implementada nesta camada. Algumas vezes a recuperação de informação e o raciocínio e gerenciamento de contexto específicos da aplicação são encapsulados na forma de agentes que se comunicam com o servidor de contexto e agem como uma camada adicional entre o pré-processamento e a camada de aplicação (CHEN, 2004).

3.4 Projetos em computação sensível ao contexto

O desenvolvimento de aplicações sensíveis ao contexto pode ser realmente simplificado com o uso de infra-estruturas (*frameworks*) genéricas que proporcionem acesso ao cliente para a recuperação de informações de contexto e permitam o registro de novas fontes de dados de contexto heterogêneas e distribuídas. Nesta seção diferentes projetos de *frameworks* para sensibilidade ao contexto são apresentados e comparados em seus principais aspectos.

3.4.1 Arquitetura

A abordagem mais comum para *frameworks* sensíveis ao contexto distribuídos é a clássica infra-estrutura hierárquica com um ou mais componentes centralizados usando uma arquitetura em camadas. Esta abordagem é útil para superar as limitações de recursos computacionais da maioria dos dispositivos móveis.

A figura 3.2 apresenta a arquitetura do *Context Managing Framework* proposta por (KORPIPää et al., 2003). Essa arquitetura é constituída pelas seguintes entidades: gerenciador de contexto, servidores de recurso, serviços de reconhecimento de contexto e aplicação.

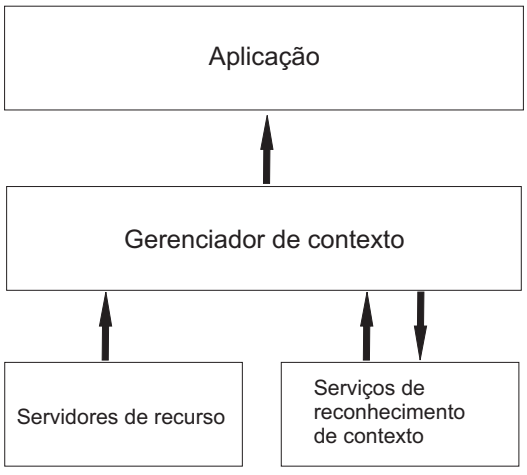

Figura 3.2: Arquitetura do *Context Managing Framework*

Enquanto os servidores de recurso e os serviços de reconhecimento de contexto

são componentes distribuídos, o gerenciador de contexto representa um servidor centralizado, gerenciando um *blackboard*, ou seja, ele armazena dados de contexto e disponibiliza informação para as aplicações cliente.

O projeto SOCAM - *Service-oriented Context-Aware Middleware* é uma arquitetura para construção e prototipação de serviços móveis sensíveis ao contexto. Essa arquitetura usa um servidor central, chamado interpretador de contexto, o qual obtém os dados de contexto através de provedores de contexto distribuídos e oferece essas informações de uma forma processada para os clientes. Os serviços móveis sensíveis ao contexto estão localizados no topo da arquitetura. Esses serviços usam diferentes níveis de contexto e adaptam seu comportamento de acordo com o contexto atual (GU; PUNG; ZHANG, 2004). A figura 3.3 apresenta a arquitetura do SOCAM.

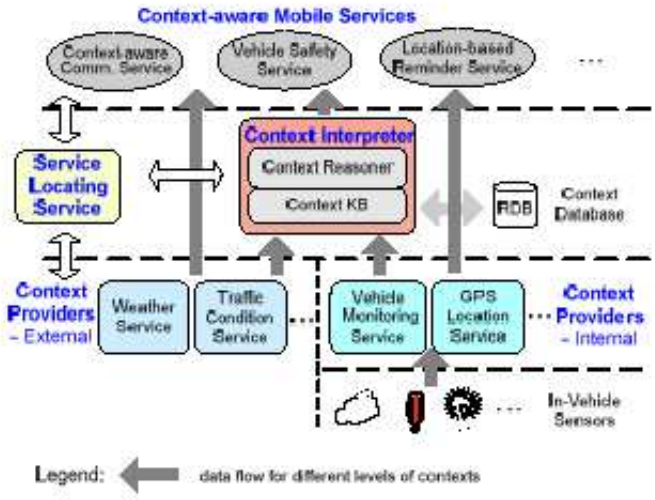

Figura 3.3: Arquitetura do SOCAM

O projeto CASS - *Context-Awareness Sub-Structure* propõe uma abordagem baseada em um middleware centralizado e expansível projetado para aplicações móveis sensíveis ao contexto (FAHY; CLARKE, 2004). A figura 3.4 mostra a arquitetura do CASS.

O *SensorListener* ”escuta”por atualizações dos sensores os quais estão localizados em computadores distribuídos chamados nodos sensores. Então, a informação obtida é armazenada em um banco de dados. O *ContextRetriever* é responsável pela recuperação das informações de contexto armazenadas. Ambos podem usar os serviços de um interpretador. O *ChangeListener* é um componente com capacidade de comunicação que permite um computador móvel ”escutar”por notificações de eventos que provocam mudança de contexto. As classes *Sensor* e *LocationFinder* também possuem capacidade de comunicação. Clientes móveis conectam-se ao servidor através de redes *wireless*. Para reduzir o impacto de intermitência nas conexões existe suporte para um cache local no lado cliente.

CoBrA - *Context Broker Architecture* é uma arquitetura baseada em agentes para suporte à computação sensível ao contexto em espaços inteligentes. Espaços inteligentes são ambientes físicos, tais como: salas, veículos, escritórios e salas de reuniões nos quais são inseridos sistemas inteligentes que proporcionam serviços típicos da computação pervasiva para os usuários. É central para o CoBrA a presença de um negociador de con-

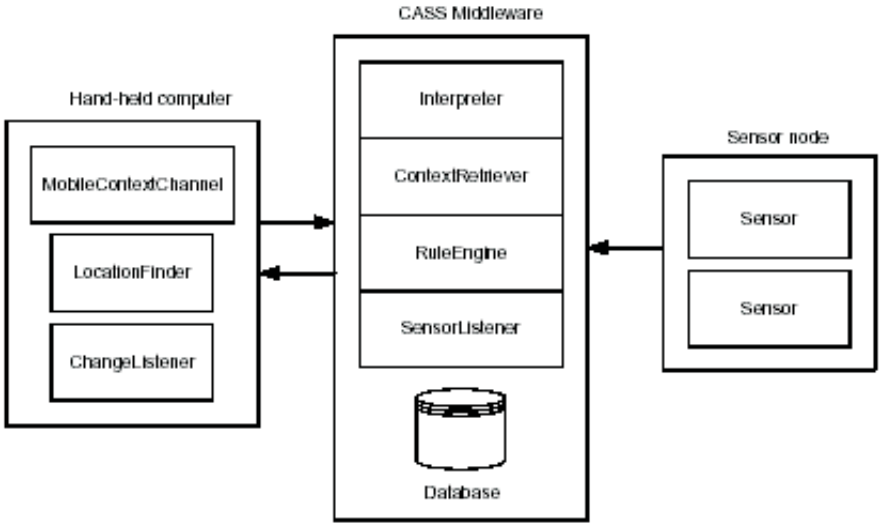

Figura 3.4: Arquitetura do CASS

texto inteligente que mantém e gerencia um modelo de contexto compartilhado ao lado de uma comunidade de agentes. Estes agentes podem ser aplicações hospedadas em dispositivos móveis que um usuário leva ou usa (telefone celular, PDA, fone de ouvido), serviços que são providos por dispositivos em uma sala (projetor multimídia, controlador de iluminação, controlador de temperatura) e serviços Web que provêem uma presença Web para pessoas, lugares e coisas do mundo físico (serviços que mantém rastro de pessoas e paradeiro de objetos). O Negociador de Contexto (*Context Broker*) é constituído de quatro componentes funcionais: base de conhecimento de contexto, motor de inferência de contexto, módulo de aquisição de contexto e módulo de gerenciamento de privacidade (CHEN, 2004). A figura 3.5 mostra a arquitetura do CoBrA.

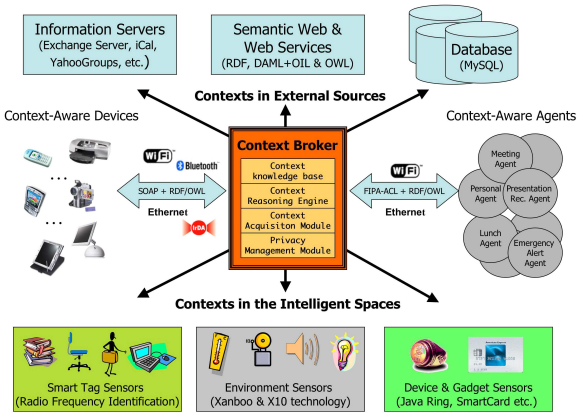

Figura 3.5: Arquitetura do CoBrA

O *Context Toolkit* é um *framework* sensível ao contexto que vai em direção a arquitetura *peer-to-peer* mas ainda necessita de um serviço de descoberta centralizado onde sensores distribuídos, interpretadores e agregadores são registrados para serem encontrados pelas aplicações clientes. A API orientada a objetos provê uma superclasse chamada *BaseObject* que propicia habilidades de comunicações genéricas para facilitar a criação

dos próprios componentes (DEY; SALBER; ABOWD, 2001). A figura 3.6 apresenta um diagrama de objetos para as abstrações do *Context Toolkit*

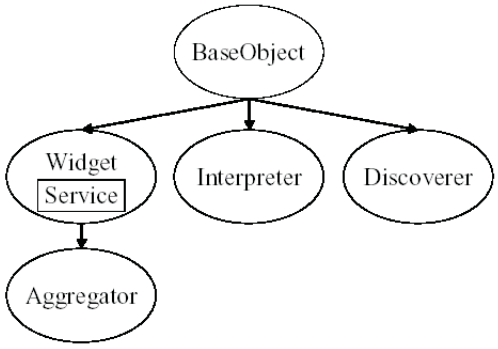

Figura 3.6: Abstrações do *Context Toolkit*

Outro *framework* baseado em uma arquitetura em camadas é construído no projeto *Hydrogen* (HOFER et al., 2002). A abordagem de aquisição de contexto é especializada para dispositivos móveis. Enquanto na maioria dos sistemas distribuídos sensíveis ao contexto o trabalho de um componente centralizado é essencial, o sistema *Hydrogen* tenta evitar essa dependência. Ele faz distinção entre um contexto remoto e um contexto local. Um contexto remoto corresponde a informação sobre outro dispositivo, por sua vez um contexto local é o conhecimento do contexto próprio do dispositivo. Quando os dispositivos estão fisicamente próximos, eles são capazes de trocar seus contextos de uma forma *peer-to-peer* através de rede local, Bluetooth, etc. Essa troca de informação de contexto entre dispositivos clientes é chamada de compartilhamento de contexto. A figura 3.7 mostra o gerenciamento do contexto de um dispositivo, o qual é constituído por seu próprio contexto local e um conjunto de contextos remotos obtidos de outros dispositivos. Ambos, contexto local e remoto, são constituídos de objetos de contexto. A superclasse *ContextObject* é estendida por diferentes tipos de contexto, tais como: *LocationContext*, *DeviceContext*. Essa abordagem permite a simples adição de novos tipos de contexto através da especialização do *ContextObject*.

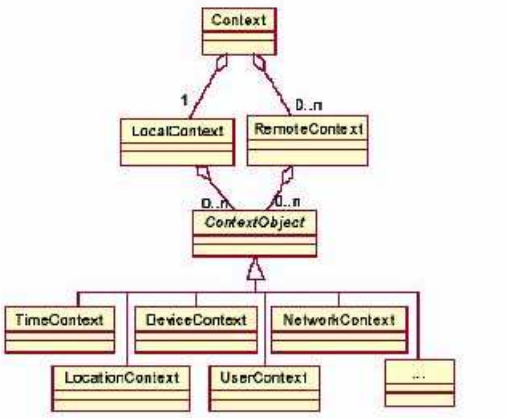

Figura 3.7: Gerenciamento de Contextos Local e Remoto do *Hydrogen*

A figura 3.8 mostra a arquitetura do projeto *Hydrogen*, a qual é constituída por

três camadas que estão localizadas no mesmo dispositivo. A camada de adaptadores é responsável pela recuperação de dados brutos do contexto através de sensores de consulta. Esta camada permite o uso simultâneo de um sensor por diferentes aplicações. A segunda camada (camada de gerenciamento) faz uso da camada de adaptadores para obter dados do sensor e é responsável pelo provimento e recuperação de contextos. O servidor de contexto oferece a informação armazenada, através de modos síncronos ou assíncronos, para as aplicações clientes. No topo da arquitetura está a camada de aplicação onde o código é implementado para reagir a mudanças específicas de contexto reportadas pelo gerenciador de contexto. Devido a independência da plataforma e da linguagem, toda a comunicação entre as camadas é baseada em protocolo XML.

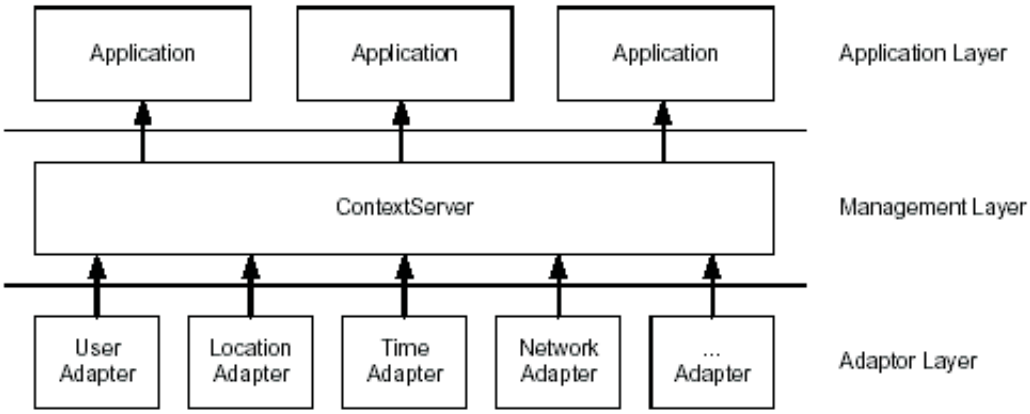

Figura 3.8: Arquitetura do *Hydrogen*

O projeto CORTEX (CO-operating Real-time senTient objects: architecture and EXperimental evaluation) é uma abordagem baseada em middleware para sistemas sensíveis ao contexto. Sua arquitetura é fundamentada no Modelo de Objeto Sensível (BIEGEL; CAHILL, 2004), o qual foi projetado para o desenvolvimento de aplicações sensíveis ao contexto em ambientes móveis *ad-hoc*. A adequação do modelo para aplicações móveis depende do uso do STEAM (Scalable Timed Events And Mobility), um middleware de serviços baseado em eventos para sensibilidade à localização, especificamente projetado para ambientes de rede *ad-hoc* sem fio. A figura 3.9 mostra o modelo de objeto sensível do CORTEX.

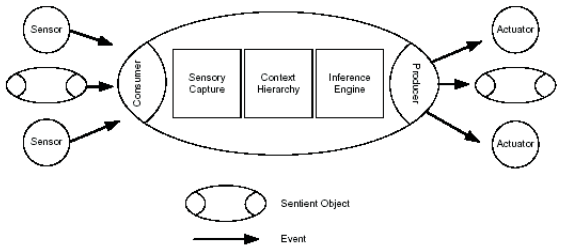

Figura 3.9: Modelo de Objeto do CORTEX

Um objeto sensível é uma entidade encapsulada constituída de três partes principais: sensoriamento, hierarquia de contexto e motor de inferência. Por *interfaces* um

objeto sensível comunica-se com sensores os quais produzem eventos de software e com atuadores que consomem eventos. A figura 3.9 mostra que objetos sensíveis podem ser produtores e consumidores de outro objeto sensível. Os próprios sensores e atuadores são programados usando o STEAM. Para construir objetos sensíveis uma ferramenta gráfica de desenvolvimento está disponível, permitindo aos desenvolvedores especificar sensores e atuadores relevantes, definir redes de fusão, especificar hierarquia de contexto e produzir regras; sem a necessidade de escrever qualquer código.

3.4.2 Modelagem de contexto

Um modelo eficiente para manipulação, compartilhamento e armazenamento de dados de contexto é essencial para os sistemas sensíveis ao contexto.

O *Context Toolkit* manipula o contexto através de tuplas com atributos e valores que são codificados usando XML. *Hydrogen* usa uma abordagem orientada a objetos para modelagem do contexto, com uma superclasse denominada *ContextObject* que oferece modelos abstratos para converter dados XML em objetos de contexto e vice-versa.

Abordagens mais avançadas baseadas em ontologias para modelagem de contexto são encontradas nos *frameworks* SOCAM, CoBrA e Context Managing. Os autores do SOCAM dividem um domínio de computação pervasiva em vários sub-domínios, tais como: casa, escritório. Também definem ontologias individuais em cada sub-domínio para reduzir a complexidade do processamento de contexto. Cada uma destas ontologias implementada em OWL (Web Ontology Language) provê um vocabulário especial usado por representar e compartilhar conhecimento de contexto.

CoBrA também usa uma abordagem baseada em uma ontologia própria, desenvolvida com OWL, denominada CoBrA-Ont (CHEN, 2004). Abaixo um exemplo de parte da CoBrA-Ont:

<loc:LocationContext>
<rdf:type rdf:resource=”InstantThing”/>
<loc:locationContextOf>
<per:Person>
<per:name rdf:datatype=”string”>Harry Chen</per:name>
</per:Person>
</loc:locationContextOf>
<loc:boundedWithin rdf:resource=”Japan”/>
<tme:at rdf:datatype=”dateTime”>2004-02-23T11:23:00</tme:at>
</loc:LocationContext>

A estrutura e o vocabulário da ontologia aplicada no *Context Managing Toolkit* são descritos em RDF (Resource Description Framework).

3.4.3 Processamento de contexto

Depois que dados brutos do contexto foram obtidos de uma fonte de dados, eles tem que ser processados, pois a maioria dos seus consumidores estão mais interessados em informação agregada e interpretada do que dados brutos. Agregação de contexto significa a composição de contextos atômicos para obter toda a informação de contexto necessária para uma entidade ou para construir objetos com nível de contexto mais elevado. Por sua vez, interpretação de contexto refere-se a transformação dos dados de contexto pela inclusão de conhecimento especial. Estas formas de abstração do contexto facilitam o

projeto de aplicações.

O *Context Toolkit* oferece facilidades tanto para agregação como interpretação de contexto. Os agregadores de contexto são responsáveis por compor o contexto sobre um entidade particular através da assinatura de componentes *widgets*, interpretadores de contexto provêem a possibilidade de transformação do contexto, por exemplo, retornar o endereço de e-mail correspondente a um nome. Como *widgets*, os agregadores e os interpretadores herdam métodos de comunicação da superclasse *BaseObject* e tem que ser registrados no *Discoverer* para serem encontrados.

No SOCAM o interpretador de contexto usa uma base de conhecimento, suas tarefas incluem inferência sobre o contexto, resolução de conflitos de contexto e manutenção da consistência da base de conhecimento sobre o contexto. Diferentes regras de inferência usadas pelo interpretador podem ser especificadas. O interpretador é implementado com o uso do Jena, uma ferramenta para Web semântica.

Na arquitetura do CoBrA existe um *engine* de inferência que processa os dados do contexto. O *engine* contem um módulo de interpretação responsável pela agregação das informações do contexto. Este módulo realiza a interpretação sobre uma base de conhecimento do contexto e informações adicionais obtidas a partir de fontes externas.

No CASS a interpretação do contexto também é baseada em um *engine* de inferência e uma base de conhecimento. A base de conhecimento contém regras examinadas pelo *engine* de inferência para encontrar metas. Como estas regras são armazenadas em um banco de dados separado do interpretador, não é necessário recompilar ou reinicializar os componentes quando as regras mudam.

No CORTEX todo o processamento do contexto é encapsulado nos objetos sensíveis. A unidade de sensoriamento executa uma fusão de sensores para gerenciar incertezas dos dados dos sensores e construir objetos de contexto com nível elevado de abstração. Diferentes contextos são representados em uma hierarquia de contexto junto com ações específicas a serem empreendidas em cada contexto. Desde que somente um contexto esteja ativo em um determinado momento, o número de regras que tem que ser avaliadas são limitadas, o que aumenta a eficiência do processo de inferência. O *engine* de inferência é baseado no CLIPS (C Language Integrated Production System). Ele é responsável por alterar o comportamento da aplicação de acordo com o contexto corrente, usando regras condicionais.

3.4.4 Histórico do contexto

Algumas vezes pode ser necessário ter acesso a dados históricos sobre o contexto para estabelecer tendências e prever futuros valores do contexto. Como a maioria das fontes de dados constantemente provêem dados de contexto, a manutenção de um histórico dos contextos é principalmente uma questão de armazenamento, assim um componente de armazenamento centralizado é necessário. Como em uma arquitetura baseada em servidor os dados de contexto providos pelos sensores tem que ser armazenados no lado servidor para ser oferecido aos clientes, a maioria dos *frameworks* citados anteriormente possuem alguma funcionalidade para consultar dados históricos sobre o contexto.

Os *frameworks* Context Toolkit, CoBrA, CASS, SOCAM e CORTEX salvam os dados de contexto de forma persistente em um banco de dados. Uma vantagem adicional do uso de banco de dados é a possibilidade de utilizar SQL (Structured Query Language) para as pesquisas e manutenção dos dados. O CASS usa seu banco de dados não apenas para salvar o contexto, mas também para armazenar domínios de conhecimento e regras

de inferência necessárias para a construção de contextos de alto nível.

Devido a limitação de recursos para armazenamento, redes *peer-to-peer* de dispositivos móveis como *Hydrogen* não oferecem possibilidade de um armazenamento persistente de dados do contexto.

3.4.5 Segurança e privacidade

Como o contexto muitas vezes inclui informações sobre pessoas, por exemplo, sua localização e atividade, é necessário que se tenha a possibilidade de proteger a privacidade.

Com este propósito, o *Context Toolkit* introduz o conceito de propriedade de contexto. Usuários são associados a contextos como seus respectivos proprietários e passam a ter permissão de controlar o acesso de outros usuários. Os componentes envolvidos neste controle de acesso são: *Mediated Widgets*, *Owner Permissions*, *BaseObject* modificado e *Authenticators*. A classe *Mediated Widgets* é uma extensão da *Widget* básica que contém a especificação de quem é o proprietário dos dados de contexto que estão sendo obtidos. O *Owner Permission* é o componente que recebe permissão de consulta e determina a concessão ou negação de acesso baseado em situação armazenadas. Estas situações incluem usuários autorizados, tempo de acesso, etc. O *BaseObject* modificado contém todos os métodos originais acrescidos de mecanismos de identificação. Aplicações e componentes agora têm que prover a sua identidade junto com a habitual solicitação de informação. Por fim, o *Authenticator* é responsável por comprovar a identidade usando uma infra-estrutura de chave pública.

CoBrA inclui uma política flexível própria para controlar o acesso ao contexto. Esta política é modelada sob conceitos deônticos de direitos, proibições, obrigações e controles de acesso aos dados através de políticas dinamicamente modificáveis e dependentes de domínio.

4 ONTOLOGIAS

4.1 Fundamentos teóricos

As ontologias vem sendo utilizadas por várias áreas da Ciência da Computação, principalmente com o intuito de dotar os sistemas de meta-conhecimento. A utilização de ontologias para descrição semântica de um determinado vocabulário proporciona um entendimento amplo das características e propriedades das classes pertencentes a um domínio, assim como seus relacionamentos e restrições.

Inicialmente, é preciso estabelecer uma distinção entre o significado da palavra Ontologia (escrita com letra maiúscula) e ontologia (escrita com letra minúscula). A palavra Ontologia refere-se a uma disciplina específica da Filosofia, enquanto que a palavra ontologia possui diversas definições, sendo primariamente dividida em dois diferentes sentidos: um assumido pela Filosofia, onde ontologia se refere ao sistema particular de categorias de acordo com certa visão do mundo, não tendo uma linguagem específica para representação; e outro assumido pela Inteligência Artificial e, em geral, toda comunidade da Ciência da Computação, onde ontologia se refere a artefatos de engenharia, constituídos por um vocabulário específico usado para descrever certa realidade (GUARINO, 1998).

Na área de computação, uma das definições mais citadas na literatura é a que define ontologia como uma ”especificação explícita de uma conceituação”. Nesta definição uma ontologia representa a especificação de um vocabulário representativo dentro de um domínio compartilhado, definindo classes, relações, funções e outros objetos. Tendo o desenvolvimento de uma ontologia o objetivo de compartilhar conhecimento (GRUBER, 1993).

Um refinamento para a definição de Gruber associa ontologia ao conceito de compromissos ontológicos. Nesta definição, uma ontologia consiste de uma teoria lógica representando uma significação, a qual objetiva definição de um vocabulário formal, ou seja, seu compromisso ontológico para uma conceituação particular do mundo (GUARINO, 1998).

Uma visão diferenciada das anteriores, propõe o compartilhamento e reuso de ontologias. A proposta sugere o uso de ontologias para modelagem de problemas e domínios, onde estas iriam fornecer uma biblioteca para fácil reutilização de classes de objetos para a modelagem. O objetivo fundamental desta proposta é o desenvolvimento de uma biblioteca de ontologias, a qual poderia ser reusada e adaptada a diferentes classes de problema e ambientes (GRUNINGER, 1996).

Atualmente, a definição mais amplamente aceita e citada pelos autores da área de

computação é a que define ontologia como uma ”especificação formal e explícita de uma conceituação compartilhada” (GRUBER, 1993) (FENSEL, 2000) onde: (a) Conceituação se refere ao modelo abstrato do mundo real. (b) Explícita significa que os conceitos e seus requisitos são definidos explicitamente. (c) Formal indica que a ontologia é processável por máquina, permite raciocínio automático e possui semântica lógica formal. (d) Compartilhada significa que uma ontologia captura o conhecimento apresentado não apenas por um único indivíduo, mas por um grupo.

Uma ontologia não se resume somente a um vocabulário, mas também possui relacionamentos e restrições entre os conceitos definidos pelo vocabulário. Um tipo de relacionamento básico é o hierárquico ”é-um”. Existem diversas especificações definidas somente com relacionamentos hierárquicos que são denominadas taxonomias. Entretanto, ontologias também incluem relacionamentos não hierárquicos. Pode-se ter relacionamentos como ”tem-interesse-em”entre os conceitos pessoa e interesse, sem que se trate de um relacionamento hierárquico.

Além de definir relacionamentos, as ontologias geralmente possuem restrições, sendo então definidas como axiomas. Em uma ontologia sobre pessoas, pode-se construir uma restrição sobre o conceito pessoa baseada no relacionamento ”tem-nome”, ”uma pessoa tem exatamente um nome”. Desta maneira, construiu-se uma restrição sobre o conceito pessoa.

Quando um sistema processa uma ontologia, também é possível inferir novas informações por meio de regras de inferência. Por exemplo, uma ontologia em que parente é um relacionamento mais geral do que mãe. Se Maria é mãe de João, o sistema será capaz de concluir que Maria é parente de João. Assim, se um usuário consultar esta ontologia perguntando quem é parente de Maria, o sistema poderá responder que João é parente de Maria sem que esse fato tenha sido declarado.

Então, pode-se considerar que uma ontologia compreende um vocabulário que possui relacionamentos e restrições entre seus termos e, por meio de regras de inferência, é possível derivar novos fatos baseando-se em fatos existentes.

4.2 Motivações para o uso de ontologias

Nesta seção serão discutidas as principais motivações para o uso de ontologias, especialmente na área de Ciência da Computação.

De modo geral, pode-se afirmar que ontologias são aplicadas para possibilitar ou facilitar a comunicação entre diferentes pessoas, aplicações, sistemas, entre outros, os quais fazem parte do mesmo domínio do conhecimento, mas nem sempre compartilham de uma mesma conceituação a respeito dos componentes deste domínio. A falta de entendimento compartilhado pode provocar problemas na interoperabilidade e possibilidade de reuso e compartilhamento de conhecimento, o que é muito importante tendo-se em vista a grande variedade de métodos, paradigmas, linguagens e ferramentas existentes na área de computação.

A interoperabilidade é possibilitada pelo uso de ontologias no desenvolvimento de modelagens que expressem o conhecimento possuído pelo domínio, formando uma camada de comunicação única e comum a todos os usuários.

O reuso e o compartilhamento tornam-se possíveis porque, no momento em que se utilizam ontologias para representação do conhecimento, este se encontra padronizado e expresso em alguma linguagem formal. Desta forma, se torna mais fácil à leitura

e interpretação da ontologia por outros domínios, permitindo a sua modificação (inserindo/retirando conceitos, axiomas, relacionamentos) para se adequar a um novo domínio.

A necessidade de confiabilidade com relação aos conceitos do vocabulário ou linguagem que se está utilizando em certo ambiente é outro forte motivo para utilização de ontologias, pois a representação formal adquirida com a aplicação das mesmas pode tornar possível a automação da checagem de consistência, resultando em ambientes mais confiáveis (GRUNINGER, 1996).

Além do exposto acima, pode-se destacar algumas vantagens da utilização de ontologias na área de Ciência da Computação:

- conhecimento representado através de um vocabulário, o qual possui uma conceituação que o sustenta e evita interpretações ambíguas;
- compartilhamento e reuso da ontologia que modele adequadamente certo domínio por pessoas que desenvolvam aplicações dentro desse domínio;
- descrição exata do conhecimento fornecido por uma ontologia, em função de sua escrita em linguagem formal, a qual evita o *gap* semântico existente na linguagem natural, na qual as palavras podem ter semântica totalmente diferente conforme o seu contexto. Por exemplo, a palavra ”memória”pode estar associada a um dispositivo de armazenamento de dados em um computador, bem como pode se referir à memória humana (capacidade de natureza psicológica de adquirir, armazenar e evocar informações). A interpretação da palavra pode ser atribuída a um conceito ou outro conforme o estado mental do indivíduo. Então, no exemplo, se existir uma conceituação comum e as pessoas envolvidas concordarem em uma ontologia sobre o domínio ”computadores”, possivelmente não haverá mal entendido;
- possibilidade de fazer o mapeamento da linguagem da ontologia sem que com isso seja alterada a sua conceituação, ou seja, uma mesma conceituação pode ser expressa em várias línguas;
- possibilidade de estender o uso de uma ontologia genérica de forma a que ela torne-se adequada a um domínio específico.

4.3 Tipos de ontologia

As ontologias podem ser classificadas em diferentes tipos, os quais podem variar em função dos autores que os propõem. De modo geral, autores como (BORST, 1997) e (STUDER; BENJAMINS; FENSEL, 1998) concordam na existência de quatro tipos de ontologias:

- ontologias de domínio: capturam o conhecimento válido para um tipo particular de domínio (como mecânica, medicina, biologia, entre outros). Expressam o vocabulário relativo a um domínio particular, descrevendo situações reais deste domínio;
- ontologias genéricas: são similares às ontologias de domínio, entretanto os conceitos definidos por elas são considerados genéricos entre diversas áreas. Descrevem conceitos tipicamente gerais, como: estado; espaço; tempo; processo;

evento; importância ; ação; entre outros, os quais são independentes de um domínio ou problema particular. Conceitos em ontologias de domínio são freqüentemente definidos como especializações de conceitos de ontologias genéricas;

- ontologias de aplicação: contém todos os conceitos necessários para modelagem do conhecimento requerido por uma aplicação em particular. Esses conceitos correspondem freqüentemente aos papéis desempenhados por entidades do domínio enquanto executam certa atividade;

- ontologias de representação: não se comprometem com nenhum domínio em particular. Determinam entidades representacionais sem especificar o que deve ser representado. Ontologias de domínio e ontologias genéricas são descritas por meio das primitivas fornecidas pelas ontologias de representação.

(GUARINO, 1998) ainda descreve um outro tipo de ontologia: ”ontologias de tarefas”que descrevem tarefas ou atividades genéricas através da especialização dos termos introduzidos pelas ontologias genéricas. A figura 4.1 mostra a visão deste autor quanto à classificação das ontologias de acordo com o seu nível de generalidade, bem como representa os relacionamentos de especialização entre os tipos e o grau de reusabilidade.

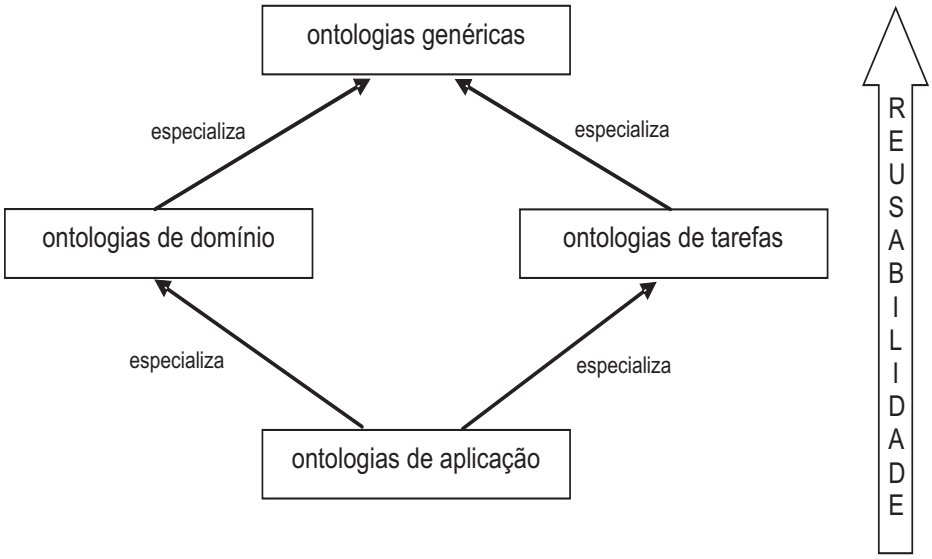

Figura 4.1: Tipos de Ontologias

4.4 Projeto de ontologias

O primeiro passo para a construção de uma ontologia é a definição clara do propósito e escopo da sua aplicação. Para tanto, é necessário realizar a captura do conhecimento, tratando de: a) identificar os principais conceitos e relacionamentos do domínio de interesse; b) definir um texto preciso a respeito destes conceitos e relacionamentos encontrados; c) definir os termos usados para se referir a estes conceitos e relacionamentos (GRUNINGER, 1996).

Após a captura do conhecimento, parte-se realmente para o projeto da ontologia. Neste momento é fundamental ter conhecimento dos principais critérios e princípios a serem seguidos para o desenvolvimento do projeto, assim como todos os componentes que deverão fazer parte da ontologia a ser criada. Depois de definidos os critérios e princípios de projeto e os componentes da ontologia, é necessário identificar uma metodologia e definir a linguagem e o ambiente que serão utilizados para construção da ontologia.

4.4.1 Princípios para construção de ontologias

Nesta seção serão apresentados alguns critérios de projeto e um conjunto de princípios para o desenvolvimento de ontologias propostos por Gómez-Pérez (GÓMEZ-PÉREZ, 1999), baseados principalmente no trabalho de Gruber (GRUBER, 1993).

- Clareza e objetividade: significa que uma ontologia deve prover o significado dos termos através de definições objetivas e também por meio de um documentação em linguagem natural.
- Complementação: significa que a definição expressa através de uma condição necessária e suficiente é preferida ao invés de uma definição parcial.
- Coerência: permite inferências que sejam consistentes com as definições.
- Extensibilidade: novos termos gerais ou especializados devem ser incluídos na ontologia de tal forma que não seja requerida uma revisão das definições existentes.
- Compromissos ontológicos mínimos: mínimo de imposições possíveis sobre o mundo que está sendo modelado, permitindo liberdade às partes comprometidas com a ontologia, para possibilitar especialização e instanciação quando necessário.
- Princípio da distinção ontológica: classes em uma ontologia devem ser disjuntas. O critério usado para isolar o núcleo das propriedades consideradas não variantes para uma instância de uma classe é chamada Critério de Identidade.
- Diversificação de hierarquias para aumentar o poder provido por mecanismos de herança múltipla. Se um suficiente conhecimento é representado na ontologia, bem como diferentes critérios de classificação sejam usados, torna-se mais fácil incluir novos conceitos e herdar propriedades de diferentes pontos de vista.
- Modularidade para minimizar o acoplamento entre os módulos.
- Minimizar a distância semântica entre conceitos próximos. Conceitos similares são agrupados e representados como subclasses de uma classe e devem ser definidos usando as mesmas primitivas.
- Padronizar nomes sempre que possível.

4.4.2 Componentes de uma ontologia

Uma ontologia provê um vocabulário comum para uma área e define - com diferentes níveis de formalidade - o significado de termos e os relacionamentos entre eles. O conhecimento nas ontologias é formalizado usando cinco tipos de componentes: classes,

relacionamentos, funções, axiomas e instâncias (GRUBER, 1993). Classes em ontologias são geralmente organizadas em taxonomias (STUDER; BENJAMINS; FENSEL, 1998). Com base nestes autores, Gómez-Pérez (GÓMEZ-PÉREZ, 1999) considera que as ontologias são composta por um conjunto de:

- conceitos e uma hierarquia entre esses conceitos, ou seja, uma taxonomia. Os conceitos podem ser abstratos ou concretos, elementares ou compostos, reais ou fictícios.
- relacionamentos entre os conceitos.
- funções, as quais são um caso especial de relacionamento em que um conjunto de elementos tem uma relação única com um outro elemento.
- axiomas usados para modelar sentenças que são sempre verdadeiras.
- instâncias que são usadas para representar elementos.

4.5 Linguagens

As linguagens utilizadas para construção de ontologias são, em geral, divididas em dois grupos: linguagens baseadas em uma lógica de primeira ordem e as que se baseiam em XML (eXtensible Markup Language) e HTML (Hiper Text Markup Language).

O primeiro grupo de linguagens é usado principalmente para representação do conhecimento, sendo que algumas delas foram desenvolvidas a partir da adaptação de linguagens para esta finalidade já existentes e outras foram desenvolvidas especificamente para construção de ontologias (CORCHO; FERNÁNDEZ-LÓPES; GÓMEZ-PÉRES, 2001). Neste trabalho serão descritas algumas destas linguagens: Loom (LOOM, 2006), Ontolingua (GRUBER, 1992), OCML (DOMINGUE; MOTTA; GARCIA, 1992) e Flogic (KIFER; LAUSEN; WU, 1995).

O segundo grupo de linguagens é usado especialmente em ambientes Web, tendo impacto principal sobre aplicações da Web Semântica, a qual consiste de uma extensão da Web já existente e conhecida, onde as informações são apresentadas a partir de significados bem definidos, possibilitando que pessoas e computadores cooperem mais facilmente entre si para o desenvolvimento de suas tarefas (BERNERS-LEE; HENDLER; LASSILA, 2001). Neste trabalho serão descritas algumas destas linguagens: RDF e RDF Schema (LASSILA; SWICK, 1999), SHOE (LUKE; JEFF, 2000), XOL (KARP; CHAUDHRI; THOMERE, 1999), OML (KENT, 1999), OIL (FENSEL; HORROCKS; VAN HARMELEN, 2000), DAML+OIL (HORROCKS; PATEL-SCHNEIDER; VAN HARMELEN, 2002) e OWL (MCGUINNESS; VAN HARMELEN, 2004). Estas linguagens, com exceção da linguagem SHOE que tem sua sintaxe baseada em HTML, possuem sintaxe baseada em XML e são adotadas como linguagens padrão para troca de informações na Web. RDF e RDF *Schema* não podem ser consideradas linguagens para construção de ontologias, sendo em geral linguagens para especificação de metadados na Web (CORCHO; FERNÁNDEZ-LÓPES; GÓMEZ-PÉRES, 2001).

4.5.1 Linguagens baseadas em lógica de primeira ordem

A linguagem **Loom** foi desenvolvida em 1991 pelo ISI (*Information Sciences Institute - University of Southern California*). Sua principal característica é o sistema de

representação do conhecimento, o qual é usado para prover suporte conclusivo para a parte declarativa da linguagem Loom, a qual é constituída por definições, regras definidas pela aplicação, regras definidas de forma padrão e fatos. Para seu funcionamento um mecanismo dedutivo, chamado classificador, utiliza proibições, unificações semânticas e tecnologia orientada a objetos com o objetivo de compilar o conhecimento declarativo inserido em uma rede projetada para sustentar de forma eficiente a dedução *on-line* de procedimentos de consulta. A linguagem Loom ainda permite a representação de conceitos, taxonomias, relações n-árias, funções, axiomas e regras de produção. O raciocínio é limitado a classificações automáticas (taxonomias podem ser criadas automaticamente para a definição de conceitos), checagem de consistência e execução de regras de produção.

A **Ontolingua** é uma linguagem para descrição de ontologias compatível com múltiplas linguagens de representação, provendo suporte para definição de classes, relações, funções, objetos e teorias. Traduz definições escritas em uma linguagem declarativa em formas que são aceitas para entrada de dados em diversos sistemas de representação do conhecimento. As ontologias que são construídas a partir do Ontolingua podem ser compartilhadas por múltiplos usuários e grupos de pesquisa, cada um usando seu próprio sistema de representação. Esta linguagem foi desenvolvida em 1992 pelo KSL (*Knowledge Systems Laboratory - Stanford University*) e sua sintaxe e semântica foram desenvolvidas com base na linguagem KIF (*Knowledge Interchange Format*). KIF é uma linguagem formal para troca de conhecimento entre sistemas computacionais muito diferentes. Suas principais características são: a) semânticas declarativas, isto é, o significado das expressões existentes na representação pode ser compreendido sem recorrer a um interpretador para manipular as expressões; b) é compreensível logicamente, isto é, fornece suporte para expressão de sentenças arbitrárias em cálculo de predicados de primeira ordem; c) suporte para a representação de regras de raciocínio não-monotônicas; d) fornece suporte para a definição de objetos, funções e relações. A linguagem KIF contém uma base de conhecimento clara para o leitor, mas não possibilita o raciocínio automático por meio dela. Desta forma, a Ontolingua traduz definições escritas em KIF para uma forma apropriada, visando desenvolver sistemas de representação do conhecimento.

A **OCML** (*Operational Conceptual Modelling Language*) é uma linguagem de modelagem desenvolvida em 1993 pelo KMI (*Knowledge Media Institute - Open University*). A OCML é bastante similar à Ontolingua, porém apresenta componentes adicionais a esta, que são: regras dedutivas e de produção e definições operacionais para funções. Permite especificação e operacionalização de funções, relações, classes, instâncias e regras, além de apresentar uma poderosa checagem de restrições, que podem checar restrições de tipo e de cardinalidade, associadas às relações, *slots* e classes.

A linguagem **FLogic** (*Frame Logic*) é um formalismo que descreve, de maneira limpa e declarativa, os mais diversos aspectos estruturais de linguagens baseadas em *frames* e orientadas a objetos, sendo, além disso, adequado para a definição, execução de consultas e manipulação de esquemas de bancos de dados. Desenvolvido em 1995 na Universidade de *Karlsruhe* - Alemanha, integra *frames* em primeira ordem e cálculo de predicados, sustentando o mesmo relacionamento para o paradigma de orientação a objetos que o cálculo de predicados mantém para a programação relacional. Suas principais características incluem: identidade de objetos, objetos complexos, herança, polimorfismo, métodos para consulta, encapsulamento, entre outros. Através desta linguagem é possível a representação de conceitos, taxonomias, relações binárias, funções, axiomas, instâncias e regras dedutivas.

4.5.2 Linguagens para aplicações da *web* semântica

RDF e RDF *Schema* (*Resource Description Framework*) são linguagens que têm como objetivo prover interoperabilidade entre aplicações que trocam informações interpretáveis por máquina na *Web*. Essas linguagens buscam facilitar o processamento autônomo de recursos da *Web*, tornando possível a especificação de semânticas de dados baseadas em XML, de uma forma padronizada e interoperável. Uma das áreas em que RDF é bastante aplicada é no descobrimento de recursos, onde é capaz de prover maior capacidade aos mecanismos de busca. O principal objetivo da linguagem RDF é definir um mecanismo para descrição de recursos que não faça suposições a respeito de um domínio de aplicação em particular e que não defina as semânticas de nenhum domínio de aplicação. Neste caso, a definição do domínio deve ocorrer de forma imparcial, mesmo que o mecanismo deva ser apropriado para a descrição de informações a respeito de qualquer domínio. A linguagem RDF é orientada a objetos, nela uma coleção de classes é chamada de *schema*, e estas classes são organizadas em uma hierarquia, provendo extensibilidade através das subclasses definidas. A RDF *Schema* provê informações a respeito da interpretação dos enunciados apresentados em um modelo de dados RDF, o que difere do propósito das DTDs (*Document Type Definition*) pertencentes à linguagem XML, as quais provêem restrições específicas a estrutura de um documento XML.

SHOE (*Simple HTML Ontology Extensions*) é uma extensão da linguagem HTML, que adiciona uma maneira de incorporar conhecimento semântico interpretável por máquina em documentos HTML e, também, pode ser usado em documentos XML. A linguagem SHOE foi desenvolvida na Universidade de Maryland, em 1996, ela permite a representação de conceitos e suas taxonomias, relações n-árias, instâncias e regras de dedução, os quais são usados pelo seu mecanismo de inferência para obter novo conhecimento.

XOL (*Ontology Exchange Language*) é uma linguagem para ontologias baseada em XML, desenvolvida inicialmente para uso somente pela comunidade de bioinformática. Apesar de possibilitar atualmente utilização por qualquer domínio, é uma linguagem bastante restrita, a qual permite especificação somente de conceitos, taxonomias e relações binárias. Essa linguagem foi desenvolvida no SRI *International* (*Stanford Research Institute*) em 1999.

OML (*Ontology Markup Language*) é uma linguagem para especificação de ontologias desenvolvida na Universidade de Washington em 1999. Esta linguagem apresenta uma estrutura ontológica e semântica, onde a estrutura ontológica é composta por classes, relacionamentos, objetos e restrições. Possui como base a descrição lógica e conceitual da estrutura da ontologia na forma de grafos, permitindo a representação de conceitos, definidos a partir de suas taxonomias, axiomas e relações em lógica de primeira ordem.

OIL (*Ontology Inference Layer*) é uma proposta para representação e inferência de ontologias voltadas a Web, combinando as primitivas de modelagem usadas para linguagens baseadas em *frames* com as semânticas formais e os serviços de raciocínio providos pelas lógicas de descrição. OIL foi a primeira linguagem para representação de ontologias fundamentada corretamente nos padrões estabelecidos pela W3C (*World Wide Web Consortium*), como são as linguagens RDF/RDF *Schema*. Ela unifica três aspectos importantes vindos de diferentes entidades: a) semânticas formais e suporte eficiente ao raciocínio, providos pelas lógicas de descrição; b) primitivas epistemológicas de modelagem ricas providas pelos sistemas baseados em *frames*; e c) uma proposta padronizada para troca de notações sintáticas, como provido pela *Web*.

DAML+OIL (*DARPA Markup Language + Ontology Inference Layer*) é uma linguagem semântica voltada a aplicações ligadas a recursos da *Web*. Resultado da união das linguagens DAML e OIL, ela foi desenvolvida por um grupo de pesquisadores europeus e norte-americanos. Possui conformidade com os padrões de linguagem estabelecidos pela W3C, assim como RDF e RDF *Schema*. Em sua linguagem para construção de ontologias, DAML+OIL pretende descrever a estrutura de um domínio, apresentando uma modelagem orientada e objetos, com o domínio sendo descrito em termos de classes e propriedades. Além disso, esta linguagem permite a representação de conceitos, taxonomias, relações binárias, funções e instâncias.

OWL (*Web Ontology Language*) é uma linguagem para especificação de ontologias que foi recomendada como um padrão de linguagem pela W3C. Esta linguagem apresenta todos os benefícios das outras linguagens destinadas a especificação de ontologias, tais como: DAML+OIL, RDF e RDF Schema. A linguagem OWL revisa e incorpora alguns melhoramentos à linguagem DAML+OIL. Um dos melhoramentos é a criação de um vocabulário mais extenso para descrição de propriedades e classes, permitindo descrição de relacionamentos entre classes, cardinalidade, igualdade, características de propriedades, entre outras. A OWL é dividida em três sub-linguagens, distintas pelo nível de formalidade exigido e oferecido e a liberdade dada ao usuário para a definição de ontologias: OWL- *Lite*, OWL-DL e OWL- *Full*. A OWL- *Lite* tem como finalidade principal dar suporte para que classificações hierárquicas com restrições simples sejam criadas. Por exemplo, restrições de cardinalidade são permitidas apenas para valores iguais a zero ou um. Assim, OWL- *Lite* é própria para a construção de taxonomias e tesauros. A OWL-DL provê um maior grau de expressividade onde todas as conclusões são computáveis e todas as computações terminam em tempo finito. OWL-DL corresponde à lógica de descrição. A OWL- *Full* tem como objetivo prover o máximo de expressividade e liberdade sintática. Com a OWL- *Full*, os usuários podem aumentar o vocabulário pré-definido de RDF ou OWL.

4.6 Ferramentas de desenvolvimento

Nesta seção serão apresentadas algumas ferramentas voltadas ao desenvolvimento de ontologias.

Apollo: é uma aplicação para a modelagem de ontologias desenvolvida através de primitivas básicas, como classes, funções, relações, instâncias, entre outras. A sua modelagem interna é construída de acordo com um sistema de frames, além disso, a ferramenta Apollo fornece checagem de consistência na medida em que a ontolo gia é desenvolvida. A ferramenta não obriga utilização de uma linguagem específica para a criação de ontologias e pode ser adaptada a diversos formatos de armazenamento, por meio da utilização de plug-ins. Essa ferramenta é desenvolvida em linguagem Java e possui uma arquitetura aberta (APOLLO, 2003). Aceita as linguagens OCML, Ontolingua, entre outras.

OILEd: é uma ferramenta para edição de ontologias baseadas em linguagem OIL e DAML+OIL. Em sua funcionalidade básica, a ferramenta permite a definição e descrição de classes, slots, entidades e axiomas, onde classes são definidas em termos de suas superclasses e restrições de propriedade. Uma inovação apresentada pelo OILEd é a utilização de raciocínio para checagem de consistência entre as classes e para inferência de classificação entre relacionamentos. Este serviço de raciocínio é provido pelo sistema FaCT, o qual consiste de um classificador de ontologias via tradução de DAML+OIL para

lógica de descrição (BECHHOFER et al., 2001). A ferramenta é desenvolvida em linguagem Java e é disponível para uso através de licença GPL (General Public License). Essa ferramenta trabalha com as linguagens: RDF, RDF Schema, OIL e DAML+OIL.

Ontolingua: servidor criado pelo KSL (*Knowledge Systems Laboratory - Stanford University*), sendo constituído por um conjunto de ferramentas e serviços que suportam a construção de ontologias compartilháveis entre grupos geograficamente distribuídos. A arquitetura do servidor de ontologias fornece acesso a bibliotecas de ontologias, tradutores de linguagens e um editor para criação. Editores remotos podem visualizar e editar ontologias, aplicações remotas ou locais podem acessar qualquer ontologia das bibliotecas, através do protocolo OKBC (Open Knowledge Based Connectivity) (CORCHO; FERNÁNDEZ-LÓPES; GÓMEZ-PÉRES, 2001). O Servidor Ontolingua aceita as linguagens: Loom, Ontolingua, entre outras.

OntoSaurus: é um servidor *Web* para ontologias criadas a partir da linguagem Loom, desenvolvido pelo ISI (*Information Sciences Institute - University of Southern California*). É constituído por um servidor de navegação de ontologias, o qual cria dinamicamente páginas em formato HTML para exibição da hierarquia de uma ontologia.

Protégé: é um ambiente extensível e independente de plataforma escrito em Java. É uma das ferramentas mais utilizadas para criação e edição de ontologias e bases de conhecimento, suportando a criação, visualização e manipulação de ontologias em vários formatos. O Protégé possui uma vasta quantidade de *plugins*, importa e exporta ontologias em diversos formatos, facilitando a reutilização e o intercâmbio de ontologias, além de incorporar diversas outras funcionalidades, como visualizadores e integradores e possibilitar o uso de *plugins* desenvolvidos por usuários. O Protégé foi desenvolvido na Universidade de Stanford e é disponível para utilização gratuitamente.

WebOnto: é uma ferramenta para criação de ontologias desenvolvida pelo KMI (*Knowledge Media Institute - Open University*). Ela suporta navegação colaborativa, criação e edição de ontologias, as quais são representadas na linguagem OCML. Suas principais características são: (a) gerenciamento de ontologias utilizando uma interface gráfica; (b) suporte para modelagem de tarefas; (c) verificação de elementos, considerando herança de propriedades e checagem de consistência; (d) suporte ao trabalho colaborativo. WebOnto é um servidor disponível gratuitamente, através dele é possível o acesso a mais de 100 ontologias, as quais podem ser visualizadas sem restrições de acesso (CORCHO; FERNÁNDEZ-LÓPES; GÓMEZ-PÉRES, 2001).

4.7 Lógica de Descrições

As pesquisas no campo da representação do conhecimento e do raciocínio automático são normalmente voltadas à definição de linguagens formais para representar o conhecimento e métodos de inferências associados a essas linguagens. Desta forma, é possível construir sistemas que tenham a capacidade de encontrar informações implícitas através da análise de um conhecimento representado explicitamente. Tais sistemas envolvem dois aspectos básicos: (a) caracterização precisa da base de conhecimento, ou seja, é necessário definir claramente o tipo de conhecimento a ser especificado e os serviços de raciocínio disponíveis. (b) disponibilização de um *framework* para facilitar tanto o processo de representação como o de análise do conhecimento.

As lógicas de descrições são uma evolução dos formalismos de representação do conhecimento baseados em objeto, tais como: redes semânticas e *frames*, correspondendo

a um subconjunto estruturado da lógica de primeira ordem. De modo geral, as lógicas de descrições são formalismos para representar conhecimento e raciocinar sobre ele. Isso significa que a sintaxe deste formalismo foi definida para facilitar o raciocínio, gerando um menor custo computacional (BAADER, 2002).

As lógicas de descrições são voltadas para a especificação e prova de propriedades onde as noções de conceito e relacionamento entre estas são centrais, sendo particularmente útil na especificação de ontologias (GRäDEL, 1998). Neste sentido, observa-se que a OWL é uma linguagem baseada na lógica de descrição, possuindo um suporte para inferência fundamentado nessa lógica, o qual é utilizado por APIs Java, tais como o *framework* Jena.

A grande capacidade de representação é uma das características principais das lógicas de descrições, além disso existem métodos de dedução eficientes para os serviços de raciocínio. Entre estes métodos encontra-se o Algoritmo Tableau, que é usado pela RACER (Renamed Abox and Concept Expression Reasoner), uma ferramenta de raciocínio automática para OWL. Este algoritmo é um método refutacional de prova de teoremas, ou seja, ao invés de provar um teorema diretamente, ele prova que sua negação é falsa. Isso é feito através da aplicação de regras específicas para a lógica de descrições que dependem dos construtores presentes na lógica em questão.

A sintaxe das lógicas de descrições é formada por símbolos representando conceitos e papéis, construtores e quantificadores. Os conceitos são representações de classes, conjuntos de indivíduos que representam as mesmas características gerais. Podem ser conceitos base ou primitivos, que não dependem de outros conceitos ou relacionamentos para serem definidos, ou conceitos complexos, que são formados a partir da utilização de outros conceitos já declarados. Os construtores são operadores que permitem a criação de conceitos complexos, dando um significado especial à interpretação do conceito. Os papéis são propriedades dos conceitos. Eles representam relacionamentos entre os elementos da base de conhecimento (conceitos e instâncias). Os quantificadores são operadores que quantificam os papéis.

Uma interpretação de um conceito A, representada por Ai, pode ser definida como o conjunto de valores que torna este conceito verdadeiro. O conceito mais geral do qual todos os outros conceitos são subconceitos é representado por Di e é equivalente ao Verdadeiro. O Verdadeiro e o Falso são representados, respectivamente, pelos símbolos $\top$ e $\bot$.

Considerando que A e B sejam nomes de conceitos, P um nome de papel (propriedade) pode-se ter os seguintes construtores básicos:

- conjunção (A $\sqcap$ B). Representa a interseção entre as interpretações dos conceitos que fazem parte da conjunção;
- disjunção A $\sqcup$ B (U). Representa a união entre as interpretações dos conceitos que fazem parte da disjunção;
- quantificação universal $\forall$P.A. Determina que, através da propriedade P, todos os indivíduos do conceito declarado devem se relacionar com indivíduos da classe determinada pela quantificação universal (conceito A). Desta forma, a propriedade P não pode ser usada para relacionar elementos de outras classes;
- quantificação existencial $\exists$P.A (E). Determina que, através da propriedade P, deve existir pelo menos um indivíduo do conceito declarado que se relaciona com in-

divíduos da classe determinada pela quantificação existencial (conceito A). Os indivíduos do conceito declarado podem se relacionar com outras classes através desta mesma propriedade P;

- negação $\neg$A (C). Representa a interpretação de D*i* menos a de A*i*;
- restrição de número $\leq$n.P e $\geq$n.P (N). Determinam o número máximo ou mínimo de relacionamentos que devem ser realizados através da propriedade P.

Existem vários tipos de lógicas de descrições, cada uma caracterizada pelos construtores que possuem e quais propriedades podem ser atribuídas aos papéis, por exemplo, simetria, inversão, transitividade. A lógica **AL** (*Attributive Language*) é a que proporciona menor expressividade, sendo composta por quantificação universal, conjunção, Verdadeiro, Falso, quantificação existencial e negação de conceitos atômicos. Ao adicionar outros construtores (E,U,C,N) podem ser criadas lógicas com mais expressividade que estas duas, tais como: ALCN, ALUE. As letras que compõem os nomes das lógicas são compostas pelos símbolos dos construtores e das propriedades que ela possui.

4.7.1 Representação do conhecimento

Uma base de conhecimento contém as informações de um determinado domínio, ou seja, ela é a representação de um conhecimento específico. Para representar de forma mais adequada este conhecimento, é necessário dividir a base em duas partes:

- conhecimento *intensional*: conhecimento geral sobre o domínio do problema. Representa o conhecimento sobre grupos (conjuntos) de indivíduos que apresentam as mesmas características;
- conhecimento *extensional*: especifica um problema particular. Representa o conhecimento sobre cada indivíduo que faz parte de um conjunto.

Na lógica de descrições, o conhecimento *intensional* é chamado de TBox e o *extensional* de ABox.

O TBox contém o conhecimento *intensional* na forma de terminologia. Ele representa as características gerais dos conceitos, que são grupos de indivíduos semelhantes. A forma básica de declaração em um TBox é a definição de conceito, a qual corresponde a definição de um novo conceito em termos de outros conceitos definidos previamente.

As declarações do TBox são representadas como equivalências lógicas (condições necessárias e suficientes, denotado por $\equiv$) ou como uma inclusão (condições necessárias, denotado por $\subseteq$). Estas declarações possuem as seguintes características:

- somente é permitida uma definição para cada nome de conceito;
- é recomendado que as definições sejam acíclicas, ou seja, elas não podem ser definidas em termos delas mesmas e nem de outros conceitos que indiretamente se referem a elas.

Um exemplo de TBox é mostrado na figura 4.2. Os conceitos como Pessoa e Fêmea, no exemplo, definidos apenas pelo próprio nome, são chamdos de conceitos base. O conceito Mulher é declarado como conceito resultante da interseção entre as

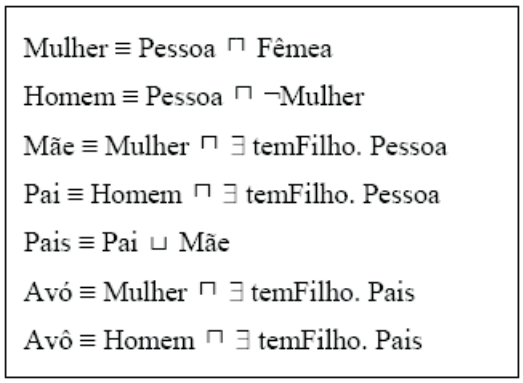

Figura 4.2: Exemplo de TBox

interpretações dos conceitos Pessoa e Fêmea, ou seja, o conceito Mulher é formado pelos indivíduos que pertencem aos conceitos Pessoa e Fêmea simultaneamente. Este conceito é dito complexo, pois precisa de outros para ter um significado.

Para facilitar o desenvolvimento de procedimentos de raciocínio, pode-se reduzir os problemas de raciocínio com relação a um TBox acíclico T para problemas com respeito ao TBox vazio. Um TBox vazio é um TBox no qual todos os conceitos complexos são definidos apenas com a utilização de conceitos base. Isto permite que os algoritmos consigam encontrar mais facilmente as semelhanças entre conceitos, contradições, etc. Para se obter este TBox é necessário expandir as definições de conceitos armazenados no TBox até que os conceitos complexos sejam definidos apenas por conceitos primitivos. Isso significa, conceitos onde cada definição que esteja na forma A $\equiv$ D, D contém somente conceitos primitivos (base). Para cada conceito C, definimos uma expansão de C com respeito a T como o conceito C' que é obtido de C pela substituição de cada ocorrência de um nome de símbolo A em C pelo conceito D.

A figura 4.3 mostra o TBox vazio correspondente à figura 4.2. Neste exemplo, todos os conceitos complexos foram substituídos pelos conceitos base que lhes dão origem, ou seja, todos eles estão definidos em função dos conceitos Pessoa e Fêmea.

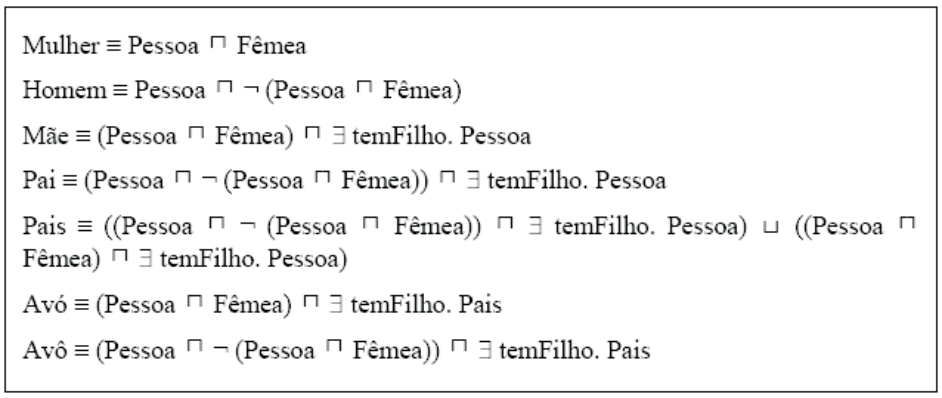

Figura 4.3: Exemplo de Expansão de um TBox

O ABox contém o conhecimento *extensional*, especifica os indivíduos do domínio. Ele é a instanciação da estrutura de conceitos. Existem dois tipos de declarações no ABox:

- declaração de conceitos: C(a). Declara que "a"é um indivíduo do conceito "C". Por

exemplo, Pessoa(Ana);

- declaração de papel: R(a,b). Declara que o indivíduo ”a”está relacionado com o indivíduo ”b”através da propriedade ”R”. Por exemplo: temFilho(Mauro,Ana).

A figura 4.4 mostra instâncias dos conceitos Mulher e Homem, bem como relacionamentos entre indivíduos da duas classes, utilizando a propriedade temFilho para expressar o grau de parentesco entre eles.

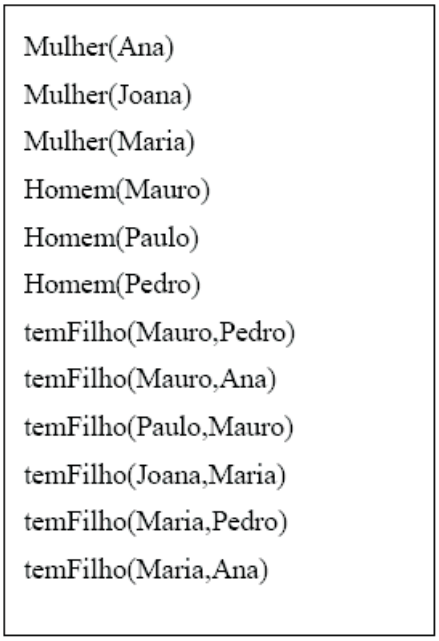

Figura 4.4: Exemplo de ABox

4.7.2 Raciocínio

Uma das grandes vantagens da utilização de lógicas de descrições como método de representação do conhecimento é a possibilidade de se utilizar sistemas de raciocínio. Os sistemas de raciocínio tem como objetivo processar conhecimento representado explicitamente e encontrar informações implícitas nestas informações, através da utilização de serviços específicos. Um destes sistemas é o RACER, que pode ser utilizado para obtenção de resultados de inferências através de uma interface gráfica, o RICE (RACER *Interactive Client Environment*), bem como pode ser utilizado em conjunto com outras ferramentas, como o Pretégé, para facilitar o processo de edição de ontologias (HAARSLEV; MOLLER, 2001).

O RACER implementa um cálculo de tableaux otimizado para uma lógica de descrições muito expressiva, a ALCQHIR+ também conhecida como SHIQ. Ele também oferece serviços de raciocínio para múltiplos TBoxes, bem como para múltiplos ABoxes. A lógica ALCQHIR+ é composta de conceito atômico, conceito universal (verdadeiro), conceito base (falso), negação atômica, interseção de conceitos, quantificação universal, quantificação existencial limitada, negação, restrição de números qualificada, regras hierárquicas, regras de inversão e regras de transitividade.

Os serviços do RACER disponíveis para o TBox são:

a) Satisfatibilidade

Um conceito é *satisfatível* em relação a um TBox T se existe uma interpretação I de T tal que C^I é não vazio. Neste caso dizemos que I é um modelo de T. Para que um conceito *satisfatível* deve ser possível criar pelo menos uma instância dele, ou seja, o conceito não pode ter uma contradição na sua definição.

Na figura 4.5, um teste de satisfatibilidade é exemplificado para o conceito Mãe. Para isto são criadas instâncias genéricas para este conceito (Mãe(a)) e para os relacionamentos desta instância (temFilho(a,b)). A partir disso, observa-se durante a expansão do TBox se ocorrerá alguma contradição, se nenhuma ocorrer é verificado que o conceito Mãe é *satisfatível*.

TBox

Mulher ≡ Pessoa ⊓ Fêmea

Mãe ≡ Mulher ⊓ ∃ temFilho. Pessoa

Teste de Satisfatibilidade

Mãe(a) ≡ Mulher(a) ⊓ temFilho(a,b)

Mãe(a) ≡ (Pessoa(a) ⊓ Fêmea(a)) ⊓ temFilho(a,b)

Figura 4.5: Teste de Satisfatibilidade

b) Subclassificação

Um conceito C é subclassificado por um conceito D com respeito a TBox T se $C^I \subseteq D^I$ para toda interpretação I de T. Isso significa que todas as instâncias de C também são instâncias de D. O conceito D é chamado classificador e o conceito C classificado.

Na figura 4.6 observa-se que Mãe é um tipo especial de mulher, pois este conceito é composto pelas instâncias do conceito Mulher que tem um relacionamento através da propriedade temFilho com a classe Pessoa. Como todos os indivíduos do conceito Mãe fazem parte do conceito Mulher pode-se concluir que Mãe $\subseteq$ Mulher.

TBox

Mulher ≡ Pessoa ⊓ Fêmea

Mãe ≡ Mulher ⊓ ∃ temFilho. Pessoa

Relacionamento entre os conceitos

Mãe ⊆ Mulher

Classificador : Mulher

Classificado : Mãe

Figura 4.6: Teste de Subclassificação

c) Equivalência

Dois conceitos C e D são equivalentes com respeito a um Tbox T se CI = DI para toda interpretação I de T. Isso significa que ambos conceitos têm as mesmas instâncias.

Na figura 4.7 observa-se que após fazer a expansão e simplificação do conceito Homem, verifica-se que ele e o conceito Masculino são definidos da mesma maneira, logo são conceitos equivalentes.

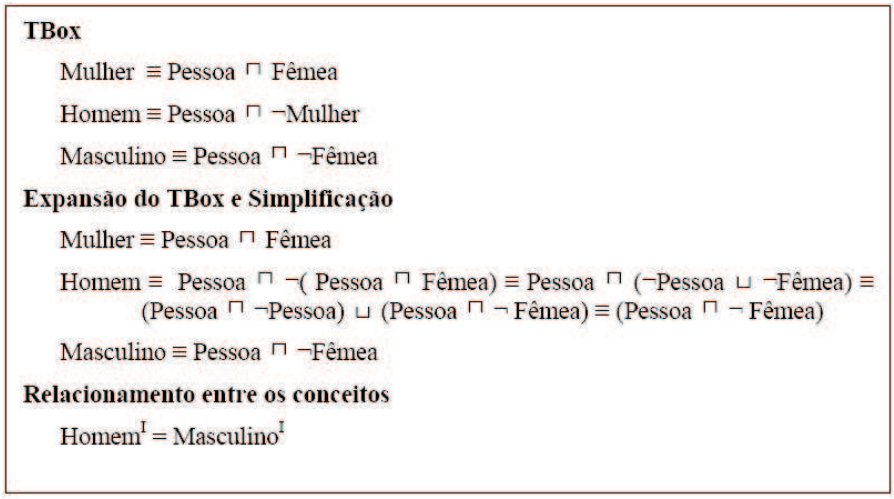

Figura 4.7: Teste de Equivalência

d) Disjunção

Dois conceitos C e D são disjuntos com respeito a um TBox T se CI ∩ DI = ⊘ para toda interpretação I de T. Isso significa que dois conceitos disjuntos não podem compartilhar a mesma instância.

Na figura 4.8 após fazer a expansão e simplificação do conceito Homen, pode-se observar que ele e o conceito Mulher possuem conceitos contraditórios em suas definições. Como não é possível uma instância pertencer a um conceito e a seu complemento simultaneamente (com exceção do vazio), os conceitos Homem e Mulher não podem ter instâncias comuns a ambos.

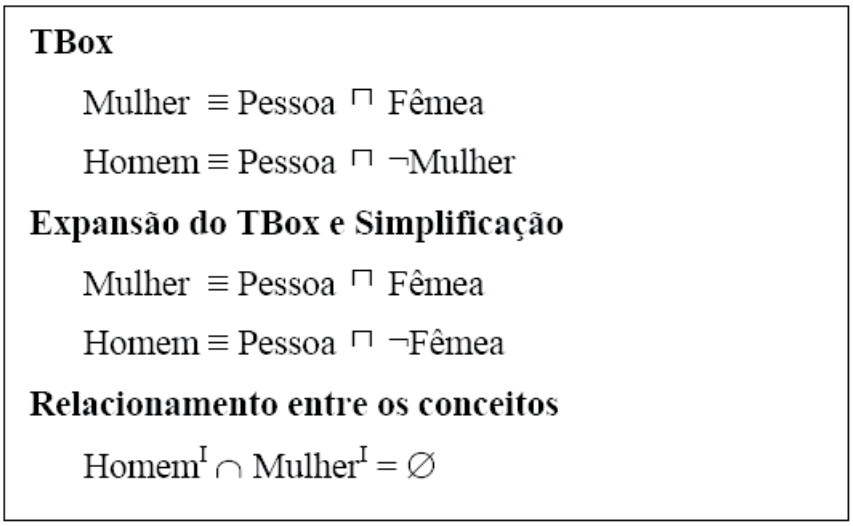

Figura 4.8: Teste de Disjunção

Os serviços do RACER disponíveis para o ABox são:

a) *Checagem* de Consistência

Um Abox A é consistente em relação a um TBox T se existe uma interpretação que é modelo de ambos, A e T. A é consistente se ele é consistente ao TBox vazio. Isso significa que se existir ums instância que torne tanto o TBox quanto o Abox verdadeiros, o ABox será consistente.

Na figura 4.9, foi criada uma instância do conceito Mulher denominada Maria. Como Mulher é a interseção das interpretações dos conceitos Pessoa e Fêmea, Maria também fará parte destes conceitos. Como o conceito Mulher é *satisfatível*, o ABox será consistente.

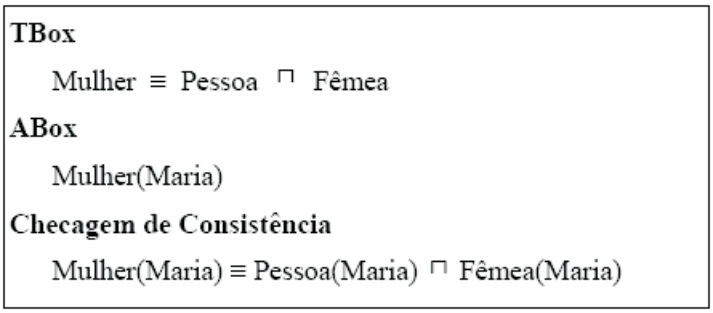

Figura 4.9: *Checagem* de Consistência

b) *Checagem* de Instância

Verifica se um dado indivíduo é uma instância de um conceito específico. Na figura 4.10, deseja-se saber se o indivíduo Maria faz parte do conceito Mãe. Como ela é instância do conceito Mulher e possui um relacionamento através da propriedade temFilho com o indivíduo Pedro, pode-se concluir que Maria atende as condições necessárias e suficientes para pertencer à classe Mãe.

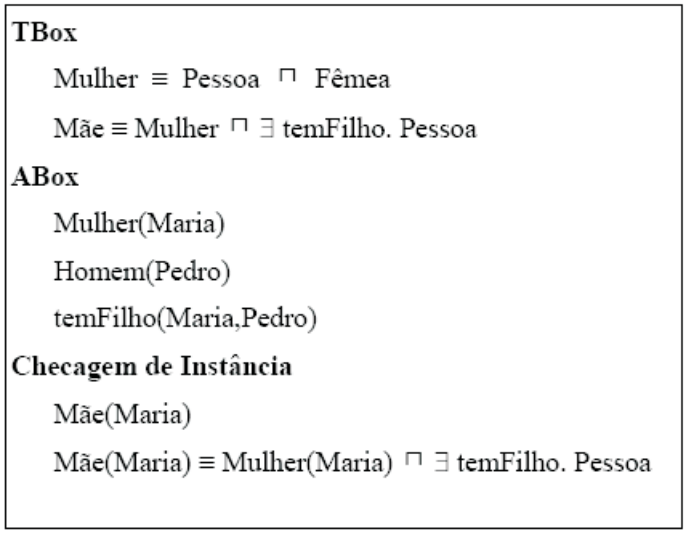

Figura 4.10: *Checagem* de Instância

c) Retorno

Encontra o conceito mais específico do qual um dado indivíduo é uma instância. Na figura 4.11, ao aplicar este serviço nos indivíduos Maria, João e Pedro, obtêm-se como

resultado os conceitos mais baixos na hierarquia de conceitos aos quais estas instâncias pertencem.

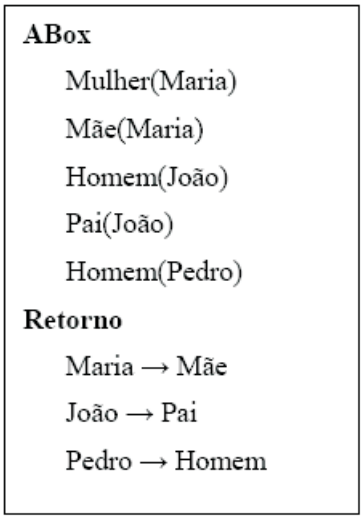

Figura 4.11: Serviço de Retorno do RACER

d) Realização

Encontra os indivíduos na base de conhecimento que são instâncias de um dado conceito. Na figura 4.12, ao aplicar este serviço no conceito Homem e no conceito Mulher, obtêm-se como resultado os indivíduos que fazem parte destes conceitos.

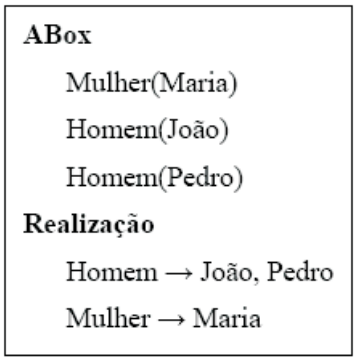

Figura 4.12: Serviço de Realização do RACER

O sistema de raciocínio RACER pode ser utilizado em conjunto com o Protégé, fazendo inferências sobre o conhecimento expresso em ontologias no formato OWL, dentre outros. O Protégé é um dos mais utilizados editores para ontologias. Ele provê uma interface para criação de ontologias que torna o processo transparente ao usuário, no que diz respeito à necessidade de conhecimentos mais aprofundados em relação a linguagens. Entretanto, é necessário um razoável conhecimento sobre lógica de descrições, visto que todo o processo de criação das ontologias é baseado neste formalismo. O Protégé utiliza o RACER para obter resultados para os serviços de hierarquia inferida (subclassificação), consistência de conceitos (satisfatibilidade) e equivalência de conceitos.

Para explorar toda a potencialidade do RACER com relação aos serviços de raciocínio, deve ser utilizada a ferramenta RICE. Esta ferramenta utiliza o RACER para

fazer inferências no TBox e no ABox, além de responder questões sobre uma ontologia (CORNET, 2004).

5 COMPUTAÇÃO PERVASIVA, SENSÍVEL AO CONTEXTO E ONTOLOGIAS

O foco deste trabalho é a avaliação do uso de ontologias na sensibilidade ao contexto no âmbito da computação pervasiva. Suas premissas fundamentais são: ambiente pervasivo com composição dinâmica, aplicações distribuídas, móveis e conscientes de contexto e construção de ontologias para representação do contexto. Assim, neste capítulo será explorada a correlação entre computação pervasiva, sensível ao contexto e ontologias, avaliando o emprego de ontologias na qualificação dos mecanismos utilizados para expressar informações de contextos.

5.1 Trabalhos relacionados

Uma das idéias centrais da computação pervasiva é a transparência na resolução das tarefas computacionais dos usuários, possibilitando acesso independente de localização, tempo e equipamento. Neste sentido, existem alguns projetos que utilizam ontologias para a representação de contextos que refletem a situação do usuário no mundo real. Essas ontologias podem ser construídas usando linguagem OWL e nelas as instâncias dos conceitos podem referenciar uma pessoa, um dispositivo ou um local.

Um projeto que se destaca nessa área é o SOUPA (*Standard Ontology for Ubiquitous and Pervasive Applications*) (CHEN et al., 2004) que utiliza OWL na criação de sua ontologia. O objetivo do projeto é definir uma ontologia, conforme mostra a figura 5.1, para suportar aplicações destinadas a ambientes pervasivos. O vocabulário do SOUPA coincide com o vocabulário de algumas ontologias existentes e seu mérito está em providenciar aos desenvolvedores de aplicações uma ontologia que combina muitos vocabulários úteis de diferentes ontologias consensuais.

5.2 Ontologias e computação pervasiva

Os ambientes pervasivos são repletos de dispositivos computacionais e de telecomunicações, plenamente integrados com os usuários. Estes ambientes envolvem a construção de sistemas para computação distribuída, caracterizados por um grande número de entidades autônomas. Estes agentes podem ser dispositivos, aplicações, serviços, bases de dados, usuários. Diversos tipos de *middlewares* já foram desenvolvidos para possibilitar a comunicação entre as diferentes entidades. Entretanto, os *middlewares* existentes não possuem dispositivos para facilitar a interoperabilidade semântica entre

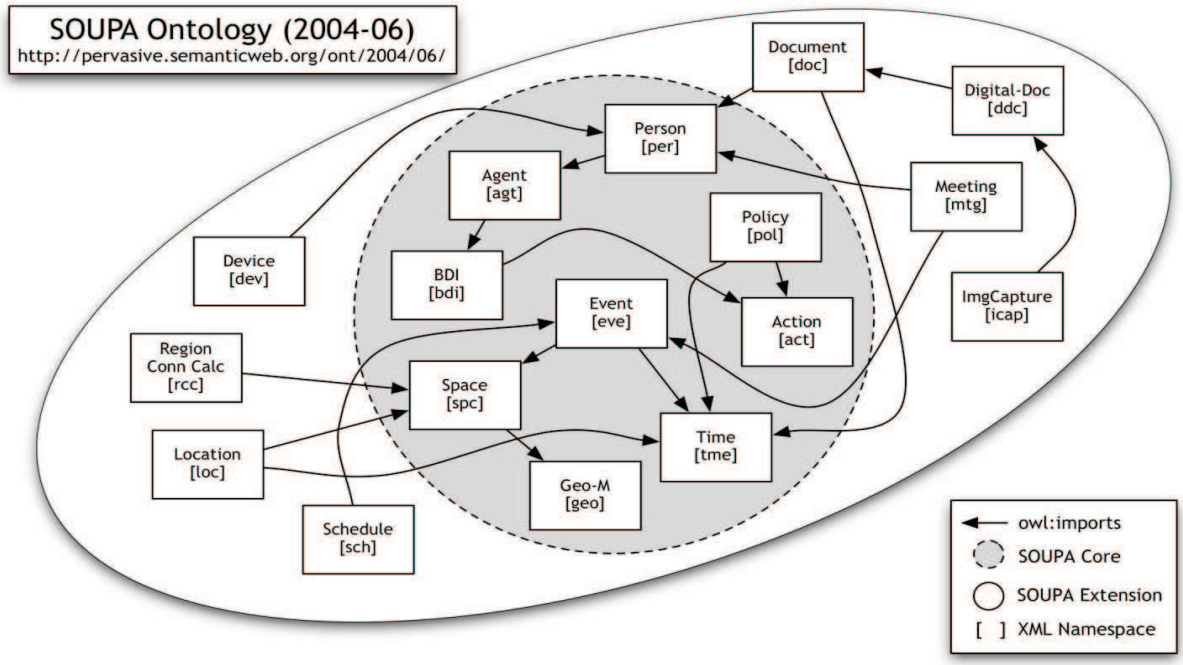

Figura 5.1: Ontologia SOUPA

os diversos tipos de entidades. Considerando estes aspectos, a seguir são descritas três características importantes da computação pervasiva nas quais o uso de ontologias pode produzir avanços significativos (CHEN; FININ; JOSHI, 2004).

a) *Discovery* e *Matchmaking*

Um ambiente de computação pervasiva deve possuir um ou mais registros para que seja mantido um estado de tempo real, por exemplo: as entidades que estão disponíveis e presentes no momento no ambiente, um protocolo de descoberta para o controle da chegada e partida das entidades móveis comunicando sua disponibilidade e notificando as partes envolvidas sobre as mudanças. Assim é caracterizado o termo ”Serviço de Descoberta”(*Discovery Service*). Na função de *Discovery*, esquemas padronizados são necessários para descrever muitos tipos de entidades, incluindo pessoas, lugares e coisas. Além disso, o sistema possui políticas, restrições e relacionamentos, os quais eventualmente necessitarão ser *descobertos*. Para que o sistema seja robusto é necessário ter um mecanismo flexível que proporcione o intercâmbio descritivo de informações de diversos tipos, com uso de ontologias apoiadas pelas ferramentas de desenvolvimento, tais como: a linguagem OWL e o editor Protégé.

b) Interoperabilidade entre as diversas entidades

Novas entidades podem entrar no ambiente a qualquer hora e estas novas entidades devem interagir com as entidades existentes. A interação precisa ser baseada em conceitos comuns, muito bem definidos, não devendo haver desentendimentos entre as entidades. As entidades devem possuir um entendimento comum de vários termos e conceitos utilizados nas interações. Para que entidades autônomas interajam umas com as outras, elas precisam conhecer, antecipadamente, os tipos de interfaces suportadas e quais protocolos e comandos são entendidos. Em um cenário distribuído, como um ambiente de computação pervasiva, assume-se que tais acordos devem existir. Mecanismos similares são necessários para que as pessoas interajam com as diferentes entidades. As pessoas precisam entender o que as diversas entidades fazem e precisam compreender também os

relacionamentos entre elas. Torna-se necessário, então, formalizar um modelo conceitual do ambiente, para que esta interação ocorra facilmente. Neste caso, a formalização destes mecanismos pode ser obtida através do uso de ontologias.

c) Sensibilidade ao contexto

As aplicações em um ambiente móvel e pervasivo necessitam ser cientes do contexto, assim elas podem adaptar-se rapidamente às mudanças de situações. Aplicações em um ambiente pervasivo utilizam diferentes tipos de contexto como por exemplo: localização das pessoas, tarefas individuais ou em grupo, informações sobre tempo, etc. Os diversos tipos de informações de contexto que podem ser utilizadas precisam estar muito bem definidas, assim as diferentes entidades componentes do ambiente pervasivo terão um entendimento comum do contexto. Também precisam atuar como mecanismos para que os usuários possam especificar como as diferentes aplicações e serviços devem comportar-se em diferentes contextos. Portanto, estes mecanismos precisam ser baseados em estruturas muito bem definidas, já que existem diferentes tipos de informações de contexto que podem ocorrer em um ambiente pervasivo. Neste sentido, as ontologias podem ser aplicadas com sucesso para esta tarefa, definindo descrições padronizadas para os diversos tipos de informações de contexto relevantes.

5.3 Modelagem do contexto

Neste trabalho, o contexto é definido como ”toda informação relevante para a aplicação e que pode ser obtida por esta”. O programador explicitamente identifica os aspectos da entidade de onde provém a informação e define seus atributos (elementos de contexto), os quais passam a integrar o contexto da aplicação. Por exemplo, um nodo de processamento poderá ter como elemento de contexto: a carga computacional, a ocupação de memória, o tamanho da fila de processos, etc. A alteração em um destes atributos poderá ser utilizada para disparar um procedimento de adaptação, tanto na aplicação como no próprio ambiente de execução (YAMIN, 2004).

A seguir é apresentada uma classificação, descrita em (YAMIN, 2004), dos principais aspectos das informações de contexto, de interesse deste trabalho, que servirão como referência para a definição do modelo de contexto que será usado no desenvolvimento da ontologia descritiva do contexto do ambiente pervasivo, a qual será apresentanda nas seções seguintes. A classificação adotada aborda três aspectos: temporal, uso e tratamento da informação.

a) Temporalidade da informação

- **Informações estáticas**: descrevem aspectos que não se alteram com o tempo, por exemplo, atributos relativos ao tipo de equipamento. As características estáticas são obtidas a partir de perfis construídos de forma automática ou pelo usuário. Estes perfis ficam registrados através de arquivos descritores de configuração.

- **Informações dinâmicas**: traduzem aspectos do contexto que oscilam com freqüência, por exemplo a ocupação do processador e a localização do usuário. Este tipo de informação é obtido através de um serviço de monitoramento, que atua periodicamente ou ativado por eventos. As informações monitoradas podem ficar imprecisas por diversos motivos; dentre estes se destacam: atrasos de propagação através da rede, desde o momento da geração até o uso da informação monitorada,

falha no sensor ou no algoritmo de tratamento, perdas de conexão com o sensor ou com o usuário da informação monitorada, etc.

b) Uso da informação de estado do contexto

- **Direto**: quando a informação monitorada pode ser utilizada de forma bruta como informação de contexto. Via de regra, traduz uma informação corrente - atual - do meio monitorado, por exemplo a ocupação do processador.
- **Interpretado**: neste caso a informação monitorada é processada antes de ser utilizada, por exemplo a localização do usuário: casa, escritório, etc.

c) Tratamento da informação obtida

- **Corrente**: neste caso a informação monitorada traduz um evento atual, podendo ser alvo de processamento como filtragem e/ou refinamento. É importante observar que as diversas aplicações podem requerer diferentes interpretações para um mesmo dado monitorado;
- **Histórica**: as informações monitoradas neste caso constituem séries históricas com o objetivo de prever o futuro. Estas séries são constituídas por registros persistentes dos dados monitorados.
- **Derivada**: as informações de contexto ainda podem ser formadas pela composição de informações mais simples. Um exemplo neste sentido diz respeito à localização; este tipo de informação pode indicar a atividade provável do usuário ou que estabelecimentos estão próximos a ele. Esta situação traduz a existência de relacionamentos entre elementos do contexto, os quais podem ser considerados para aumentar a certeza quando da interpretação do contexto.

5.3.1 Metodologia para construção da ontologia

Nesta seção será apresentada a metodologia utilizada para a construção das ontologias descritivas do ambiente pervasivo. Essa metodologia é baseada nas propostas de (FERNÁNDEZ; GÓMES-PÉREZ; JURISTO, 1997) e (GRUBER, 1993). A seguir são descritos os passos que constituem a metodologia.

- Primeiro Passo: definição do domínio e captura do conhecimento.

- Segundo Passo: conceituação do conhecimento capturado em um conjunto de representações intermediárias. As atividades realizadas neste passo são as seguintes:

- identificar as classes e suas descrições em um Glossário de Termos;
- classificar os grupos de conceitos em uma Árvore de Classificação de Conceitos;
- descrever os atributos de instâncias e os atributos de classe em Tabelas de Atributos de Instâncias e Tabelas de Atributos de Classe;
- descrever as instâncias em Tabelas de Instâncias;
- caso a ontologia possua valores numéricos inferidos a partir de atributos, descrever as fórmulas usadas para obtê-los em uma Tabela de Fórmulas;
- reunir a seqüência inferida dos atributos em Árvores de Classificação de Atributos;

- Terceiro Passo: desenvolvimento do modelo conceitual em uma linguagem formal.

Tabela 5.1: Glossário de termos da ontologia do ambiente pervasivo

Termos	Descrição
Ambiente Pervasivo	*Classe principal que contém as especificações relacionadas à: ambiente de execução, atores, dispositivos, recursos, rede e sensores*
Ambiente de Execução	*Conceito relacionado ao middleware responsável pelo gerenciamento de recursos e serviços destinados a execução das aplicações no ambiente pervasivo*
Atores	*Corresponde às pessoas e agentes que integram o ambiente pervasivo*
Dispositivos	*Conceito que representa os equipamentos que compõem o ambiente pervasivo*
Recursos	*Conceito que representa os recursos que constituem o ambiente pervasivo*
Rede	*Conceito relacionado às redes de interconexão do ambiente pervasivo*
Sensores	*Conceito relacionado aos sensores lógicos e físicos que fazem o monitoramento do ambiente pervasivo.*

5.3.2 Desenvolvimento da ontologia

Pelo exposto ao longo deste trabalho e, especialmente, nas seções anteriores, as ontologias desenvolvidas visam descrever o ambiente pervasivo e seu contexto, caracterizando a computação pervasiva como domínio para a construção destas ontologias.

Os glossários de termos apresentados nas tabelas 5.1 e 5.2 mostram as classes e suas respectivas descrições correspondentes à ontologia do ambiente pervasivo e do contexto do ambiente pervasivo, respectivamente.

As figuras 5.2 e 5.3 mostram as Árvores de Classificação de Conceitos, agrupando as classes e subclasses das ontologias do ambiente pervasivo e do seu contexto, respectivamente.

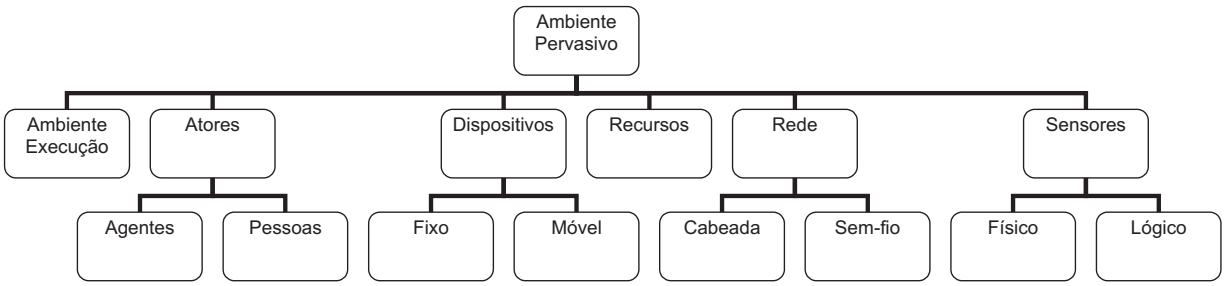

Figura 5.2: Árvore de classificação de conceitos da ontologia do ambiente pervasivo

O desenvolvimento do modelo conceitual foi feito na linguagem OWL com uso do editor Protégé. As figuras 5.4 e 5.5 mostram as classes das ontologias do ambiente pervasivo e do contexto do ambiente pervasivo construídas no editor Protégé. As figuras 5.6 e 5.7 mostram, respectivamente, a representação na linguagem OWL do conhecimento referente às ontologias do ambiente pervasivo e do seu contexto.

Tabela 5.2: Glossário de termos da ontologia do contexto do ambiente pervasivo

Termos	Descrição
Contexto do Ambiente Pervasivo	*Classe principal que contém as especificações relacionadas à: espaço, localização, papel, perfis, serviços, tempo e topologia de rede*
Espaço	*Conceito relacionado ao raciocínio sobre relações espaciais entre vários tipos de regiões geográficas, mapeando coordenadas geoespeciais em representação simbólica do espaço e vice-versa, bem como representação e a representação de medidas geográficas do espaço*
Localização	*Conceito referente à descrição do contexto detectado da localização de uma pessoa ou de um objeto. O contexto da localização é a informação que descreve onde está uma pessoa ou um objeto, incluindo propriedades temporal e espacial*
Papel	*Descreve os papéis atribuídos a agentes e a pessoas no contexto do ambiente pervasivo*
Perfis	*Representa os perfis dos dispositivos: celular, impressora, notebook, PDA e workstation*
Serviços	*Conceito relacionado aos serviços disponibilizados no contexto do ambiente pervasivo*
Tempo	*Expressa o tempo e relações temporais. Usado para descrever propriedades temporais de diferentes eventos que ocorrem no contexto do ambiente pervasivo*
Topologia de rede	*Descreve parâmetros e protocolos relacionados às rede de interconexão*

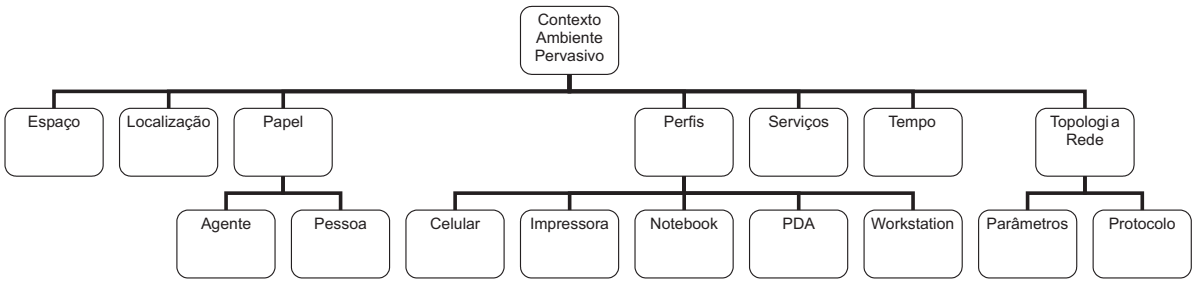

Figura 5.3: Árvore de classificação de conceitos da ontologia do contexto do ambiente pervasivo

Figura 5.4: Ontologia do ambiente pervasivo

Figura 5.5: Ontologia do contexto do ambiente pervasivo

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Ambiente_Pervasivo"/>
  <owl:Class rdf:ID="Sem-fio">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Rede"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Dispositivos">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Atores">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Recursos">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Ambiente_Execução">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Sensores">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Fixo">
    <rdfs:subClassOf rdf:resource="#Dispositivos"/>
  </owl:Class>
  <owl:Class rdf:ID="Lógico">
    <rdfs:subClassOf rdf:resource="#Sensores"/>
  </owl:Class>
  <owl:Class rdf:ID="Físico">
    <rdfs:subClassOf rdf:resource="#Sensores"/>
  </owl:Class>
  <owl:Class rdf:ID="Agentes">
    <rdfs:subClassOf rdf:resource="#Atores"/>
  </owl:Class>
  <owl:Class rdf:about="#Rede">
    <rdfs:subClassOf rdf:resource="#Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Pessoas">
    <rdfs:subClassOf rdf:resource="#Atores"/>
  </owl:Class>
  <owl:Class rdf:ID="Móvel">
    <rdfs:subClassOf rdf:resource="#Dispositivos"/>
  </owl:Class>
  <owl:Class rdf:ID="Cabeada">
    <rdfs:subClassOf rdf:resource="#Rede"/>
  </owl:Class>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)  http://protege.stanford.edu -->
```

Figura 5.6: Representação em OWL do ambiente pervasivo

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Workstation">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Perfis"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Pessoa">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Papel"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Localização">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Contexto_Ambiente_Pervasivo"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Protocolo">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Topologia_Rede"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Notebook">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Perfis"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Espaço">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Tempo">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:about="#Perfis">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="PDA">
    <rdfs:subClassOf rdf:resource="#Perfis"/>
  </owl:Class>
  <owl:Class rdf:ID="Impressora">
    <rdfs:subClassOf rdf:resource="#Perfis"/>
  </owl:Class>
  <owl:Class rdf:ID="Celular">
    <rdfs:subClassOf rdf:resource="#Perfis"/>
  </owl:Class>
  <owl:Class rdf:ID="Serviços">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Agente">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Papel"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Topologia_Rede">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
  <owl:Class rdf:ID="Parâmetros">
    <rdfs:subClassOf rdf:resource="#Topologia_Rede"/>
  </owl:Class>
  <owl:Class rdf:about="#Papel">
    <rdfs:subClassOf rdf:resource="#Contexto_Ambiente_Pervasivo"/>
  </owl:Class>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)  http://protege.stanford.edu -->
```

Figura 5.7: Representação em OWL do contexto do ambiente pervasivo

5.4 Interoperabilidade de ontologias

Considerando o processo dinâmico de agregação e desagregação de ontologias que altera o contexto do ambiente pervasivo, é necessário estabelecer uma estratégia para a interoperabilidade das ontologias. Abaixo são destacados alguns mecanismos que podem ser usados para propiciar a compatibilidade de ontologias:

- combinação de ontologias: tem-se como resultado a versão das ontologias originais combinadas em uma ontologia única com todos seus termos juntos, sem a definição clara de suas origens. Normalmente as ontologias originais descrevem domínios similares ou de sobreposição (NOY; MUSEN, 1999);

- alinhamento de ontologias: tem-se como resultado as duas ontologias originais separadas, mas com as ligações estabelecidas entre elas, permitindo que as ontologias alinhadas reusem as informações uma das outras. O alinhamento normalmente é realizado quando as ontologias são de domínios complementares (NOY; MUSEN, 1999);

- mapeamento de ontologias: tem-se como resultado uma estrutura formal que contém expressões que fazem a ligação de conceitos de um modelo em conceitos de um segundo modelo. Este mapeamento pode ser usado para transferir instâncias de dados, esquemas de integração, esquemas de combinação e tarefas similares (NOY; MUSEN, 2003);

- integração de ontologias: tem-se como resultado uma ontologia única criada pela montagem, extensão, especialização ou adaptação de outras ontologias que tratam não necessariamente do mesmo assunto. Na integração de ontologias é possível identificar as regiões que foram criadas a partir das ontologias originais (PINTO; GóMEZ-PéREZ; MARTINS, 1999).

5.5 Especificações formais com gramática de grafos

As ontologias têm sido, em muitos casos, representadas por modelos baseados em texto. Mesmo sendo de menor complexidade para construção, a estrutura de um modelo textual é de visualização complexa. Por este motivo, os grafos se mostram oportunos para representar, construir, manipular e visualizar relacionamentos estruturais de ontologias de forma simples, clara, elegante e computacionalmente tratável (SOWA, 2001). Neste sentido, para representar a dinâmica das ontologias a proposta é empregar uma gramática de grafos com extensões semânticas.

Desta forma, com o intuito de prover uma fundamentação teórica, nesta seção serão apresentados aspectos relativos ao uso da gramática de grafos em especificações formais.

A gramática de grafos é um formalismo de especificação na qual os estados de um sistema são representados por grafos e as transições são especificadas por regras. Neste formalismo os estados de um sistema são descritos através de estruturas algébricas (grafos que podem ou não ter atributos especificados como tipos de dados abstratos) e o comportamento do sistema é determinado operacionalmente por mudanças de estado.

Os grafos são meios muito naturais para explicar situações complexas em um nível intuitivo. A idéia básica da gramática de grafos é análoga à da gramática de Chomsky.

A noção resultante da gramática de grafos generaliza a da gramática de Chomsky. A composição da gramática de grafos parte de um Grafo Tipo e de um Grafo Inicial. Um Grafo Tipo descreve as possíveis ocorrências em um grafo. É uma forma simplificada, porém bastante representativa e significativa que substitui um provável grande conjunto de regras que controlariam o grafo. Um Grafo Inicial apresenta a primeira instância do grafo, nela está representado o estado inicial do sistema aguardando a aplicação de regras que o transformarão.

Ao contrário das regras de Chomsky, uma regra de grafo r: L $\rightarrow$ R não só consiste do grafo L (lado à esquerda) e R (lado à direita), mas tem também uma parte adicional: um grafo parcial (*morphism* r) mapeando extremidades e vértices em L para extremidades e vértices em R de um modo compatível. Assim, uma gramática de grafo especifica um sistema em termos de estados - modelado por grafos - e mudanças de estados - modelados por derivações. A seguinte interpretação operacional de uma regra r: L $\rightarrow$ R provê a base para esta aproximação de especificação:

- itens em L que não tem uma imagem em R são apagados;
- itens em L que não são mapeados para R são preservados;
- itens em R que não tem pré-imagem em L são criados.

Em lugar de usar grafos evidentes com vértices e extremidades, normalmente usa-se algum tipo de mecanismo dentro dos grafos, como tipos e atributos dos grafos. Aqui se usa grafos rotulados, isto é, cada vértice/extremidade tem um rótulo de algum alfabeto de rótulos.

O comportamento operacional de um sistema descrito por uma gramática de grafo é representado através da aplicação das regras de gramática de grafos para os grafos atuais. A aplicação de uma regra para um grafo atual, chamado ”passo de derivação”, é possível existir para uma ocorrência do lado à esquerda desta regra no grafo atual. Esta ocorrência, chamada ”partida”, é um morfismo de grafo total porque um espera intuitivamente que todos os elementos do lado à esquerda estejam presentes ao grafo atual para aplicar a regra.

A semântica seqüencial de uma gramática de grafos GG é determinada por todas as seqüências de passos de derivação que usam as regras de GG, começando com o grafo inicial GG, e na qual o grafo de saída de um passo é o grafo de entrada do seguinte.

Usando uma pura regra de formalismo pode-se especificar facilmente como os grafos serão transformados. Entretanto, para a especificação de quando estas transformações acontecerão, existem restrições à aplicação de condições positivas: são os vértices e as extremidades especificadas no lado à esquerda da regra que estão presentes no grafo atual, a regra pode ser aplicada se uma determinada condição for satisfeita. Os modelos semânticos mais usados para gramática de grafos são: linguagem semântica (conjunto de grafos gerados), semântica seqüencial (seqüência de transformações) e semântica concorrente (ordens parciais de transformações).

6 CONSIDERAÇÕES FINAIS

A computação pervasiva está sendo considerada o novo paradigma computacional do século XXI, bem como a SBC a destaca como um dos grandes desafios da pesquisa na área da Ciência da Computação para os próximos dez anos. Este paradigma contempla a mobilidade física e lógica em escala global, e para tanto considera que o usuário poderá acessar seu ambiente computacional independente de localização, de tempo e de dispositivo.

Na computação pervasiva um aspecto fundamental relaciona-se ao monitoramento e a manipulação das informações de contexto. Neste sentido, a computação sensível ao contexto é um paradigma computacional que se propõe a permitir que as aplicações tenham acesso e tirem proveito de informações que digam respeito às computações que realizam, buscando otimizar seu processamento.

Uma questão relevante na sensibilidade ao contexto é o grau de expressividade que se pode obter na descrição dos possíveis estados do mesmo. Neste aspecto, o uso de ontologias contribui para qualificar os mecanismos de sensibilidade ao contexto, em função da elevada expressividade que o uso destas pode propiciar.

Neste cenário, o trabalho buscou explorar a correlação entre computação pervasiva, sensível ao contexto e ontologias, avaliando o emprego de ontologias na qualificação dos mecanismos utilizados para expressar informações de contexto. Como resultado, foi possível constatar que:

- os mecanismos para sensibilidade ao contexto estão presentes nas diferentes propostas para computação pervasiva;
- os *frameworks* para sensibilidade ao contexto mais atuais já prevêem o uso de ontologias;
- a manipulação e processamento de ontologias se mostra oportuna para contextos complexos:

  a) permite o compartilhamento e a representação do conhecimento em sistemas distribuídos dinâmicos e abertos;

  b) ontologias com sua semântica declarativa provê significados para as informações contextuais;

  c) potencializam a interoperabilidade das entidades computacionais com o servidor de contexto;

  d) interpretação de contexto pode ser realizada em alto nível.

A avaliação do uso de ontologias na sensibilidade ao contexto indica que seu uso é bastante oportuno para a qualificação dos mecanismos usados para obtenção de informações de contexto. A elevação do grau de expressividade proporcionada pelas ontologias permite a identificação do contexto de interesse usando um *Reasoner* (inferência - pesquisas) em uma linguagem de alto nível, em substituição ao uso de algoritmos e estruturas de dados específicos por tipo de aplicação para o processo de tradução dos dados sensorados para contextualizados.

Como trabalhos futuros, destaca-se a continuidade da pesquisa do Mestrado em Ciência da Computação, que ocorre dentro do mesmo escopo geral deste trabalho individual, ou seja, a computação pervasiva, tendo também como tema a sensibilidade ao contexto. Assim, considerando os estudos realizados neste trabalho, a pesquisa pretende propor um mecanismo de sensibilidade ao contexto para o EXEHDA (YAMIN, 2004) baseado no emprego de ontologias.

REFERÊNCIAS

ABOWD, G.; ATKESON, C.; HONG, J.; LONG, S.; KOOPER, R.; PINKERTON, M. Cyberguide: A mobile context-aware tour guide. **Wireless Networks**, [S.l.], v.3, n.5, 1997.

AILISTO, H.; ALAHUHTA, P.; HAATAJA, V.; KYLLöNEN, V.; LINDHOLM, M. Structuring Context Aware Applications: Five-Layer Model and Example Case. , [S.l.], 2002.

APOLLO. **Apollo Project Home Page**. Disponível em: http://apollo.open.ac.uk.

AUGUSTIN, I. **Abstrações para uma Linguagem de Programação Visando Aplicações Móveis Conscientes do Contexto em um Ambiente de Pervasive Computing**. 2003. 193p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS.

AUGUSTIN, I.; YAMIN, A.; GEYER, C. Distributed Mobile Applications With Dynamic Adaptive Behavior. In: INTERNATIONAL CONFERENCE ON COMPUTERS AND THEIR APPLICATIONS, 2002, San Francisco, EUA. **Anais...** ., 2002.

BAADER, F. e. a. **The Description Logic Handbook**: Theory, Implementation and Applications. [S.l.]: Cambridge University Press, 2002.

BECHHOFER, S.; HORROCKS, I.; GOBLE, C.; STEVENS, R. OilEd: a Reasonable Ontology Editor for the Semantic Web. **Joint German/Austrian Conference on Artificial Intelligence**, Vienna, p.396–408, September 2001.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, [S.l.], v.5, n.284, p.34–43, May 2001.

BIEGEL, G.; CAHILL, V. A Framework for Developing Mobile, Context-aware Applications. In: IEEE CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS, 2., 2004. **Proceedings...** IEEE Press, 2004.

BORST, W. N. **Construction of Engineering Ontologies for Knowledge Sharing and Reuse**. 1997. 243p. PhD Thesis — University of Twente, Enschede.

BUDZIK, J.; HAMMOND, K. User Interactions with Everyday Applications as Context for Just-in-time Information Access. In: INTELLIGENT USER INTERFACES 2000, 2000. **Proceedings...** ACM Press, 2000.

CHEN, G.; KOTZ, D. A Survey of Context-Aware Mobile Computing Research. , Dept. of Computer Science, Dartmouth College, 2000.

CHEN, H. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**. 2004. 121p. Dissertation (Doctor of Philosophy) — University of Maryland, Baltimore.

CHEN, H.; FININ, T.; JOSHI, A. An Ontology for Context-Aware Pervasive Computing Environments. **Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review**, Department of Computer Science and Electrical Engineering, University of Maryland Baltimore County, v.18, n.3, p.197–207, 2004.

CHEN, H.; PERICH, F.; FININ, T.; JOSHI, A. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. **International Conference on Mobile and Ubiquitous Systems: Networking and Services**, Boston, 2004.

CORCHO, O.; FERNÁNDEZ-LÓPES, M.; GÓMEZ-PÉRES, A. OntoWeb - Tecnical Roadmap v 1.0. , [S.l.], 2001.

CORNET, R. **RICE (RACER Interactive Client Environment) Manual**. [S.l.]: Department of Medical Informatic University of Amsterdam, 2004. Technical Report 2003-01.

DEY, A. Context-aware computing: The CyberDesk project. In: SPRING SYMPOSIUM ON INTELLIGENT ENVIRONMENTS, 1998. **Anais...** AAAI, 1998.

DEY, A.; ABOWD, G. Towards a Better Understanding of Context and Context-Awareness. **Workshop on the what, who, where, when and how of context-awareness at CHI 2000**, [S.l.], Abril 2000.

DEY, A.; SALBER, D.; ABOWD, G. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. **Human-Computer Interaction**, [S.l.], v.16, 2001.

DOMINGUE, J.; MOTTA, E.; GARCIA, O. C. Knowledge Modelling in WebOnto and OCML: A User Guide. , [S.l.], 1992.

FAHY, P.; CLARKE, S. CASS - Middleware for Mobile Context-Aware Applications. **MobiSys - International Conference on Mobile Systems, Applications, and Services**, [S.l.], 2004.

FENSEL, D. Ontologies: Silver Bullet for Knowledge Management and Eletronic Commerce. **Springer - Verlag**, Berlin, 2000.

FENSEL, D.; HORROCKS, I.; VAN HARMELEN, F. OIL in a Nutshell. In: EUROPEAN WORKSHOP ON KNOWLEDGE ACQUISITION, MODELING AND MANAGEMENT, 12., 2000. **Anais...** ., 2000.

FERNÁNDEZ, M.; GÓMES-PÉREZ, A.; JURISTO, N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI) SPRING SYMPOSIUM ON ONTOLOGICAL ENGINEERING, 14., 1997, Stanford, EUA. **Anais...** AAAI press, 1997. p.33–40. (Papers from the AAAI Spring Symposium).

FLEISCHMANN, A. M. P. **Ontologias Aplicadas à Descrição de Recursos em Grids Computacionais**. 2004. 109p. Dissertação (Mestrado em Ciência da Computação) — UFSC, Florianópolis, SC.

GARLAN, D.; STEENKISTE, P.; SCHMERL, B. Toward Distraction-free Pervasive Computing. **IEEE Pervasive Computing**, New York, v.1, n.2, p.22–31, Abril 2002.

GÓMEZ-PÉREZ, A. Ontological Engineering: A state of the art. **British Computer Society**, [S.l.], v.2, n.3, p.33–43, 1999.

GRäDEL, E. Description logics and guarded fragments of first order logic. In: INTERNATIONAL WORKSHOP ON DESCRIPTION LOGICS - DL-98, 1998, Trento, Italy. **Anais...** ., 1998. p.5–7.

GRUBER, T. A Translation Approach to Portable Ontology Specifications. **Knowledge Acquisition**, [S.l.], p.199–220, 1993.

GRUBER, T. R. Ontolingua: A Mechanism to Support Portable Ontologies. **Technical Report, Knowledge Systems Laboratory**, Stanford University, Stanford, June 1992.

GRUNINGER, M. Designing and Evaluating Generic Ontologies. In: EUROPEAN CONFERENCE OF ARTIFICIAL INTELLIGENCE, 12., 1996. **Anais...** ., 1996.

GU, T.; PUNG, H.; ZHANG, D. A Middleware for Building Context-Aware Mobile Services. In: IEEE VEHICULAR TECHNOLOGY CONFERENCE, 2004, Milão, Itália. **Proceedings...** IEEE Press, 2004.

GUARINO, N. Formal Ontology and Information Systems. In: FOIS 98, 1998, Trento, Italy. **Anais...** FOIS, 1998. v.6, n.8, p.3–15.

GUSTAVSEN, R. Condor - An Application Framework for mobility-based context-aware Applications. , [S.l.], 2002.

HAARSLEV, V.; MOLLER, R. RACER System Description. **Lecture Notes in Computer Science**, [S.l.], 2001.

HOFER, T.; SCHWINGER, W.; PICHLER, M.; LEONHARTSBERGER, G.; ALTMANN, J. Context-Awareness on Mobile Devices - the Hydrogen Approach. , [S.l.], 2002.

HORROCKS, I.; PATEL-SCHNEIDER, P.; VAN HARMELEN, F. Reviewing the Design of the DAML+OIL: An Ontology Language for the Semantic Web. **18th National Conference on Artificial Intelligence**, [S.l.], 2002.

HULL, R.; NEAVES, P.; BEDFORD-ROBERTS, J. Towards situated computing. In: INTERNATIONAL SYMPOSIUM ON WEARABLE COMPUTERS, 1997. **Anais...** ., 1997.

INDULSKA, J.; SUTTON, P. Location Management in Pervasive Systems. **Conferences in Research and Pratice in Information Technology series**, [S.l.], v.21, 2003.

KARP, P. D.; CHAUDHRI, V. K.; THOMERE, J. **XOL**: An XML-Based Ontology Exchange Language.

KENT, R. Conceptual Knowledge Markup Language: The Central Core. **Twelfth Workshop on Knowledge Acquisition, Modeling and Management**, [S.l.], 1999.

KIFER, M.; LAUSEN, G.; WU, J. ogical Foudations of Object-Oriented and Frame-Based Languages. **Journal of the Association for Computing Machinery**, [S.l.], May 1995.

KORPIPää, P.; MäNTYJäRVI, J.; KELA, J.; KERäNEN, H.; MALM, E.-J. Managing Context Information in Mobile Devices. **IEEE Pervasive Computing**, [S.l.], 2003.

LASSILA, O.; SWICK, R. Resource Description Framework (RDF) Schema Specification. **W3C Recommendation**, [S.l.], 1999.

LOOM. **LOOM Project Home Page - Overview**: Loom Knowledge Representation and Reasoning System. Disponível em: http://www.isi.edu/isd/LOOM/LOOM-HOME.html Acesso em: nov. 2006.

LUKE, S.; JEFF, H. **SHOE 1.01 - Proposed Specification. SHOE Project**.

MCGUINNESS, D.; VAN HARMELEN, F. **OWL Web Ontology Language Overview. W3C Recommendation**.

NOY, N.; MUSEN, M. SMART: Automated Support for Ontology Merging and Alignment. **Banff Workshop on Knowledge Acquisition, Modeling, and Management**, Banff, Alberta, Canada, 1999.

NOY, N.; MUSEN, M. The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. **International Journal of Human-Computer Studies**, [S.l.], 2003.

OXYGEN. **OXYGEN Project Home Page**. Disponível em: http://oxygen.lcs.mit.edu Acesso em: nov. 2006.

PINTO, S.; GóMEZ-PéREZ, A.; MARTINS, J. Some Issues on Ontology Integration. **Workshop on Ontologies and Problem Solving Methods: Lessons Learned and Future Trends**, [S.l.], 1999.

PREKOP, P.; BURNETT, M. Activities, context and ubiquitous computing. **Special Issue on Ubiquitous Computing Computer Communications**, [S.l.], v.26, n.11, 2003.

ROMAN, M.; HESS, C.; CERQUEIRA, R.; RANGANAT, A.; CAMPBELL, R.; NAHRSTEDT, K. Gaia: A Middleware Infrastructure to Enable Active Spaces. **IEEE Pervasive Computing**, [S.l.], Outubro 2002.

RYAN, N.; PASCOE, J.; MORSE, D. Enhanced reality fieldwork: the contextaware archaeological assistant. **Computer Applications in Archaeology**, [S.l.], 1997.

SAHA, D.; MUKHERJEE, A. Pervasive Computing: a Paradigm for the 21st Century. **IEEE Computer**, New York, v.36, n.3, p.25–31, Mar. 2003.

SATYANARAYANAN, M. Pervasive Computing: Vision and Challenges. **IEEE Personal Communications**, New York, v.8, n.4, p.10–17, 2001.

SCHILIT, B.; THEIMER, M. Disseminating Active Map Information to Mobile Hosts. **IEEE Network**, [S.l.], 1994.

SOWA, J. **Building, Sharing and Merging Ontologies**. Disponível em: http://www.jfsowa.com/ontology/ontoshar.htm.

STUDER, R.; BENJAMINS, R.; FENSEL, D. Knowledge Engineering: Principles and Methods. **IEEE Transactions on Data and Knowledge Engineering**, [S.l.], 1998.

WANT, R.; HOPPER, A.; FALCãO, V.; GIBBONS, J. The Active Badge Location System. **ACM Transactions on Information Systems**, [S.l.], 1992.

WEISER, M. The Computer for the 21st Century. **Scientific American**, [S.l.], v.3, n.265, p.94–104, Setembro 1991.

YAMIN, A. **Arquitetura para um Ambiente de Grade Computacional Direcionado às Aplicações Distribuídas, Móveis e Conscientes do Contexto da Computação Pervasiva**. 2004. 195p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS.

YAMIN, A.; AUGUSTIN, I.; BARBOSA, J.; SILVA, L.; REAL, R.; CAVALHEIRO, G.; GEYER, C. Towards Merging Context-Aware, Mobile and Grid Computing. **International Journal of High Performance Computing Applications**, Londres, v.17, n.2, p.191–203, 2003.