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

EXEHDA-ON: Uma Abordagem Baseada em Ontologias para Sensibilidade ao Contexto na Computação Pervasiva

por
João Ladislau Barbará Lopes

Dissertação apresentada como
requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Orientador: Prof. Dr. Adenauer Corrêa Yamin
Co-orientador: Prof. Dr. Luiz Antônio Moro Palazzo

DM-2008/1-005

Pelotas, junho de 2008

À minha esposa Patrícia,
pelo apoio, incentivo e compreensão durante o desenvolvimento deste trabalho.

AGRADECIMENTOS

Aos meus pais pela dedicação, incentivo e apoio incondicional em todos os meus projetos.

Aos meus sogros pelo constante incentivo e apoio em todos os meus projetos.

Ao meu orientador, Adenauer Yamin, pela sua orientação, dedicação e disponibilidade ao longo desta dissertação e pelo apoio e amizade oferecidos nestes anos de trabalho conjunto que temos desenvolvido.

Ao meu co-orientador, Luiz Palazzo, pela sua orientação durante o desenvolvimento deste trabalho.

À UFPel, Fatec SENAC, UCPEL e CAPES pelo apoio institucional a este trabalho.

Aos professores, funcionários e colegas do PPGInf-UCPEL que contribuíram de forma direta ou indireta para a realização deste trabalho.

You see things; and you say, “Why?”
But I dream things that never were; and I say, “Why not?”
— GEORGE BERNARD SHAW

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*
DOM	*Document Object Model*
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*
OPEN	*Ontology-Driven Pervasive Environment*
OWL	*Web Ontology Language*
XML	*eXtensible Markup 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*
WASP	*Web Architectures for Services Platforms*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*

RESUMO

A Computação Pervasiva se caracteriza por propiciar ao usuário acesso a seu ambiente computacional independente de localização, tempo e equipamento. Estudos apontam que esta proposta pode ser construída pela integração da computação móvel, computação em grade e computação sensível ao contexto. O objetivo central deste trabalho é a qualificação dos mecanismos para expressar e processar informações de contexto propondo, para isso, o uso de ontologias. O EXEHDA-ON tem como premissa de pesquisa que a possibilidade de empregar uma semântica de maior expressividade que a usualmente praticada na coleta e no processamento dos dados sensorados permite atingir melhores níveis de descrição nas informações que caracterizam o contexto do ambiente computacional. Entende-se como principais contribuições deste trabalho a escolha de tecnologias para o processamento de ontologias, a proposição de um modelo ontológico para o domínio do ambiente pervasivo provido pelo EXEHDA e a integração desta proposta à arquitetura de *software* do mesmo. Os resultados obtidos com o EXEHDA-ON apontam para a viabilidade de modelar o ambiente pervasivo utilizando ontologias, bem como para o processamento do correspondente modelo ontológico na extração de dados que possibilitem as decisões de adaptação tanto do *middleware*, como das aplicações. A prototipação realizada foi suficiente para identificar um aumento na expressividade das informações contextuais e as correspondentes implicações deste aumento no processamento das mesmas, permitindo o uso de pesquisa e inferência em uma linguagem de alto nível das informações contextuais.

Palavras-chave: Computação Pervasiva, Computação Sensível ao Contexto, ontologias.

TITLE: "EXEHDA-ON: AN APPROACH BASED ON ONTOLOGIES FOR CONTEXT-AWARENESS IN PERVASIVE COMPUTING"

ABSTRACT

Pervasive Computing is characterized by providing users with access to their computational environment independent of location, time and equipment. Recent research has shown that this proposal can be built through integration of mobile computing, grid computing and context-awareness computing. The main objective of this work is the qualification of the mechanisms used to express and to process context information, proposing for this, the use of ontologies. EXEHDA-ON makes the premise that the use of semantics of higher expressiveness than is usually practiced in collection and treatment of the obtained data provides better description levels in the information that characterizes the context of the computational environment. The main contributions of the work are the selection of technologies for processing ontologies, the proposal of an ontological model for the pervasive environment domain provided by EXEHDA, and the integration of this proposal with the software architecture of EXEHDA. The results obtained with EXEHDA-ON point to the feasibility of using ontologies for modeling pervasive environments, and for processing the corresponding ontological model in the data extraction that provides adaptation decisions in middleware and in applications. The developed prototypes were enough to identify an increased expression of contextual information and the corresponding implications of this increase in its processing, allowing the use of query and inference in high-level language of contextual information.

Keywords: Pervasive Computing, Context-aware Computing, ontologies.

1 INTRODUÇÃO

Mark Weiser idealizou ambientes físicos com dispositivos computacionais integrados que auxiliariam indivíduos na realização de suas tarefas cotidianas ao fornecer-lhes informações e serviços de forma contínua e transparente. Weiser denominou de Computação Ubíqua a área de pesquisa que estuda a integração da tecnologia às atividades humanas de forma transparente, quando e onde for necessário (WEISER, 1991).

A proposta original de Weiser relativa à Computação Ubíqua previa que a interação entre usuários e computadores se distanciaria dos dispositivos tradicionais (teclado, mouse e monitor) e se aproximaria de uma interação em que os seres humanos falam, gesticulam e escrevem para se comunicarem entre si. Essa proposta ainda está distante de uma ampla aplicação prática baseada em produtos de mercado. Porém, alguns autores, como Helal (HELAL, 2005) têm apontado a gradativa concretização da proposta de Weiser, através de vários avanços tecnológicos na micro-eletrônica, nas tecnologias relativas a sensores, redes sem-fio e redes de alta velocidade, aumento contínuo do poder de processamento computacional e disponibilização de dispositivos móveis.

Com uma premissa mais próxima das atuais tecnologias de hardware e software, a Computação Pervasiva representa uma etapa em direção à consecução das propostas de Weiser (YAMIN, 2004). A Computação Pervasiva é um paradigma que permite ao usuário o acesso a seu ambiente computacional independente de localização, tempo e equipamento. O termo Computação Pervasiva (*Pervasive Computing*) foi empregado inicialmente em uma edição do IBM *System Journal* (IBM, 1999). Nesta edição, todos os artigos trataram de aspectos relativos à 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.

Trabalhos de pesquisa na área apontam 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 componentes de software devem ser conscientes de seus contextos e automaticamente adaptar-se às suas mudanças, caracterizando a sensibilidade ao contexto (RAENTO et al., 2005) (YAMIN et al., 2005) (AUGUSTIN et al., 2006).

A Computação Sensível ao Contexto vem despertando muita atenção dos pesquisadores desde a publicação de Schilit (SCHILIT; THEIMER, 1994). Diversos sistemas de sensibilidade ao contexto já foram desenvolvidos. Entretanto, a construção de sistemas sensíveis ao contexto permanece uma tarefa complexa com custo elevado para desenvolvimento, manutenção e reaproveitamento de código. Dentre os motivos que levam a essa situação destaca-se a falta de uma apropriada infra-estrutura de suporte às

aplicações. A sensibilidade ao contexto tem duas grandes frentes: (*i*) a aquisição e tratamento de dados que expressam informações relevantes sobre o contexto, e (*ii*) a adaptação às alterações de contexto. Desta forma, constitui-se em uma área na qual estão presentes diversas oportunidades de pesquisa.

Um mecanismo para sensibilidade ao contexto deve prover suporte para diferentes tarefas, dentre as quais: (*i*) obtenção do contexto de diversas fontes (sensores físicos e sensores lógicos); (*ii*) interpretação dos dados sensorados, gerando dados contextualizados e (*iii*) disseminação das informações para as partes interessadas, de modo distribuído e personalizado.

Uma questão relevante na sensibilidade ao contexto é o grau de expressividade que se pode obter na descrição dos possíveis estados do mesmo. Quanto maior a expressividade do modelo de informação do contexto, maior é a capacidade de representar a estrutura e a semântica dos conceitos. Também, quanto mais formal o modelo para descrição do contexto, maior é a capacidade de realização de pesquisa e inferência sobre ele. Neste sentido, considera-se que o uso de ontologias contribui para qualificar os mecanismos de sensibilidade ao contexto, em função da elevada expressividade que o uso destas pode propiciar.

Desta forma, este trabalho propõe um mecanismo de sensibilidade ao contexto para coletar, processar e disseminar informações de contexto na perspectiva da Computação Pervasiva, considerando o ambiente pervasivo definido no projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis Distribuídas) e provido pelo *middleware* EXEHDA (*Execution Environment for Highly Distributed Applications*) (YAMIN, 2004).

O EXEHDA-ON utiliza uma abordagem baseada em ontologias para a modelagem do contexto do ambiente pervasivo, bem como para realizar pesquisas no correspondente modelo ontológico. Este modelo ontológico tem abrangência celular, sendo alimentado por um serviço de monitoramento. Deste modo, o modelo descreve semanticamente o estado atual do ambiente, utilizando um vocabulário comum e interpretável pelos servidores de contexto existentes nas células de execução do EXEHDA. Além disso, o fato de ser utilizado um modelo ontológico também torna possível a realização de inferências sobre o estado do ambiente pervasivo, utilizando uma lógica de descrições.

Este capítulo apresenta o tema do trabalho e o contexto de pesquisa, destaca as motivações e objetivos do trabalho, bem como descreve a estrutura do texto como um todo.

1.1 Tema

Este trabalho tem como enfoque principal a elaboração de uma proposta para uso de ontologias na qualificação do mecanismo de sensibilidade ao contexto do *middleware* EXEHDA, direcionado à Computação Pervasiva.

A Computação Pervasiva tem como requisito o tratamento dos diferentes contextos que permeiam o ambiente de execução e os componentes de software da aplicação. As informações de contexto são a base do processo de adaptação que se faz necessário para atender as demandas da computação pervasiva (AUGUSTIN et al., 2006). Deste modo, a sensibilidade ao contexto se mostra um dos grandes desafios desta área de pesquisa.

O tema deste trabalho abrange estudos que visam propor o emprego de ontologias na qualificação dos mecanismos utilizados para expressar informações de contexto, ex-

plorando a correlação entre Computação Pervasiva, Computação Sensível ao Contexto e ontologias.

1.2 Contexto de pesquisa

O EXEHDA-ON está inserido nos esforços de pesquisa dos projetos OPEN e EXEHDA, os quais são sumarizados nesta seção.

1.2.1 Projeto OPEN

O objetivo central do projeto OPEN (*Ontology-Driven Pervasive Environment*) (OPEN, 2006) é o desenvolvimento de metodologias e ferramentas para a gestão de ambientes pervasivos abertos através do emprego de ontologias dinâmicas.

O projeto OPEN propõe o desenvolvimento de um ambiente pervasivo aberto, capaz de reconhecer e processar a entrada e saída de agentes, empregando modelos de integração de ontologias para definir a mobilidade de agentes no ambiente pervasivo em tempo real, permitindo a auto-configuração dinâmica de recursos e serviços de modo a otimizar o atendimento das demandas dos agentes.

O projeto OPEN prevê o uso do EXEHDA como *middleware* direcionado à Computação Pervasiva. Neste sentido, o EXEHDA-ON busca atender as demandas do projeto OPEN através do emprego de ontologias na qualificação do mecanismo de sensibilidade ao contexto do EXEHDA.

1.2.2 *Middleware* EXEHDA

O EXEHDA é um *middleware* adaptativo ao contexto e baseado em serviços que visa criar e gerenciar um ambiente pervasivo, bem como promover a execução, sob este ambiente, das aplicações que expressam a semântica siga-me. Estas aplicações são distribuídas, móveis e adaptativas ao contexto em que seu processamento ocorre, estando disponíveis a partir de qualquer lugar, todo o tempo.

O *middleware* EXEHDA faz parte dos esforços de pesquisa do Projeto ISAM. O ISAM vem sendo desenvolvido por um consórcio de universidades gaúchas, e foi iniciado na UFRGS sob a coordenação do Prof. Cláudio Geyer (ISAM, 2007).

Para atender a elevada flutuação na disponibilidade dos recursos, inerente à computação pervasiva, o EXEHDA é estruturado em um núcleo mínimo e em serviços carregados sob demanda. Os principais serviços fornecidos estão organizados em subsistemas que gerenciam: a execução distribuída, a comunicação, o reconhecimento do contexto, a adaptação, o acesso pervasivo aos recursos e serviços, a descoberta e o gerenciamento de recursos.

No EXEHDA, 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. O mecanismo de adaptação do EXEHDA emprega uma estratégia colaborativa entre aplicação e ambiente de execução, através da qual é facultado ao programador individualizar políticas de adaptação para reger o comportamento de cada um dos componentes que constituem o software da aplicação (YAMIN, 2004).

1.3 Motivação

Desde as publicações de (SATYANARAYANAN, 2001) e (SAHA; MUKHERJEE, 2003) já se observava uma significativa elevação dos níveis de mobilidade, heterogeneidade e interação entre dispositivos conectados a redes globais. Atualmente, é possível vislumbrar que estas previsões estão rapidamente se concretizando e apresentam uma forte tendência de intensificação nos próximos anos.

Diversas questões pertinentes ao gerenciamento de recursos físicos foram respondidas pelas primeiras pesquisas envolvendo sistemas distribuídos em redes de grande abrangência. 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) (REAL et al., 2003).

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).

Embora sendo uma área recente de pesquisa, a Computação Pervasiva já 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” (SBC, 2006).

As ontologias vêm 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. Também, viabiliza a realização de pesquisa e inferência sobre o modelo ontológico (GUARINO, 1998).

Deste cenário decorre a principal motivação para este trabalho, o qual pretende apresentar uma proposta para o emprego de ontologias na qualificação do mecanismo de sensibilidade ao contexto do *middleware* EXEHDA.

1.4 Objetivos

O objetivo geral deste trabalho é a proposição de um mecanismo de sensibilidade ao contexto voltado à Computação Pervasiva que contemple o uso de ontologias, integrado ao *middleware* EXEHDA.

O EXEHDA-ON foi concebido como extensão de um *middleware* para suporte à execução de aplicações da Computação Pervasiva, contemplando os seguintes objetivos específicos:

- manter caracterizadas as frentes de pesquisa relativas à proposição de *middlewares* para a Computação Pervasiva;

- atualizar e revisar os princípios que devem nortear a concepção do EXEHDA enquanto *middleware* para Computação Pervasiva;
- garantir um modelo computacional para o EXEHDA que possa servir de base para implementação de modelos computacionais de mais alto nível direcionados à Computação Pervasiva;
- conceber um modelo ontológico para suporte a sensibilidade ao contexto na Computação Pervasiva;
- selecionar tecnologias para o processamento do modelo ontológico proposto para o ambiente pervasivo gerenciado pelo EXEHDA.
- definir a arquitetura de software do EXEHDA-ON e as funcionalidades dos diversos serviços que a compõe;
- difundir o conhecimento pertinente à área de suporte à execução de aplicações na Computação Pervasiva, sobretudo no que diz respeito aos aspectos de sensibilidade ao contexto;
- fornecer subsídios para a elaboração de relatórios, artigos e trabalhos futuros, relacionados com o tema pesquisado.

1.5 Estrutura do texto

O texto é composto por seis capítulos. O Capítulo 2 resume os principais conceitos de Computação Pervasiva, bem como introduz as potencialidades e desafios propostos por este novo paradigma.

No Capítulo 3 são apresentados os fundamentos teóricos da Computação Sensível ao Contexto, bem como são descritos os trabalhos relacionados ao EXEHDA-ON, destacando aspectos relacionados com arquitetura, modelagem, processamento e armazenamento de contexto.

O Capítulo 4 apresenta os fundamentos teóricos de ontologias, bem como descreve aspectos relativos ao projeto de ontologias, destacando linguagens e ferramentas voltadas à construção das mesmas. Ainda, este capítulo trata da representação do conhecimento e raciocínio baseados na lógica de descrições. Também, são apresentadas as principais características e funcionalidades do *middleware* EXEHDA, destacando o subsistema de reconhecimento de contexto e adaptação, o qual está diretamente relacionado com a proposta deste trabalho.

No Capítulo 5 são tratados os aspectos de modelagem e prototipação EXEHDA-ON, sendo discutidos aspectos pertinentes a concepção da sua arquitetura de software, bem como do modelo ontológico projetado para representar o ambiente pervasivo previsto. Também são discutidos aspectos relacionados à prototipação e testes do EXEHDA-ON, destacando as tecnologias utilizadas e os casos de emprego do EXEHDA-ON.

Por fim, no Capítulo 6 o trabalho é sumarizado, sendo apresentadas as principais conclusões, as contribuições da pesquisa, publicações realizadas e trabalhos futuros.

2 SENSIBILIDADE AO CONTEXTO NA COMPUTAÇÃO PERVASIVA

Neste capítulo são resumidos conceitos de Computação Pervasiva, bem como são introduzidas as potencialidades e desafios propostos por este novo paradigma. Também são apresentados projetos voltados à Computação Pervasiva.

2.1 Conceitos associados

A Computação Pervasiva é um paradigma computacional que objetiva disponibilizar 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 Computação Pervasiva também é associada aos termos: Computação Ubíqua e *Calm Technology*. O objetivo é integrar as máquinas ao mundo “real”, tornando os dispositivos capazes de detectar mudanças em seus ambientes e automaticamente adaptarem-se e agirem de acordo com estas mudanças, baseando-se nas necessidades e preferências do usuário. Este paradigma computacional se mostra uma etapa indispensável no caminho de propostas como a de Weiser (WEISER, 1991) citada no capítulo 1.

A disseminação física da Internet e o aumento da velocidade operacional das redes de computadores que a compõem conduzem a uma perspectiva de uso unificado dos recursos distribuídos, o qual pode ser realizado a partir de equipamentos posicionados em qualquer local das redes interconectadas. Esta situação materializa as premissas da Computação em Grade (*Grid Computing*). Da mesma forma, o crescente uso das redes sem fio conduz a uma proposta efetiva de Computação Móvel, na qual o usuário utilizando dispositivos portáteis poderá ter acesso a uma infra-estrutura de serviços e manterá este acesso durante seu deslocamento (BALDAUF; DUSTDAR; ROSENBERG, 2007).

Este cenário prevê uma mobilidade física (dos equipamentos e/ou dos usuários) e do software (componentes da aplicação e serviços). Para isto, o sistema de suporte à execução deve permitir que o usuário possa acessar seu ambiente computacional (dados e configurações pessoais) independente de localização e de tempo. Este sistema usa a metáfora de um ambiente virtual do usuário, onde as aplicações têm o estilo *siga-me (follow-me applications)*. Logo, para viabilizar esse novo estilo de programação é necessário uma infra-estrutura de suporte para o projeto, implementação e execução do

software a ser desenvolvido (LOPES; PILLA; YAMIN, 2007).

A classe de aplicações que surge deste cenário é sensível ao contexto onde ocorre a execução, assim o programador deve prever o uso das informações de contexto nas alterações funcionais e não-funcionais. Essa premissa contrasta com a computação distribuída tradicional que busca liberar os programadores de considerar o estado do meio físico no qual acontece o processamento.

As condições de contexto são pró-ativamente monitoradas e o modelo de execução deve permitir que tanto a aplicação como o próprio *middleware* de gerência reajam às alterações no ambiente através de mecanismos de adaptação. Este processo requer a existência de múltiplos caminhos de execução, ou de configurações alternativas, as quais exibem diferentes perfis de utilização dos recursos computacionais. A efetividade de um comportamento adaptativo ao contexto durante a execução de uma aplicação distribuída, em ambiente com suporte à mobilidade física, tanto do usuário como do equipamento, envolve diversos aspectos. A seguir um resumo feito a partir de (YAMIN, 2004):

- disponibilização de um ambiente computacional pervasivo para gerenciamento das aplicações;
- elevada heterogeneidade na disponibilidade dos recursos, mesmo quando os equipamentos envolvidos são constituídos de hardware e software similares;
- relação entre os componentes de hardware e software ativos da aplicação, na perspectiva da mobilidade lógica e física;
- disponibilidade de dados e código na perspectiva da mobilidade, ou seja, a partir de qualquer posição da infra-estrutura de rede;
- preservação da consistência da execução distribuída, na presença de desconexões assíncronas de nodos de processamento;
- gerenciamento, de forma personalizada, da disponibilização do estado atual do contexto com significado para cada aplicação;
- definição da arquitetura do ambiente de execução para que os processos de adaptação ao contexto ocorram de forma integrada aos mecanismos de gerência do processamento;
- interoperabilidade, considerando a pluralidade de natureza do hardware e sistemas operacionais na Computação Pervasiva.

Estes aspectos ressaltam que, independentemente dos mecanismos de sensibilidade ao contexto e de gerência da adaptação, a premissa siga-me da Computação Pervasiva deverá ser suportada, garantindo a execução da aplicação do usuário a partir de qualquer localização e tipo de equipamento, todo o tempo (YAMIN et al., 2005).

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, 2004).

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 trabalhados diversos aspectos, dentre estes:

- 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 Desafios de pesquisa identificados

A premissa básica dos sistemas distribuídos tradicionais de tornar o mais transparente possível ao usuário aspectos de gerência física da execução é a principal motivação de diversos projetos que implementam *middlewares*. Esta premissa é adequada para ambientes distribuídos com recursos de baixa heterogeneidade, cuja disponibilidade é previsível. A grande maioria dos projetos trata os itens necessários à pervasividade de forma independente: comunicação, adaptação, reconhecimento do contexto, gerenciamento de recursos, computação em grade. Nesta seção serão abordados projetos voltados para a execução de aplicações e/ou criação de um ambiente *pervasivo* que integre diversos aspectos inerentes a pervasividade.

Projeto Gaia

O Projeto Gaia propõe a construção dos chamados “Espaços Ativos”, os quais traduzem uma visão de futuro onde o espaço habitado pelas pessoas é interativo e programável. Na perspectiva do Projeto Gaia, os usuários interagem com seus escritórios, casa, carros, etc., para requisitar informações, beneficiar-se dos recursos disponíveis e configurar o comportamento de seu habitat. Dados e tarefas estão sempre acessíveis e são mapeados dinamicamente para os recursos convenientes e presentes na localização corrente. Este ambiente interativo, centrado no usuário, requer uma nova infra-estrutura de software para operar com os recursos, observar propriedades do contexto, assistir o desenvolvimento e execução das aplicações (ROMAN et al., 2002).

Gaia é um *middleware* usado para prototipar gerenciamento de recursos e fornecer uma interface orientada ao usuário (ROMAN et al., 2002). A figura 2.1 mostra a arquitetura proposta. GaiaOS apresenta-se como um meta-sistema operacional que objetiva suportar e executar aplicações portáteis em Espaços Ativos, roda sobre sistemas existentes e é composto por quatro partes principais: *Unified Objet Bus*, *Gaia Kernel*, *Gaia Application Model* e *Active Space Execution Environment*. Gaia coordena as entidades de software e dispositivos em rede dentro dos Espaços Ativos, exporta serviços para pesquisar e utilizar recursos, acessar e usar o contexto corrente, e também fornece um *framework* para desenvolver aplicações.

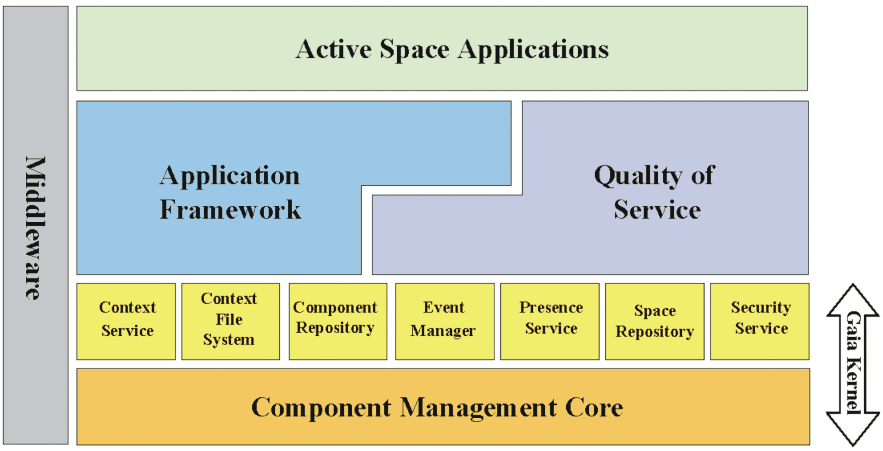

Figura 2.1: Arquitetura do Gaia
(ROMAN et al., 2002)

A estrutura do Gaia tem três grandes blocos: *kernel, framework* e aplicações. O *kernel* é composto pelo gerenciamento dos componentes e pelos serviços fornecidos. O gerenciamento de componentes distribuídos inclui carregar, descarregar, transferir, criar e destruir todos os componentes e aplicações Gaia. Os serviços são ativados de forma particularizada à localização, contexto, eventos, repositórios com informações sobre os espaços ativos. O contexto refere-se a informações de presença de objetos e pessoas, e condições ambientais como temperatura e som.

O *framework* fornece mecanismos para construir, executar ou adaptar aplicações existentes para espaços ativos (ROMAN et al., 2002), sendo composto por uma infra-estrutura de objetos distribuídos, um mecanismo de mapeamento, e políticas de customização das aplicações. A implementação do Gaia define quatro entidades básicas: pessoas, dispositivos, serviços e aplicações. Os recursos são descritos através de suas propriedades, expressas em XML (HESS; ROMAN; CAMPBELL, 2002).

No projeto Gaia torna-se visível a preocupação com a sensibilidade ao contexto. O próprio conceito de Espaços Ativos do Gaia, no qual o espaço habitado pelas pessoas é interativo e programável, indica que deve haver mecanismos de controle/monitoração de informações de contexto, tais como: presença de objetos, informações pessoais, condições ambientais.

Projeto Aura

O projeto Aura 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 projeto com diversas frentes que investiga novas arquiteturas para o ambiente pervasivo. O projeto tem seu foco no usuário, suas tarefas e preferências (GARLAN; STEENKISTE; SCHMERL, 2002). Desta forma, o conceito de contexto presente na proposta do Aura prevê a ênfase na dimensão pessoal.

Aura visa dar suporte computacional a cenários de aplicações, tais 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 PDA e caminha para a porta. Aura transfere o estado do seu trabalho do *desktop* para o PDA, 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 do Aura (Figura 2.2), 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.

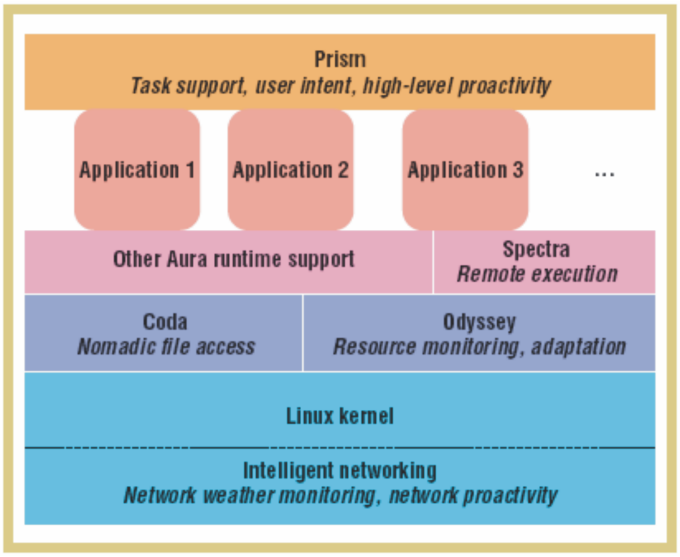

Figura 2.2: Arquitetura do Aura
(GARLAN; STEENKISTE; SCHMERL, 2002)

O projeto contempla esforços que incluem (*i*) inferência de tarefas, (*ii*) representação das tarefas e (*iii*) alocação de recursos (GARLAN; STEENKISTE; SCHMERL, 2002). Neste projeto, nota-se preocupação com o contexto, porém esta é maior no que tange à dimensão pessoal do usuário, que é o foco do projeto Aura.

Projeto Oxygen

O Projeto Oxygen (OXYGEN, 2004) tenta atingir os objetivos da Computação Pervasiva através da combinação de tecnologias de usuário e tecnologias de sistema. As tecnologias de usuário atendem diretamente às necessidades humanas. De acordo com as premissas do projeto, tecnologias como visão e fala permitem interagir diretamente com Oxygen como se estivesse interagindo com uma outra pessoa qualquer, poupando muito tempo e esforço. A automatização, o acesso individualizado ao conhecimento, e as tecnologias de colaboração ajudam a executar uma grande variedade de tarefas. Para suportar atividades humanas altamente dinâmicas e variadas, o Projeto Oxygen se propõe a superar muitos desafios técnicos:

- estar em todo lugar, com cada portal (interface) podendo acessar as mesmas bases de informação;
- interoperar no mundo real, sentindo e afetando-o, diretamente;
- permitir ao usuário e seus ambientes computacionais mover-se livremente para qualquer lugar, de acordo com suas necessidades;
- prover flexibilidade e espontaneidade, em resposta às mudanças das necessidades dos usuários condições operacionais;
- livrar-se das restrições impostas pelos limites de hardware, indo de encontro somente às restrições de sistemas impostas pelas necessidades dos usuários e fonte de energia disponível ou largura de banda para comunicações;
- permitir às pessoas nomearem 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;
- nunca parar ou reinicializar, os componentes poderão ir e vir de acordo com a demanda de erros e *upgrades*, mas o Oxygen como um todo deverá estar disponível e operacional todo o tempo.

Oxygen visa atingir esses objetivos com o desenvolvimento de tecnologias de dispositivos, redes e software, levando a computação para a casa dos usuários, escritório ou onde quer que ele esteja. Os dispositivos computacionais no Projeto Oxygen são chamados de "Enviro21s (E21s)". A proposta é que estes sejam embutidos nas casas, escritórios e carros, sensoriando e influenciando imediatamente o ambiente. Oxygen chama os dispositivos *Handheld* por "Handy21s (H21s)" e, tornando as pessoas capazes de se comunicar e utilizar seus ambientes computacionais não importando onde estiverem. Redes dinâmicas e auto-configuráveis "(N21s)", propõem-se a ajudar as máquinas a localizarem umas às outras assim como as pessoas, serviços e recursos que deseja-se obter alcançar ou encontrar. O software se adapta às mudanças no ambiente ou às necessidades do usuário "(O2S)", ajudando-o a fazer o que ele desejar e quando desejar, conforme ilustra a figura 2.3

Nota-se que no Oxygen o contexto é construído com o auxílio dos dispositivos de ambiente chamados "E21s" e um componente de software chamado "O2S", encarrega-se de efetuar as adaptações necessárias de acordo com as mudanças de contexto no ambiente, evidenciando a preocupação com a sensibilidade ao contexto.

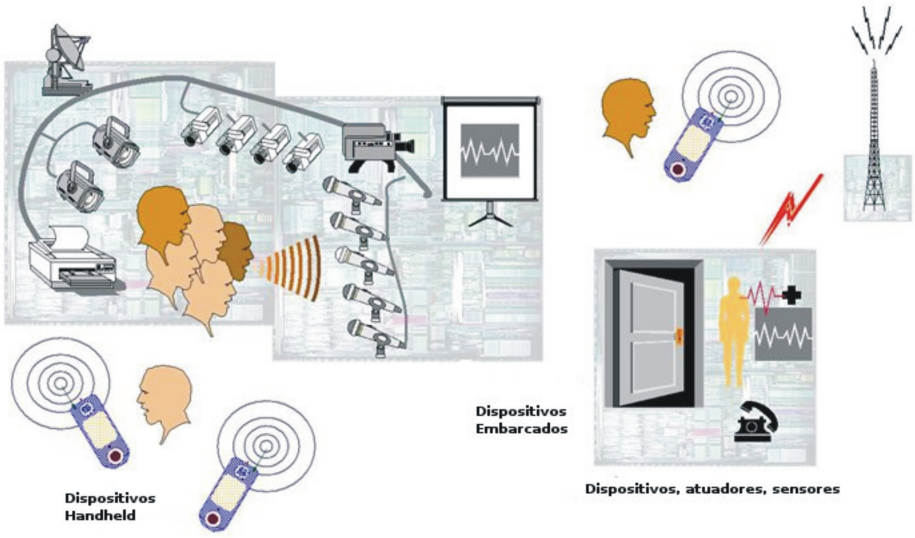

Figura 2.3: Ambientes do Oxygen
(OXYGEN, 2004)

Projeto Endeavour

No Projeto Endeavour (ENDEAVOUR, 1999) é usada a metáfora do mar e da maneira como ele "interconecta" o mundo. Sob o ponto de vista do projeto, os sistemas de informação são "fluídos", que existem todo o tempo e estão em todo lugar, são componentes que "fluem" através das infra-estruturas, transformando-se para adaptar-se ao seu uso e para cooperação em tarefas. O objetivo é criar um "Utilitário" para informação pervasiva, baseado em uma nova tecnologia de sistemas "fluídicos", trazendo novas alternativas para a resolução de problemas e aprendizado.

As suposições tecnológicas feitas pelo Endeavour dizem que servidores e *desktops* terão maior capacidade, os dispositivos serão muito mais heterogêneos e o gerenciamento da informação será melhor integrada. Porém, uma parte muito grande do tempo será gasta capturando, estruturando, organizando e armazenando um gigantesco volume de informação levando em conta também as mudanças de ambiente. O único elemento comum neste ambiente altamente heterogêneo será a comunicação, os dispositivos serão tão especializados que será incomum encontrar um, com configurações de hardware e poder de processamento semelhantes.

Em sua primeira fase, o projeto Endeavour envolve a definição da arquitetura e a prova de conceitos em quatro áreas críticas: Dispositivos de Informação, Utilitários de Informação, Aplicações e Metodologia e *Design*.

O Utilitário de Informação é o núcleo da pesquisa proposta. O Projeto pretende desenvolver a tecnologia necessária para suportar o software fluídico: mecanismos para processamento, armazenamento, e gerenciamento das informações que fluem com o sistema, dividindo-se em componentes, que escolhem onde executar, e como acessar o meio de armazenamento em qualquer lugar através de ambientes interconectados. Os componentes de software fazem o processo de descoberta (*discover*) e negociação de acordos de interoperabilidade (*matchmaking*) uns com os outros, em escalas locais e globais, provendo processamento cooperativo. O Utilitário também oferece mecanismos para acomodar uma grande variedade de entradas e saídas, e dispositivos de acesso à informação.

Deve ser implementado para estar sempre disponível, processado e administrando a escalabilidade, explorando o paradigma de software fluídico.

A segunda fase do projeto é destinada para experimentações e testes das aplicações e componentes provenientes da primeira fase. A terceira fase consiste em uma arquitetura refinada e criação plataformas de testes em colaboração com os parceiros do projeto. Uma representação das etapas de desenvolvimento pode ser visualizada na figura 2.4.

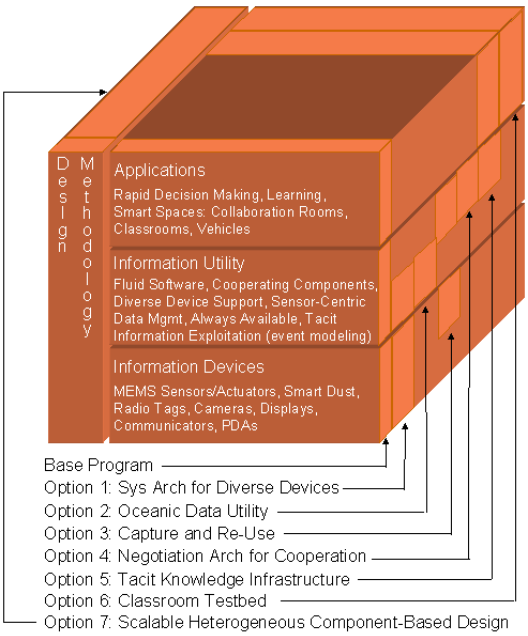

Figura 2.4: Etapas de desenvolvimento de Endeavour
(ENDEAVOUR, 1999)

Neste projeto, quando afirma-se que os sistemas de informação são fluidos, e que eles fluem através das infra-estruturas transformando-se e adaptando-se, pode-se deduzir a existência de mecanismos que possibilitem a sensibilidade ao contexto, conseqüentemente viabilizando a transformação e adaptação destes "fluidos" de informação no projeto Endeavour.

2.3 Considerações finais

A abordagem de construção da Computação Pervasiva e os projetos apresentados neste capítulo apontam a Computação Sensível ao Contexto como um dos componentes centrais para a concretização das premissas da Computação Pervasiva. Esta constatação corrobora com os objetivos deste trabalho, reforçando a motivação central do mesmo. No próximo capítulo serão apresentados os principais aspectos relacionados com a Computação Sensível ao Contexto, com intuito estabelecer uma fundamentação teórica, necessária à consolidação da proposta do EXEHDA-ON.

3 ESCOPO DO TRABALHO

Neste capítulo são apresentados fundamentos teóricos da Computação Sensível ao Contexto, destacando seus conceitos e princípios gerais, a definição, classificação, características e modelagem do contexto, bem como a arquitetura para aplicações sensíveis ao contexto. Ainda, são apresentados trabalhos relacionados ao EXEHDA-ON, também direcionados a dar suporte à construção de aplicações sensíveis ao contexto. São destacados aspectos de arquitetura de software destes projetos, bem como as estratégias utilizadas pelos projetos para modelagem, processamento e armazenamento das informações de contexto. O estudo dos fundamentos e de projetos em Computação Sensível ao Contexto, sumarizado neste capítulo, subsidiou a definição das premissas a serem seguidas na concepção do modelo para o EXEHDA-ON. Os projetos são sistematizados em quatro grandes categorias: arquitetura, modelagem, processamento e armazenamento de contexto.

3.1 Conceitos em 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 (BALDAUF; DUSTDAR; ROSENBERG, 2007).

Considera-se como marco inicial dos sistemas computacionais sensíveis ao contexto a aplicação proposta por (WANT et al., 1992) em 1992. O *Active Badge Location System* era baseado na tecnologia de infravermelho e conseguia determinar a localização atual do usuário, a qual era usada para encaminhar as ligações telefônicas para o telefone mais próximo. 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.

3.1.1 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.

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.

O contexto é trabalhado 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. Esta definição de contexto é considerada como base para a proposta direcionada à sensibilidade ao contexto apresentada neste trabalho.

Nesta perspectiva, o programador explicitamente identifica os aspectos da entidade de onde provéem 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 ambiente pervasivo, a qual será apresentanda nas seções seguintes. A classificação adotada aborda três aspectos: temporal, uso e tratamento da informação.

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.

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.

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.

3.1.2 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 externos, 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 lida-se 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).

3.1.3 Características das informações de contexto

As informações de contexto possuem características bastante peculiares que devem ser consideradas ao se construir aplicações pervasivas sensíveis ao contexto (HENRICKSEN; INDULSKA, 2006):

- **Características temporais**: pode-se caracterizar uma informação de contexto como sendo estática ou dinâmica. Informações contextuais estáticas descrevem aspectos invariáveis dos sistemas, como tipo de um dispositivo (fixo ou móvel, por exemplo). Já as informações dinâmicas, as mais comuns em aplicações pervasivas, variam freqüentemente. A persistência de uma informação de contexto dinâmica é altamente variável, por exemplo, a localização e a atividade de uma pessoa freqüentemente se alteram. A característica de persistência influencia e determina quando uma determinada informação deverá ser adquirida. As informações estáticas podem ser obtidas diretamente com o usuário ou através de arquivos de configuração dos sistemas. As informações dinâmicas devem ser obtidas instantaneamente ou periodicamente do ambiente.

- **Informação imperfeita**: a informação pode estar incorreta caso não reflita o verdadeiro estado do mundo que ela modela, inconsistente se contém informação contraditória, ou incompleta se sob alguns aspectos o contexto não é reconhecido. Em ambiente extremamente dinâmico como o da Computação Pervasiva, a informação

de contexto rapidamente se torna obsoleta, não refletindo o ambiente que deveria representar. Isso ocorre pelo fato de freqüentemente as fontes, os repositórios e os consumidores de contexto estarem distribuídos, gerando muitas vezes um atraso entre o envio e a entrega das informações de contexto. Além disso, os produtores de contextos, como sensores, algoritmos de derivação e usuários, podem prover informação imperfeita; por exemplo, quando a atividade de uma pessoa é inferida indiretamente a partir de sua localização e do nível de ruído ao seu redor. Finalmente, desconexões dos canais de comunicação, interferências e outras falhas podem ocorrer no caminho entre e o envio e a entrega da informação de contexto.

- **Representações alternativas**: geralmente existe uma grande diferença entre o que é obtido dos sensores e a abstração entendida pelas aplicações. Essa diferença de abstração se deve aos tratamentos e processamentos que uma informação de contexto deve passar. Por exemplo, um sensor de localização fornece as coordenadas geográficas de uma pessoa ou de um dispositivo, enquanto uma aplicação está interessada na identidade do prédio ou da sala em que o usuário está. Observe que os requisitos e níveis de abstração que uma informação de contexto exige podem variar de uma aplicação para a outra. Portanto, um modelo de contexto deve suportar múltiplas representações do mesmo contexto em diferentes formas e em diferentes níveis de abstração, e ainda ser capaz de entender os relacionamentos entre essas representações alternativas.

- **Elevado inter-relacionamento**: diversos relacionamentos entre as informações de contexto são evidentes, por exemplo, proximidade entre usuários e seus dispositivos. Entretanto, outros tipos de relacionamentos entre informações de contexto não são tão óbvios. Estas podem estar relacionadas entre si através de regras de derivação que descrevem como uma informação de contexto é obtida a partir de uma ou mais informações. Por exemplo, a atividade em execução pode ser derivada da localização corrente do usuário e de informações de atividades passadas nesta localização.

O modelo de contexto deve ser projetado para capturar as informações relevantes (elementos de contexto) para o projeto e construção de sistemas e aplicações que devem se ajustar ao contexto corrente ou previsto. Este deve prover informações de vários tipos. As informações de contexto podem ser sensoradas, derivadas ou fornecidas pelo usuário, como suas preferências. Podem ser correntes ou históricas. As informações podem ser simples - obtidas de uma única fonte, ou compostas - obtidas de várias fontes através do conceito de coleção (média, máximo, etc.), alternativas (ou, e). As informações também podem ser relativas ao tempo (temporal), a localização geográfica (espacial), ao usuário (pessoal) ou grupos de usuários (social) (AUGUSTIN, 2004).

3.1.4 Interpretação de contexto

A interpretação de contexto pode ser entendida como o conjunto de métodos e processos que realizam a abstração, o mapeamento, a manipulação, a agregação, a derivação, a inferência e demais ações sobre as informações contextuais, com o propósito de facilitar o entendimento de um determinado contexto pelas aplicações e auxiliá-las na tomada de decisões. O processo de interpretação de contexto consiste na manipulação e refinamento das informações de contexto de um ambiente.

Em (DEY; SALBER; ABOWD, 2001), um dos primeiros trabalhos a ampliar o conceito de contexto para aspectos lógicos, a interpretação de contexto é vista como o processo de se elevar o nível de abstração das informações de contexto de um ambiente, ou seja, gerar uma informação de contexto mais elaborada a partir de uma mais primitiva. O processo de interpretação de contexto pode ser bastante simples como derivar o nome de uma rua a partir de suas coordenadas geográficas ou bastante complexo e oneroso como inferir o humor de um usuário baseado em seu perfil e na atividade em que ele está realizando. Além disso, o ambiente da Computação Pervasiva é extremamente dinâmico e complexo. As informações de contexto podem estar espalhadas e distribuídas em qualquer lugar e com alto grau de mobilidade. Essa complexidade faz com que haja a necessidade de um suporte computacional às aplicações, de maneira a auxiliá-las na realização de interpretações de contextos. Tais atividades onerosas devem ser abstraídas das aplicações e um módulo Interpretador de Contexto torna-se, portanto, um componente essencial em uma infra-estrutura de suporte a aplicações pervasivas sensíveis ao contexto. Um Interpretador de Contexto deve ser capaz de obter e prover informação de contexto em diferentes níveis de abstração, conforme o desejo do usuário e de suas aplicações. Uma aplicação pode desejar tanto informações mais brutas, de mais baixo nível ou informações mais abstratas e elaboradas, de mais alto nível, provenientes de um processo de refinamento e interpretação.

3.1.5 Modelo de contexto

Os modelos de contexto podem ser classificados, segundo (STRANG; LINNHOFF-POPIEN, 2004), em modelos de pares atributos-valor, baseados em esquemas, modelos gráficos, modelos orientados a objetos, baseados em lógica e baseados em ontologias.

- **Modelos de pares atributo-valor**: uma informação de contexto é descrita como um conjunto de pares atributos-valor, no qual um atributo descreve uma propriedade do contexto em questão. Este é o modelo mais elementar e de mais simples implementação, mas de utilização limitada devido à ausência de tipagem e inter-relacionamento entre contextos. Este modelo foi utilizado em vários projetos de computação consciente de contexto, tais como: (SCHILIT; ADAMS; WANT, 1994).

- **Modelos baseados em esquemas**: contextos são representados como estruturas de dados hierárquicas, compostas de pares atributos-valores, e descritas por linguagens de marcadores como dialetos do XML. Diferentemente do modelo anterior, no modelo baseado em esquemas a informação de contexto possui uma estrutura bem definida e permite a composição de informações de contexto. Este modelo tem sido amplamente utilizado na representação de perfis que permitam adaptação na web, como ocorre (INDULSKA et al., 2003).

- **Modelos gráficos**: o contexto é modelado por meio de linguagens gráficas, tais como: extensões da *Unified Modeling Language* (UML), conforme apresentado por (HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2003). Modelos baseados em UML são capazes de modelar ricas composições e inter-relacionamentos entre informações de contexto, embora sejam limitados quanto à modelagem do comportamento dinâmico do contexto.

- **Modelos orientados a objetos**: as informações de contexto são representadas por meio de modelos de objetos, fazendo o uso de encapsulamento, reutilização e abstração. Esta abordagem foi utilizada em projetos como o GUIDE (CHEVERST; MITCHELL; DAVIES, 1999).

- **Modelos baseados em lógica**: o contexto é representado por meio de fatos, expressões e regras que especificam como uma informação de contexto pode ser inferida ou derivada a partir de outra. Este modelo utiliza um alto grau de formalização e foi objeto de trabalhos como (MCCARTHY; BUVAC, 1997).

- **Modelos baseados em ontologias**: contextos e seus inter-relacionamentos são descritos por meio de ontologias. Devido à sua capacidade de descrever contextos complexos, potencialidades para compartilhamento de conhecimento, inferência e reutilização; este modelo têm sido utilizado em vários projetos, tais como: (CHEN, 2004) e (GU; PUNG; ZHANG, 2004).

(STRANG; LINNHOFF-POPIEN, 2004) discutem as fraquezas e qualidades de cada um desses seis modelos de contexto, tendo como base os seguintes requisitos para Computação Pervasiva:

- **composição distribuída (CD)**: a composição e administração dos modelos de contexto são extremamente dinâmicas em termos do tempo, topologia da rede e recursos;

- **validação parcial (VP)**: capacidade para validar conhecimento parcial. Em determinados momentos, devido a composição distribuída, não é possível validar todo o conhecimento de contexto;

- **qualidade da informação (QI)**: a qualidade da informação muda de acordo com o sensor utilizado, entre outros fatores. Os métodos devem suportar o tratamento de informação com distintos níveis de qualidade;

- **incompleteza e ambiqüidade (IA)**: o método deve ser capaz de tratar informação incompleta e ambígua;

- **nível de formalidade (NF)**: modelos com sintaxe e semânticas bem definidas;

- **aplicabilidade em ambientes já existentes (AP)**: utilizar tais modelos em aplicações já existentes, por exemplo: *Web Services*.

A tabela 3.1 apresenta uma comparação entre os modelos de contexto em relação aos requisitos. O símbolo "-" indica que não há atendimento ao requisito. "+" corresponde a um atendimento ao requisito e "++" indica que o modelo atente fortemente ao requisito.

A conclusão é que modelo baseado em ontologias se mostra o mais promissor para a modelagem de contexto em ambientes pervasivos. Modelos baseados em ontologias são fortes quanto ao requisito de composição distribuída. Novos tipos de informação de contexto (conceitos) assim como instâncias (fatos) novas ou atualizadas podem ser manipulados no ambiente de uma forma distribuída. O requisito de validação parcial é possível, e um amplo conjunto de ferramentas de validação estão disponíveis. Modelos orientados a ontologias permitem não apenas validação de tipos de dados, mas também

Tabela 3.1: Comparação entre modelos de contexto

Modelos	CD	VP	QI	IA	NF	AP
Pares atributo-valor	-	-	-	-	-	+
Baseados em esquemas	+	++	-	-	+	++
Gráficos	-	-	+	-	+	+
Orientados a objetos	++	+	+	+	+	+
Baseados em lógica	++	-	-	-	++	-
Baseados em ontologias	++	++	+	+	++	+

validação de conteúdo pela especificação de faixas para informações de contexto. Todos os pontos fortes relativos a normalização e formalização relativos a ontologias são herdados por este modelo. Aplicabilidade para diferentes ambientes da Computação Pervasiva é possível pela capacidade de extensão do modelo ontológico.

3.1.6 Implementação de aplicações sensíveis ao contexto

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 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 sensor, 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 grande parte 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 extensão 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. Neste sentido, a revisão da arquitetura abstrata proposta por (AILISTO et al., 2002) se mostra oportuna como preparatória para avaliação dos trabalhos relacionados (vide seção 3.2. Esta arquitetura é constituída das seguintes camadas: sensores, recuperação de dados, pré-processamento, armazenamento-gerenciamento e aplicação.

Primeira Camada: Sensores

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**: 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, combinando 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.

Segunda Camada: Recuperação de Dados

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 (*Global Positioning System*) sem maiores modificações.

Terceira Camada: Pré-Processamento

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. A performance da rede também 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.

Quarta Camada: Armazenamento-Gerenciamento

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.

Quinta Camada: Aplicação

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).

3.2 Trabalhos relacionados sistematizados

Esta seção discute os trabalhos relacionados ao EXEHDA-ON. Foi adotada a metodologia de segmentar a análise dos mesmos em quatro grandes aspectos: arquitetura, modelagem, processamento e armazenamento de contexto. Entende-se que estes aspectos são centrais para definição de soluções para Computação Sensível ao Contexto.

3.2.1 Aspectos arquiteturais

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

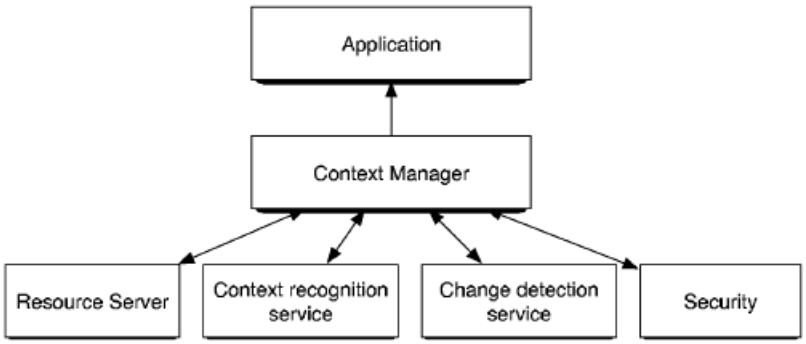

Figura 3.1: Arquitetura do *Context Managing Framework* (KORPIPAA et al., 2003)

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 3.2 apresenta a arquitetura do SOCAM.

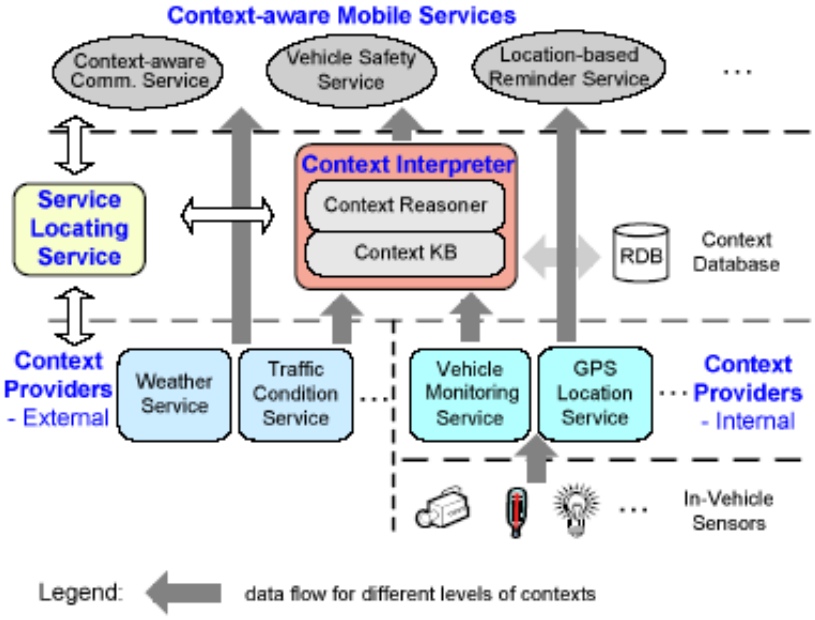

Figura 3.2: Arquitetura do SOCAM
(GU; PUNG; ZHANG, 2004)

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 3.3 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

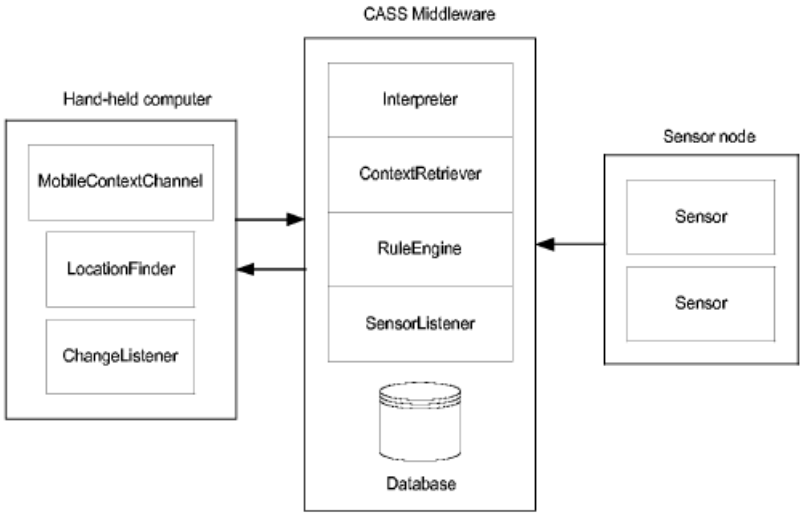

Figura 3.3: Arquitetura do CASS
(FAHY; CLARKE, 2004)

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 contexto inteligente que mantém e gerencia um modelo de contexto compartilhado ao lado de uma comunidade de agentes. A figura 3.4 mostra a arquitetura do CoBrA. 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).

O *Context Toolkit* é um *framework* sensível ao contexto que vai em direção a uma organização *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 3.5 apresenta um diagrama de objetos para as abstrações do *Context Toolkit*

Um *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

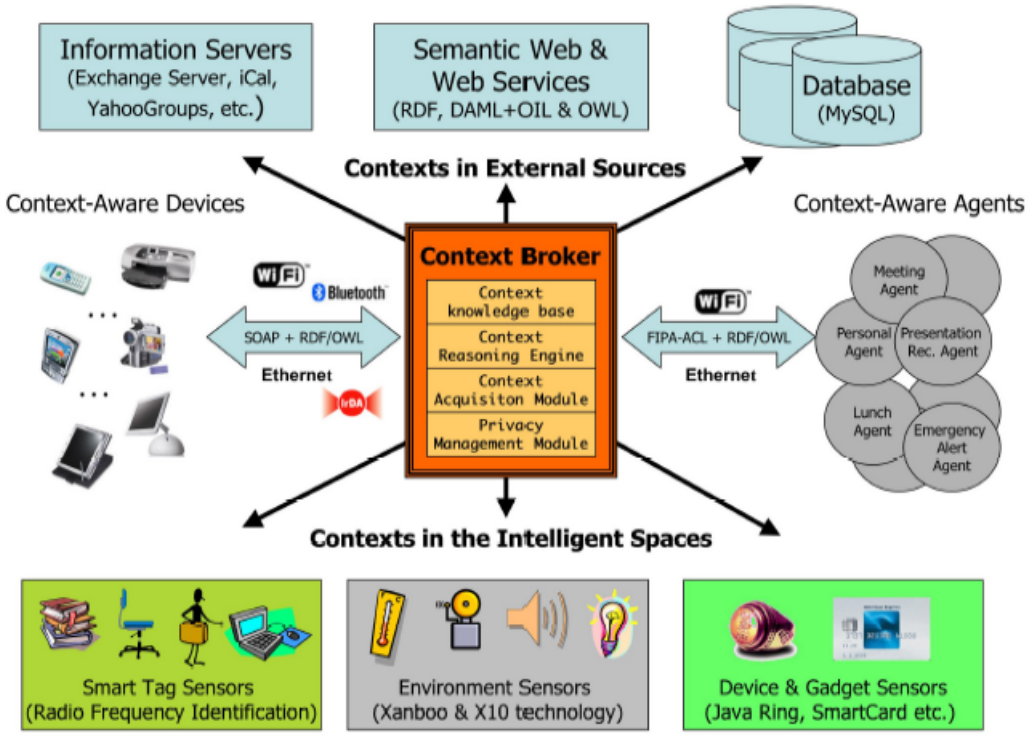

Figura 3.4: Arquitetura do CoBrA
(CHEN, 2004)

figura 3.6 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*.

A figura 3.7 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.

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 Mo-*

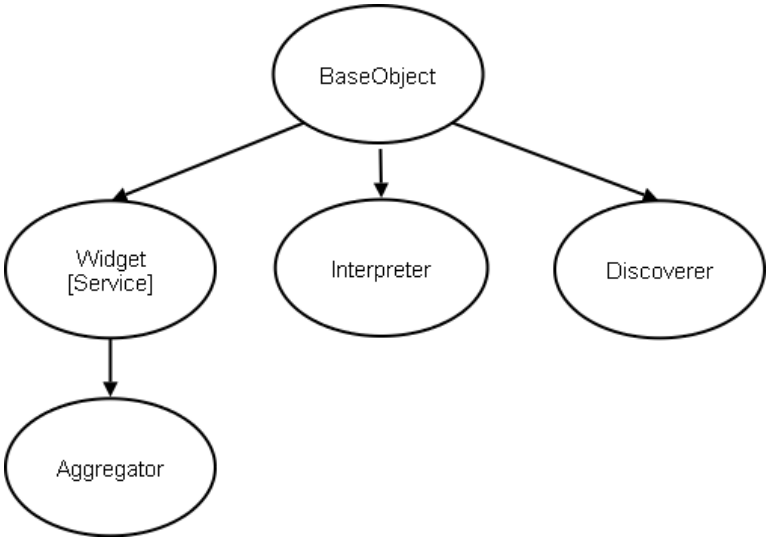

Figura 3.5: Abstrações do *Context Toolkit*
(DEY; SALBER; ABOWD, 2001)

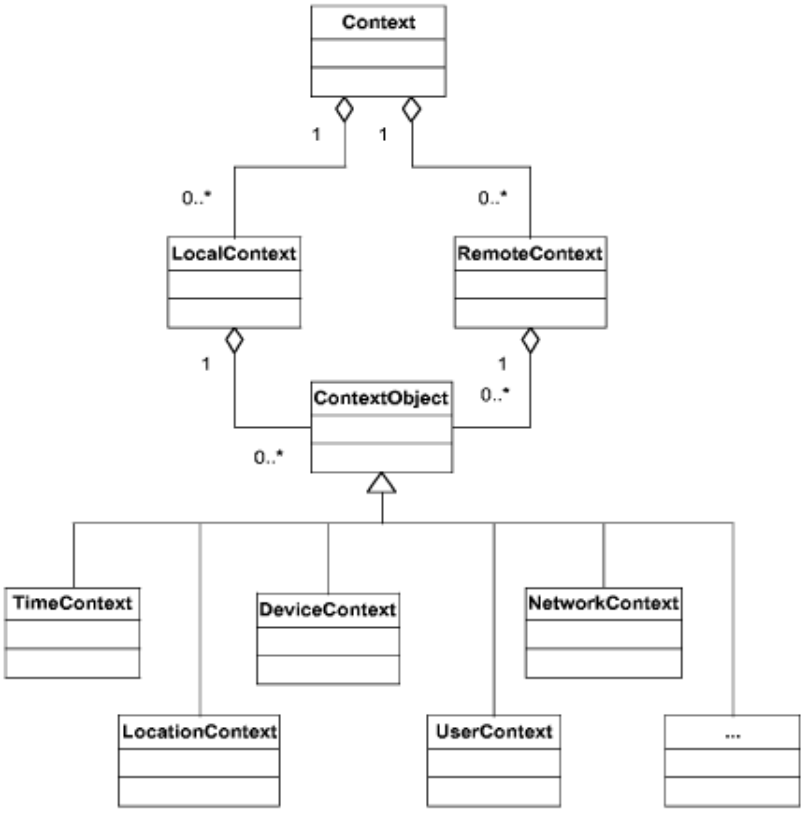

Figura 3.6: Gerenciamento de contextos local e remoto do *Hydrogen*
(HOFER et al., 2002)

bility), um *middleware* de serviços baseado em eventos para sensibilidade à localização, especificamente projetado para ambientes de rede *ad-hoc* sem fio. A figura 3.8 mostra o modelo de objeto sensível do CORTEX.

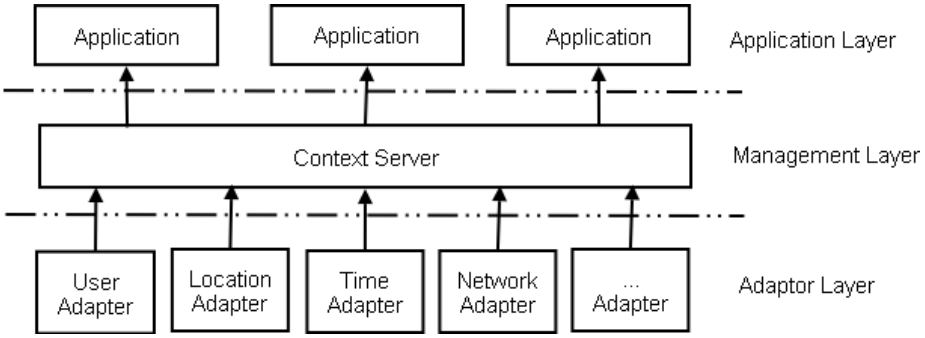

Figura 3.7: Arquitetura do *Hydrogen*
(HOFER et al., 2002)

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 3.8 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.

O projeto Gaia, apresentado de modo mais geral no capítulo 2, é uma infra-estrutura baseada em *middleware* que 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. O sistema é constituído basicamente pelo *kernel* do Gaia e pelo *framework* de aplicações (ROMAN et al., 2002). As partes do sistema Gaia que abrangem a sensibilidade ao contexto são: *Event Manager*, *Context Service* e *Context File System*.

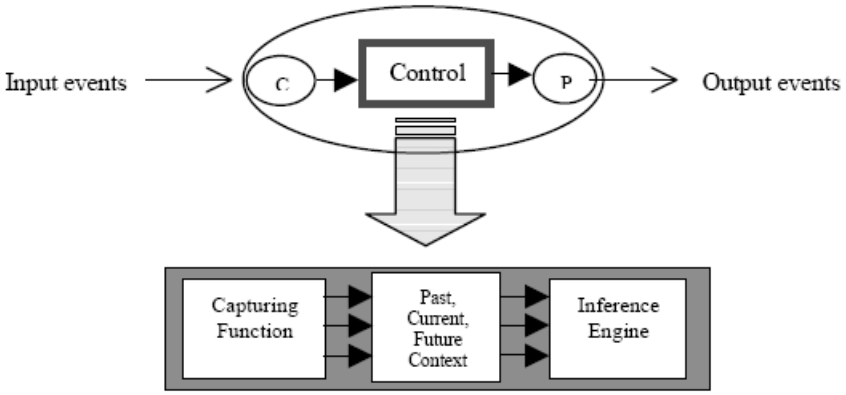

Figura 3.8: Modelo de objeto do CORTEX
(BIEGEL; CAHILL, 2004)

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.

A plataforma Infraware (PEREIRA FILHO et al., 2006) é um *middleware* baseado em *Web Services* com suporte arquitetural para o desenvolvimento, construção e execução de aplicações móveis sensíveis ao contexto. A arquitetura conceitual da Infraware estende, em vários aspectos, a da plataforma WASP (COSTA, 2003), um projeto holandês desenvolvido pela *University of Twente, Telematica Instituut* e *Ericsson*. A plataforma WASP concentra-se na interface aplicação-plataforma, definindo uma linguagem para especificar como ela deve reagir a uma correlação de eventos. A Infraware, por sua vez, foi definida visando o atendimento a vários requisitos funcionais presentes em ambientes sensíveis ao contexto e a integração desses em uma infra-estrutura única, formando uma arquitetura flexível e adequada ao desenvolvimento de aplicações ubíquas reais, em domínios variados, por exemplo, telemedicina. A figura 3.9 mostra a arquitetura geral da plataforma Infraware.

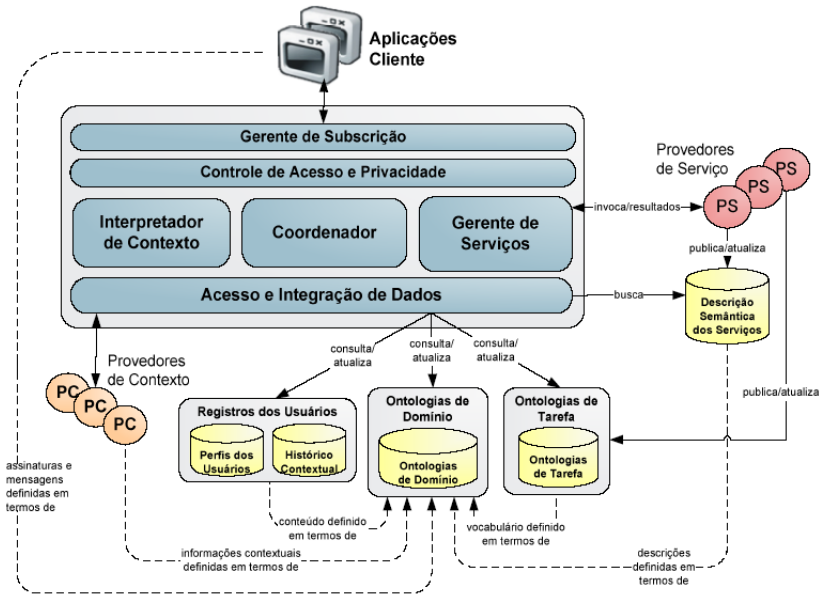

Figura 3.9: Arquitetura da plataforma Infraware
(PEREIRA FILHO et al., 2006)

3.2.2 Aspectos de 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, *Context Managing* e Infraware. 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 para 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). 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.

A Infraware utiliza conceitos e tecnologias da Web Semântica na sua concepção. Ontologias são usadas para especificar modelos formais extensíveis que descrevem não somente o domínio das aplicações, mas também os serviços. Essa abordagem diferenciada da arquitetura provê meios de configurar as interações entre as aplicações e a plataforma em tempo de execução (*run-time*).

3.2.3 Aspectos de 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* (componentes de *interface* gráfica com o usuário), 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 (MCBRIDE, 2007), uma ferramenta para Web semântica.

Na arquitetura do CoBrA existe um *engine* de inferência que processa os dados do contexto. O *engine* contém 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.

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) $\Rightarrow$ Context(Atividade Social, Sala 501, is, Apresentação).

A Infraware introduz componentes para tratamento de questões relacionadas à interpretação semântica de contexto, à aquisição e integração de dados contextuais heterogêneos e distribuídos, à gerência integrada de serviços com descrição semântica, à resolução de conflitos e coordenação entre aplicações. A máquina de inferência do Interpretador de Contexto utiliza a API Jena para manipulação e inferência de informações contextuais. Estas informações são descritas em termos das Ontologias de Domínio, expressas em OWL, e definem uma descrição de conceitos do domínio das aplicações. Os pedidos de subscrição das aplicações são então realizados pela definição de regras de inferência, escritas na linguagem GRL (*Generic Rule Language*), especificada pela

API Jena. Essas regras expressam contextos possíveis e são avaliadas pelo Interpretador de Contexto, que utiliza as informações enviadas pelos provedores de contexto e as informações semânticas capturadas da Ontologia de Domínio para disparar ações específicas na ocorrência de um determinado contexto. Através da linguagem declarativa RDQL (SEABORNE, 2007a), as aplicações podem ainda realizar consultas semânticas com suporte à inferência e validação sintática.

3.2.4 Aspectos de armazenamento 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 informações de contexto, a manutenção de um histórico dos contextos é principalmente uma questão de armazenamento.

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.

3.2.5 Análise das abordagens dos projetos

Na tabela 3.2 são sumarizados os principais aspectos das abordagens apresentadas. A arquitetura de um sistema sensível ao contexto é principalmente orientada pelo método de aquisição do contexto. O principal critério para uma abordagem arquitetural é a separação entre aquisição de contexto e seu uso. Todos os projetos de Computação Sensível ao Contexto apresentados neste capítulo suportam essa separação.

Portanto, surgiram diferentes soluções proprietárias, como as usadas nos projetos apresentados neste capítulo. SOCAM e Infraware usam a abordagem mais sofisticada para sensoriamento de informações de contexto. Informações de sensores virtuais externos são disponibilizadas aos consumidores através de *Web services*. Provedores internos para sensores consultados tem suas informações disponibilizadas através de eventos de contexto baseados ontologias representadas em OWL .

A modelagem e processamento do contexto suportado pelos diferentes *frameworks* é um critério fundamental para prover serviços ou aplicações adaptáveis ao contexto. Neste sentido, ontologias proporcionam um rico formalismo para especificação de informações de contexto. Baseado em cada modelo ontológico, motores de inferência podem derivar novos conceitos para adaptar o comportamento do serviço ou aplicação. A principal desvantagem do *Context Toolkit* é, portanto, seu modelo de contexto, um conjunto de tuplas atributo-valor. O processamento do contexto é limitado devido ao fato que os atributos não possuem um significado. Além disso, o uso de modelos não baseados em ontologias requer um esforço maior de programação e fecha o modelo de contexto para o resto do sistema. Também, a falta de uma semântica declarativa não permite raciocínio e compartilhamento de conhecimento.

O gerenciamento de dados históricos de contexto provê habilidade para imple-

mentar algoritmos de aprendizagem que permitem serviços de sensibilidade ao contexto com grande capacidade de suporte para adaptação. Além disso, baseado em algoritmos de aprendizagem, informações de contexto podem ser previstas para pró-ativamente prover um conjunto de serviços ao usuário. A maioria dos *frameworks* analisados armazenam informações de contexto, mas nenhum deles usa técnicas de aprendizagem para prover serviços de sensibilidade ao contexto pró-ativamente.

Tabela 3.2: Comparação dos projetos em Computação Sensível ao Contexto

Projetos	Arquitetura	Modelagem de Contexto	Processamento de Contexto	Armazenamento de Contexto
CASS	*Middleware* centralizado	Modelo de dados relacional	Motor de inferência e base de conhecimento	Disponível
CoBrA	Baseado em agentes	Ontologias (OWL)	Motor de inferência e base de conhecimento	Disponível
Context Managing Framework	Baseado em *blackboard*	Ontologias (RDF)	Serviço de reconhecimento de contexto	Não disponível
Context Toolkit	Baseado em *widgets*	Tuplas atributo-valor	Agregação e interpretação de contexto	Disponível
CORTEX	Modelo de objeto sensível	Modelo de dados relacional	*Framework* de descoberta de serviços	Disponível
Gaia	MVC com extensões	Predicados com 4 valores (DAML+OIL)	Módulo de serviço de contexto (Lógica de primeira ordem)	Disponível
Hydrogen	Arquitetura em três camadas	Orientado a objetos	Interpretação e agregação de dados brutos	Não disponível
SOCAM	Distribuído com servidor centralizado	Ontologias (OWL)	Motor de inferência	Disponível
Infraware	*Middleware*	Ontologias (OWL)	Motor de inferência	Disponível

3.3 Considerações finais

Um dos aspectos que se destaca neste capítulo é a comparação entre diferentes abordagens para modelagem de contexto em ambientes de Computação Pervasiva. Esta comparação demonstra que o modelo baseado em ontologias é o que contempla de forma mais ampla os requisitos avaliados, corroborando com as decisões deste projeto que

prevêem o uso de ontologias. Também, cabe salientar neste capítulo a apresentação da descrição de uma arquitetura geral para aplicações sensíveis ao contexto. Neste sentido, no presente trabalho entende-se como relevante esta abordagem, bem como considera-se a mesma como fundamentação teórica para a construção da proposta de arquitetura de software do EXEHDA-ON.

A análise dos projetos destacados neste capítulo, baseada na segmentação do estudo nas quatro grandes visões apresentadas, caracterizou diversos desafios de pesquisa que foram considerados na proposição do EXEHDA-ON, a qual será apresentada no capítulo 5. Especialmente, os aspectos relacionados a modelagem e processamento de contexto reforçam a idéia de uso de ontologias como base para o mecanismo de sensibilidade ao contexto proposto para o EXEHDA-ON.

No próximo capítulo serão apresentados os princípios gerais do EXEHDA-ON, destacando os conceitos e tecnologias relativos a ontologias, bem como as características e funcionalidades do *middleware* EXEHDA, com intuito que consolidar um referencial teórico para a proposta deste trabalho.

4 TECNOLOGIAS E MÉTODOS EMPREGADOS

Este capítulo aborda ontologias enquanto metodologia, destacando (*i*) seu uso na área de Ciência da Computação, (*ii*) as motivações para sua utilização, (*iii*) seu projeto e construção, (*iv*) seus tipos e componentes e (*v*) as linguagens e ferramentas utilizadas em seu desenvolvimento.

Também, são apresentadas as principais características e funcionalidades do *middleware* EXEHDA enquanto tecnologia de base para o EXEHDA-ON, com destaque para o subsistema de reconhecimento de contexto e adaptação, o qual inclui o mecanismo de sensibilidade ao contexto objeto de estudo deste trabalho.

Considerando a perspectiva do EXEHDA-ON, nesta dissertação, o paradigma de ontologia estará focado no domínio dos ambientes pervasivos, especificamente o ambiente pervasivo provido pelo EXEHDA.

4.1 Ontologias na Ciência da Computação

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 (FENSEL, 2000).

Inicialmente, é preciso estabelecer uma distinção entre o significado da palavra "Ontologia" (escrita com letra inicial maiúscula) e "ontologia" (escrita com letra inicial 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:

- "conceituação" se refere ao modelo abstrato do mundo real;
- "explícita" significa que os conceitos e seus requisitos são definidos explicitamente;
- "formal" indica que a ontologia é processável por máquina, permite raciocínio automático e possui semântica lógica formal;
- "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, 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.

4.1.1 Motivações para o uso de ontologias em sistemas distribuídos

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 nos sistemas distribuídos modernos.

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 motivo para utilização de ontologias, pois a representação formal adquirida com a aplicação das mesmas pode tornar possível a automação da verificação de consistência, resultando em ambientes mais confiáveis (GRUNINGER, 1996).

Ainda, pode-se destacar vantagens da utilização de ontologias na interoperabilidade de sistemas computacionais:

- 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.

4.1.2 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, as proposições feitas por autores como (BORST, 1997) e (STUDER; BENJAMINS; FENSEL, 1998), ainda se mantêm, e apontam para a 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. As ontologias de domínio e as 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.

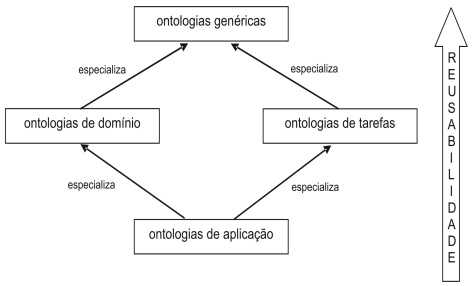

Figura 4.1: Tipos de ontologias
(GUARINO, 1998)

4.1.3 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: (*i*) identificar os principais conceitos e relacionamentos do domínio de interesse, (*ii*) definir um texto preciso a respeito destes conceitos e relacionamentos encontrados e (*iii*) 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.

4.1.3.1 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). Dentre estes destacaríamos:

- 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.

4.1.3.2 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.

4.2 Linguagens identificadas para construção de ontologias

As linguagens utilizadas para construção de ontologias são, em geral, classificadas em duas categorias (SU; ILEBREKKE, 2002): as linguagens de ontologia tradicionais e as linguagens de ontologia para Web. Essa classificação, apresentada a seguir, é baseada em uma avaliação sobre linguagens ontológicas realizada anteriormente por (CORCHO; GÓMEZ-PÉRES, 2000):

- Linguagens de ontologia tradicionais: Loom, Ontolingua, OCML, FLogic;
- Linguagens de ontologia para *Web*: SHOE, XOL, OIL, DAML+OIL, OWL.

Linguagens de ontologia tradicionais

Esse grupo de linguagens tem origem na Inteligência Artificial ou na Engenharia de Conhecimento. Essa categoria possui várias linguagens: (*i*) linguagens que já eram usadas para representar o conhecimento; (*ii*) linguagens que foram adaptadas de linguagens de representação de conhecimento e (*iii*) linguagens especificamente criadas para a representação de ontologias.

A linguagem Loom (LOOM, 2006) é baseada em lógica de descrições e caracteriza-se pela possibilidade de definir conceitos em termos de descrições que especificam as propriedades que os objetos devem satisfazer para pertencer ao conceito.

As linguagens Ontolingua (GRUBER, 1992), OCML (*Operational Conceptual Modelling Language*) (DOMINGUE; MOTTA; GARCIA, 1992) e FLogic (*Frame Logic*) (KIFER; LAUSEN; WU, 1995) são baseadas em classes.

Linguagens de ontologia para *Web*

Essas linguagens são baseadas em padrões *Web* e constituem-se em uma das camadas da arquitetura da Web Semântica (vide figura 4.2).

A *Web* Semântica 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).

Na seção a seguir serão apresentadas de forma mais detalhada as linguagens de especificação de ontologia baseadas na Web, bem como os padrões que elas seguem. A maior ênfase nesta categoria, se justifica pelo fato das ontologias empregadas nesse trabalho terem sido construídas na linguagem OWL (BECHHOFER et al., 2004).

4.2.1 Linguagens de ontologia no contexto da *Web* Semântica

A *Web* Semântica tem como objetivo fundamental associar a semântica correspondente às informações disponíveis na *Web* atual. Assim, documentos *Web* com conteúdo estruturado e com significado permitem um uso mais efetivo da informação para descoberta, automação, integração e reuso pelas várias aplicações. Para alcançar este objetivo há necessidade de estabelecer padrões de troca de informações que sejam interpretáveis por máquinas. Estes padrões não somente definem a sintaxe da informação, mas também o seu significado.

Desta forma, a *Web* Semântica define uma nova arquitetura para a *Web*, a qual é mostrada na figura 4.2 e detalhada nas seções seguintes. Linguagens e padrões são sugeridos para cada camada desta arquitetura, por exemplo, as linguagens de ontologia voltadas para a *Web* estão associadas à "Camada de Descrição Semântica e Lógica".

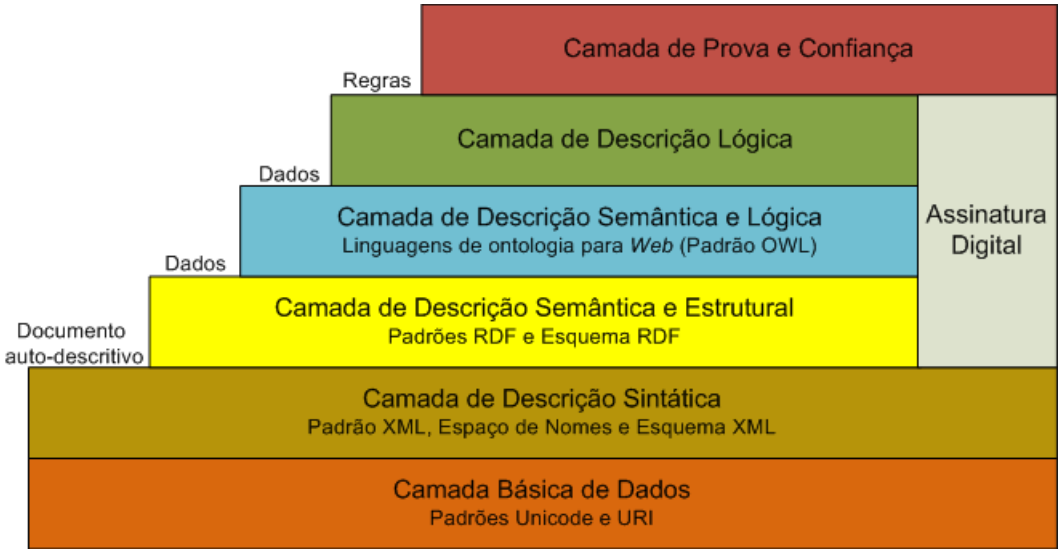

Figura 4.2: Arquitetura da *Web* Semântica
Adaptado de (BERNERS-LEE; HENDLER; LASSILA, 2001)

4.2.1.1 Camada Básica de Dados

A primeira camada garante o uso padronizado do mesmo conjuntos de caracteres (*Unicode*) e uma forma unívoca para a identificação e localização de recursos através do

URI (*Uniform Resource Identifier*).

4.2.1.2 Camada de Descrição Sintática

Essa camada é constituída pela tecnologia XML (*eXtensible Markup Language*), que é uma linguagem de marcação de propósito geral que permite criar outras linguagens de marcação de propósito específico, ou seja, é uma meta-linguagem de editoração. XML é adotada pelo W3C como linguagem padrão para troca de informações na Web. Seu principal objetivo é facilitar o compartilhamento de dados através de diferentes sistemas, principalmente sistemas conectados via Internet. A idéia é que essa camada descreva a estrutura do documento, deixando para as camadas superiores a definição do seu conteúdo.

4.2.1.3 Camada de Descrição Semântica e Estrutural

Modelos de dados comuns e padrões de troca de dados são desejáveis para garantir uma rápida integração de diferentes fontes de dados e para ligar diferentes representações semânticas. O W3C propôs o RDF (*Resource Description Framework*) (KLYNE; CARROLL, 2004) como um modelo de dados adequado para representar os recursos e suas informações na *Web*, permitindo a integração dos mesmos. RDF é uma linguagem declarativa que é usada para expressar as proposições usando vocabulários formais, particularmente aqueles especificados na linguagem RDF *Schema* (BRICKLEY; GUHA, 2004). Além disso, o RDF fornece a base para linguagens declarativas mais avançadas com um propósito similar.

Uma definição em RDF é vista como um conjunto de triplas que forma um grafo RDF. Cada tripla é constituída por: um sujeito, um predicado (também conhecido como propriedade) e um objeto (MANOLA; MILLER, 2004). Portanto, os nodos de um grafo RDF, representam os sujeitos e objetos das triplas que o formam. A direção do arco que liga um sujeito e um objeto possui significado e está sempre apontado para o objeto. A declaração de uma tripla RDF da forma <sujeito,predicado,objeto> informa que existe algum relacionamento, indicado pelo predicado, entre os recursos denotados pelo sujeito e o objeto da tripla. Como um grafo RDF é um conjunto de triplas, a sua declaração representa a agregação de todas as declarações das triplas contidas nele. Desta forma, o significado de um grafo RDF é a conjunção lógica dos enunciados correspondentes a todas as triplas que ele contém.

A linguagem RDF não possui mecanismos para descrever relacionamentos entre nodos de diferentes triplas, porém o RDF *Schema* possui. RDF *Schema* é uma linguagem de descrição do vocabulário RDF, que estende a linguagem RDF semanticamente, provendo um conjunto fixo de primitivas básicas de modelagem para a definição de metadados (classe, propriedade, e os relacionamentos *is-a* e *element-of*) e uma maneira padrão de codificá-los em XML. RDF *Schema* serve de base para as linguagens de especificação de ontologias (CORCHO; GÓMEZ-PÉRES, 2000).

4.2.1.4 Camada de Descrição Semântica e Lógica

As linguagens de ontologia para a *Web* estão presentes nesta camada. Essas 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* (CORCHO; FERNÁNDEZ-LÓPES; GÓMEZ-PÉRES, 2001).

SHOE

A linguagem SHOE (*Simple HTML Ontology Extensions*) (LUKE; JEFF, 2000) 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, permitindo 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

XOL (*Ontology Exchange Language*) (KARP; CHAUDHRI; THOMERE, 1999) 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.

OIL

A linguagem OIL (*Ontology Inference Layer*) (FENSEL; HORROCKS; VAN HARMELEN, 2000) 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: (*i*) semânticas formais e suporte eficiente ao raciocínio, providos pelas lógicas de descrição; (*ii*) primitivas epistemológicas de modelagem ricas providas pelos sistemas baseados em *frames*; e (*iii*) uma proposta padronizada para troca de notações sintáticas, como provido pela *Web*.

DAML+OIL

DAML+OIL (*DARPA Markup Language* + *Ontology Inference Layer*) (HORROCKS; PATEL-SCHNEIDER; VAN HARMELEN, 2002) 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

OWL (*Web Ontology Language*) (BECHHOFER et al., 2004) 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*: 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;

- 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, porém sem qualquer garantia computacional.

4.2.1.5 Camada de Descrição Lógica

Essa camada permite a especificação de regras que atuam sobre recursos da *Web*. Regras são necessárias quando não é possível, apenas por meio de construtores da linguagem OWL, expressar mapeamentos sobre conceitos de uma ontologia, como relações parte-todo. Neste sentido, destacam-se duas linguagens para descrição de regras: RuleML (*Rule Markup Language*) (HIRTLE et al., 2005) e SWRL (*Semantic Web Rule Language*) (HORROCKS et al., 2004).

Baseada em XML, RuleML utiliza um subconjunto da linguagem Prolog (*Programming Logic*) para representação de fatos e regras. SWRL é uma linguagem para especificação de fatos e regras baseada na combinação dos dialetos OWL *Lite* e OWL DL com a linguagem RuleML. Assim, SWRL utiliza todos os padrões das camadas inferiores da arquitetura da *Web* Semântica.

4.2.1.6 Camada de Prova e Confiança

A Camada de Prova e Confiança executa as regras definidas na Camada de Descrição Lógica e também avalia a correção e a confiabilidade da execução dessas regras. A confiabilidade dos fatos envolvidos em regras depende do contexto de execução das aplicações e de assinaturas digitais.

Para que esta camada se desenvolva, as camadas inferiores devem estar sedimentadas, o que ainda não ocorreu. Mecanismos de prova e confiança merecem profunda investigação pela comunidade de pesquisa da *Web* Semântica (HORROCKS et al., 2005).

4.3 Resumo das ferramentas para desenvolvimento de ontologias

Na perspectiva de desenvolver uma ontologia para emprego no EXEHDA-ON, foram estudadas ferramentas, tais como:

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 ontologia 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. Aceita as linguagens OCML, Ontolingua, entre outras (APOLLO, 2003).

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: (*i*) gerenciamento de ontologias utilizando uma interface gráfica; (*ii*) suporte para modelagem de tarefas; (*iii*) verificação de elementos, considerando herança de propriedades e checagem de consistência; (*iv*) 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).

4.4 Ontologias na perspectiva do EXEHDA-ON

Nesta seção serão discutidos os fundamentos referentes ao uso de ontologias na busca da qualificação do mecanismo de sensibilidade ao contexto do *middleware* EXEHDA. As premissas contempladas nestes fundamentos são: ambiente pervasivo com composição dinâmica, aplicações distribuídas, móveis e sensíveis ao contexto e construção de ontologias para representação e processamento do contexto.

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. Os componentes do sistema distribuído podem ser dispositivos, aplicações, serviços, bases de dados e usuários. 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).

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.

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.

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 utilizados precisam estar muito bem definidos, 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.

4.4.1 Metodologia para construção do modelo ontológico do EXEHDA-ON

Devido a falta de uma metodologia estabelecida para o desenvolvimento de ontologias, os desenvolvedores costumam usar seus próprios critérios no processo de desenvolvimento. Uma prática comum entre os desenvolvedores de ontologias é passar diretamente do passo de aquisição de conhecimento para o passo de implementação, o que gera os seguintes problemas:

- os modelos conceituais da ontologia ficam implícitos no código da implementação;
- dificuldades de reuso da ontologia, pois o *design* da ontologia e as decisões de projeto estão implícitos no código;
- gera problemas de comunicação devido às dificuldades que o *expert* no domínio da ontologia tem para entender o código da implementação. Isso é um sério problema, pois ele tende a ser a principal fonte de informação sobre o domínio;
- gera dificuldades no desenvolvimento de ontologias complexas, pois a passagem da aquisição de conhecimento para a implementação é muito abrupta;
- dependendo da linguagem escolhida para a codificação pode-se limitar a capacidade de descrição conceitual do domínio da ontologia.

Desta forma, se faz necessário adotar de uma metodologia para que se reduza as dificuldades acima citadas. Nesta seção são analisadas três metodologias: (USCHOLD; KING, 1995), (GRUNINGER; FOX, 1995) e (FERNÁNDEZ; GÓMES-PÉREZ; JURISTO, 1997).

A metodologia proposta por Mike Uschold e Martin King (USCHOLD; KING, 1995) compreende os seguintes estágios para o desenvolvimento de ontologias:

- identificação do propósito: visa identificar o porquê da construção da ontologia e as suas intenções de uso;
- construção da ontologia - subdividida em três estágios:

  - captura da ontologia: identificar conceitos e relacionamentos do domínio de interesse para produzir uma definição precisa dos mesmos;
  - codificação: codificar a ontologia em uma linguagem formal;
  - integração com ontologias existentes: integrar a nova ontologia com as ontologias existentes.

- avaliação da ontologia;
- documentação da ontologia.

A principal desvantagem dessa metodologia é que ela não descreve de uma forma precisa as técnicas para execução das diferentes atividades. O nível de detalhamento da metodologia é muito pequeno, só oferecendo princípios gerais muito vagos.

A metodologia proposta por Michael Grüninger e Mark S. Fox (GRUNINGER; FOX, 1995) foi desenvolvida baseada na experiência dos autores no desenvolvimento de ontologias para empresas. Ela é formada pelos seguintes estágios:

- definir os cenários motivadores: identifica possíveis problemas que demandem uma nova ontologia. O cenário motivador também fornece intuitivamente um conjunto de soluções possíveis para o problema;
- definir informalmente questões de competência: dado o cenário motivador, um conjunto de perguntas irão surgir que necessitarão de uma ontologia para que elas sejam respondidas. Essas perguntas são as questões de competência da ontologia. Elas não são expressas em linguagem formal;
- especificar em lógica de primeira ordem a terminologia: uma vez que foram definidas informalmente as questões de competência a fim de propor ou estender uma ontologia, a terminologia da ontologia deve então ser especificada usando lógica de primeira ordem ou equivalente;
- especificar as questões de competência formalmente: uma vez que foram definidas informalmente as questões de competência e a terminologia da ontologia, as questões de competência são definidas em linguagem formal;
- especificar em lógica de primeira ordem os axiomas: os axiomas na ontologia especificam definições de termos da ontologia e limitações de sua interpretação. Esses axiomas são definidos em lógica de primeira ordem usando os predicados da ontologia;
- verificação através dos teoremas de completude: através desses teoremas são fornecidos meios de determinar a extensibilidade da ontologia, fazendo explicitamente o papel que cada axioma executa no teorema.

Diferentemente da metodologia anterior, essa última fornece mais do que princípios gerais. Observa-se que a partir da segunda etapa, a metodologia exige uma codificação em linguagem formal. Isso significa que após a aquisição do conhecimento sobre o domínio do problema que vai até a segunda etapa, a metodologia parte para a geração de artefatos em lógica de primeira ordem, ou equivalente.

Apesar desse formalismo ser adequado para avaliar se a ontologia atende os requisitos e evitar ambigüidades na especificação, ele dificulta muito a comunicação entre o desenvolvedor da ontologia e o *expert* do domínio que provavelmente é um componente muito importante no desenvolvimento de ontologias. Sendo assim, a desvantagem dessa metodologia está na falta de modelos intermediários que facilitem a comunicação entre desenvolvedor da ontologia e o *expert* do domínio.

A metodologia proposta por Mariano Fernández, Asunción Gómes-Pérez e Natalia Juristo (METHONTOLOGY) (FERNÁNDEZ; GÓMES-PÉREZ; JURISTO, 1997) é a que parece ter a proposta mais madura, pois além de descrever de forma aprofundada a metodologia no que tange os passos a serem seguidos e dos artefatos a serem criados para a geração de um modelo conceitual, fornece também um processo de desenvolvimento de ontologias e também propõe um ciclo de vida baseado em evolução de protótipos.

O processo de desenvolvimento de ontologias proposto identifica que as seguintes atividades devem ser cumpridas quando da construção de uma ontologia:

- **planejamento**: identifica as tarefas principais da ontologia, e planeja a utilização dos recursos;
- **especificação**: define por que a ontologia está sendo construída, e quem serão seus usuários;
- **aquisição de conhecimento**: adquire conhecimento sobre o domínio da ontologia. Pode ser feito: através de entrevistas com *expert* no domínio da ontologia a ser criada, análise de livros sobre o domínio, etc.
- **conceitualização**: criação de um modelo conceitual que descreve o problema e sua solução;
- **formalização**: transforma o modelo conceitual em um modelo formal ou semiformal;
- **integração**: procura integrar o máximo possível as ontologias existentes à nova ontologia;
- **implementação**: implementa a ontologia em uma linguagem formal de modo que ela seja computável;
- **avaliação da ontologia**;
- **aocumentação**: faz a documentação da ontologia visando facilitar futuro reuso e manutenção;
- **manutenção**: executar a manutenção da ontologia quando necessária.

As atividades de aquisição de conhecimento, avaliação e documentação, são executadas em todos os estágios do ciclo de vida. O planejamento é a primeira atividade a ser executada. A maior parte da atividade de aquisição é feita simultaneamente com o estágio de especificação da ontologia e vai decaindo conforme o ciclo de vida avança. A maior parte da atividade de avaliação da ontologia é feita durante os estágios iniciais do ciclo, de forma a diminuir a propagação de erro. A atividade de documentação deve ser realizada em todos os estágios.

A parte que mais relevante desta metodologia é onde são descritos os artefatos que devem ser gerados no seu desenvolvimento, principalmente aqueles gerados na parte de conceitualização da ontologia. Esses artefatos não são codificados em lógica formal e sim em uma representação intermediária que descreve a ontologia. Assim sendo, esses artefatos são de fácil entendimento tanto para o desenvolvedor da ontologia quanto para o *expert* no domínio. Outra grande vantagem da geração desses artefatos durante a conceitualização é que eles fornecem uma boa documentação sobre a ontologia.

O artefato proposto na metodologia que é gerado durante o estágio de especificação é um documento que cobre o objetivo principal, o propósito, a granularidade e o escopo da ontologia. O objetivo é de identificar o conjunto de termos a serem representados, suas características, e sua granularidade. Essa especificação deve ser o mais completa e concisa possível.

Os principais artefatos propostos na metodologia são gerados durante o estágio de especificação e conceitualização:

- **glossário de termos:** inclui todos os termos do domínio (conceitos,instâncias, atributos, verbos, etc.) e suas descrições. A descrição deve ser clara e concisa;

- **árvore de classificação de conceitos:** define relações tais como: subclasse e exclusividade mútua entre classes. Dessa forma são definidas várias taxinomias do domínio, sendo que cada uma gera uma ontologia. Deve-se verificar a consistência (não devem existir ciclos na árvore) e a concisão (não deve existir repetição de conceitos);

- **diagrama de relações binárias:** esse diagrama estabelece os relacionamentos entre conceitos da mesma ontologia e de ontologias diferentes;

- **dicionário de conceitos:** esse documento contém todos os conceitos do domínio, instâncias desses conceitos, atributos de classes e atributos de instâncias dos conceitos, e opcionalmente, sinônimos e antônimos dos conceitos. Para cada árvore de classificação de conceitos deve existir um dicionário de conceitos;

- **tabela de relações binárias:** especifica o nome da relação, o nome dos conceitos origem e destino da relação, a relação inversa e a cardinalidade da relação. Para cada árvore de classificação de conceitos deve existir uma tabela de relações binárias;

- **tabela de atributos de instância:** descreve cada atributo de instância do dicionário de conceitos. Atributos de instância são os atributos definidos nos conceitos, mas que são valorados nas instâncias. Para cada atributo de instância deve ser especificado: nome, tipo de valor, unidade de medida para valores numéricos, precisão dos valores numéricos, faixa de valores aceitos, valor padrão, cardinalidade máxima e mínima, atributos ou constantes utilizadas para inferir o valor do atributo que está sendo definido, atributos que podem ser inferidos desse atributo, fórmulas ou regras para inferir o atributo que está sendo definido e por fim as referências usadas para preencher o atributo;

- **tabela de atributos de classe:** descreve os atributos de classe no dicionário de conceitos. Esses atributos são aqueles em que o valor é dado para o conceito, ou

seja, o valor será o mesmo para todas as instâncias do conceito. Para cada atributo de classe deve ser especificado: nome do atributo, tipo de valor, unidade de medida para valores numéricos, precisão dos valores numéricos, cardinalidade máxima e mínima, atributos que podem ser inferidos desse atributo, e referências;

- **tabela de axiomas:** define conceitos por meio de expressões lógicas que são sempre verdadeiras. Para cada axioma deve ser definido: nome, descrição em linguagem natural, os conceitos aos quais o axioma se refere, os atributos usados no axioma, a expressão lógica que define o axioma formalmente e referências;

- **tabela de constantes:** especifica para cada constante: nome, descrição em linguagem natural, o tipo lógico do valor (o que o valor é), seu valor constante, sua unidade de medida, os atributos que podem ser inferidos usando a constante e referências;

- **tabela de fórmulas:** descreve cada fórmula das tabelas de atributos de instância. Essa tabela é usada para inferir os valores numéricos dos atributos de instância. Para cada fórmula deve ser especificado: nome, atributos inferidos com a fórmula, expressão matemática,descrição em linguagem natural, atributos e constantes usados no cálculo da fórmula, precisão do cálculo, situações em que a fórmula faz sentido e referências;

- **árvore de classificação de atributos:** mostra graficamente os atributos e as constantes relacionados a uma seqüência de cálculo de um atributo raiz. É usado para validar que todos os atributos usados na fórmula fazem sentido e nenhum atributo foi omitido;

- **tabela de instâncias:** descreve as instâncias do domínio. É descrito através do nome, seus atributos de instância e valores para os mesmos.

4.4.2 Lógica de descrições prevista para uso no EXEHDA-ON

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: (*i*) 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; e (*ii*) 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 (GRADEL, 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 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 indivíduos 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 Di menos a de Ai;

- 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.

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.

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).

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.

4.5 *Middleware* EXEHDA: revisão arquitetural e funcional

O EXEHDA (YAMIN, 2004) é um *middleware* que integra o projeto ISAM (Infra-estrutura de Suporte às Aplicações Móveis Distribuídas) (ISAM, 2007), sendo direcionado às aplicações distribuídas, móveis e sensíveis ao contexto da Computação Pervasiva.

Abaixo são apresentadas premissas perseguidas na concepção do EXEHDA (YAMIN, 2004), e consideradas na modelagem do EXEHDA-ON:

Dinamicidade e heterogeneidade do ambiente de processamento

O deslocamento do usuário (mobilidade física), aspecto característico da Computação Pervasiva, determina a execução de aplicações a partir de diferentes equipamentos e/ou pontos da rede global, nos quais a oferta e/ou disponibilidade dos recursos computacionais é variável. Disto decorre:

- elevada flutuação na banda passante disponível para as comunicações;
- equipamentos de usuários com acentuadas diferenças nos atributos de hardware e sistema operacional;
- diferentes infra-estruturas para conexão à rede global.

Não comprometimento com uma classe específica de aplicações adaptativas

Mobilidade implica adaptabilidade, o que significa que sistemas devem ter consciência da localização e do contexto, e devem tirar vantagem desta informação, estruturando-se de modo distribuído e reconfigurando-se dinamicamente.

Controle da adaptação

A gerência da adaptação pode ocorrer no limite de dois extremos: (*i*) no primeiro, denominado *laissez-faire*, a aplicação é responsável por toda a adaptação que será realizada; por sua vez, no segundo extremo, (*ii*) denominado *application-transparent*, o sistema é encarregado de gerenciar toda a adaptação que vier a ocorrer. Nenhuma dessas estratégias pode ser considerada a melhor. Uma estratégia diferente pode ser requerida para cada circunstância e/ou aplicação. Há situações, por exemplo, onde o código fonte da aplicação não está disponível e a estratégia a ser utilizada deve ser a *application-transparent*. Em outros casos, pode ser mais oportuno incluir apenas na aplicação os mecanismos adaptativos, sem envolver o ambiente de execução. Logo, a proposta para o EXEHDA é modelar um *middleware* que faculte uma estratégia colaborativa com a aplicação nos procedimentos de adaptação. Deste modo, considerando a natureza da aplicação, o programador poderá definir a distribuição de responsabilidades entre o *middleware* e a aplicação no processo de adaptação.

Suporte às mobilidades lógica e física

A Computação Móvel genericamente se refere a um cenário onde todos, ou alguns nodos que tomam parte no processamento, são móveis. Desta definição podem derivar diferentes interpretações. Em um extremo, a mobilidade leva em conta as necessidades dos

usuários nômades, isto é, usuários cuja conexão na rede ocorre de posições arbitrárias e que não ficam permanentemente conectados. Em outro extremo, estão os usuários móveis, os quais retêm a conectividade durante o deslocamento, tipicamente explorando conexões sem fio. Desta forma, a Computação Móvel é caracterizada por três propriedades: mobilidade, portabilidade e conectividade.

Na proposta do EXEHDA estas propriedades desdobram duas preocupações de pesquisa: (*i*) os segmentos sem fio da rede global levantam novas condições operacionais, entre as quais se destaca a comunicação intermitente. A ocorrência de desconexões de nodos no ambiente móvel, sejam estas voluntárias ou não, é mais uma regra do que uma exceção; (*ii*) a natureza dinâmica do deslocamento do hardware e do software na rede global introduz questões relativas tanto à identificação física dos nodos quanto à localização dos componentes de software que migram.

Estas questões apontam para a necessidade de mecanismos dinâmicos que realizem o mapeamento dos componentes móveis, de modo a viabilizar sua localização e permitir a interação com os mesmos. Portanto, o *middleware* para suporte à Computação Pervasiva deve levar em conta essas limitações, de modo que as aplicações não percam sua consistência quando um componente de software migrar entre nodos da rede global, ou quando um nodo temporariamente não estiver disponível por estar desligado ou sem conexão, ou ainda trocar sua célula de execução em função do deslocamento.

Enfim, o *middleware* deverá viabilizar a semântica siga-me das aplicações pervasivas. 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, o ambiente de execução deve fornecer informações de contexto que extrapolam a localização onde o componente móvel da aplicação se encontra.

Suporte à semântica siga-me

Oferecer suporte à semântica siga-me é uma das contribuições centrais do EXEHDA ao Projeto ISAM. No EXEHDA, este suporte é construído pela agregação de funcionalidades relativas ao reconhecimento de contexto, ao acesso pervasivo e à comunicação. Como estratégia para tratamento da complexidade associada ao suporte da semântica siga-me, no EXEHDA é adotada a decomposição das funcionalidades de mais alto nível, recursivamente, em funcionalidades mais básicas. Nesta perspectiva, no EXEHDA o reconhecimento de contexto está relacionado com dois mecanismos: (*i*) um de monitoração que permite inferir sobre o estado atual dos recursos e das aplicações, e (*ii*) outro que pode promover adaptações funcionais e não funcionais, tendo em vista o contexto monitorado (YAMIN, 2004).

A adaptação não-funcional consiste na capacidade do sistema atuar sobre a localização física dos componentes das aplicações, seja no momento de uma instanciação do componente, seja, posteriormente, via migração do mesmo. Ambas operações demandam a existência de mecanismo para instalação sob demanda do código, assim como mecanismos para descoberta e alocação dinâmicas de recursos e acompanhamento de seu estado. Por sua vez, a adaptação funcional consiste na capacidade do sistema atuar sobre a seleção da implementação do componente a ser utilizado em um determinado contexto de execução. Novamente surge a necessidade do suporte à instalação de código sob demanda. A funcionalidade da instalação sob demanda implica que o código a ser instalado

esteja disponível em todos os dispositivos nos quais este venha a ser necessário. Considerando as dimensões do ambiente pervasivo, é impraticável manter a cópia de todos os possíveis códigos em todos os eventuais dispositivos. Procede daí a necessidade de um mecanismo que disponibilize acesso pervasivo ao repositório de código, mecanismo este, que deve considerar fortemente o aspecto escalabilidade. O aspecto de mobilidade, tanto dos componentes das aplicações quanto do usuário, inerente à semântica *siga-me*, faz propícia uma estratégia de comunicação caracterizada pelo desacoplamento espacial e temporal.

4.5.1 Organização do EXEHDA

O *middleware* EXEHDA tem por finalidade definir a arquitetura para um ambiente de execução destinado às aplicações da Computação Pervasiva, no qual as condições de contexto são pró-ativamente monitoradas, e o suporte à execução deve permitir que tanto a aplicação como ele próprio utilizem estas informações na gerência da adaptação de seus aspectos funcionais e não-funcionais. Entende-se por adaptação funcional aquela que implica a modificação do código sendo executado. Por sua vez, adaptação não-funcional, é aquela que atua sobre a gerência da execução distribuída. Também a premissa *siga-me* das aplicações *pervasivas* deverá ser suportada, garantindo a execução da aplicação do usuário em qualquer tempo, lugar e equipamento (YAMIN et al., 2005).

As aplicações-alvo são distribuídas, adaptativas ao contexto em que executam e compreendem a mobilidade lógica e a física. Na perspectiva do EXEHDA, entende-se por mobilidade lógica a movimentação entre equipamentos de artefatos de software e seu contexto, e por mobilidade física o deslocamento do usuário, portando ou não seu equipamento (ISAM, 2007).

4.5.1.1 Arquitetura de software

A figura 4.3 destaca o Subsistema de Reconhecimento de Contexto e Adaptação do EXEHDA. Este subsistema é responsável pelo suporte à adaptação, incluindo 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.

Os principais requisitos que o EXEHDA deve atender são: (i) gerenciar tanto aspectos não-funcionais como funcionais da aplicação, e de modo independente, (ii) dar suporte à adaptação dinâmica de aplicações; (iii) disponibilizar mecanismos para obter e tratar informações de contexto; (iv) utilizar informações de contexto na tomada de decisões, (iv) decidir as ações adaptativas de forma colaborativa com a aplicação e (v) disponibilizar a semântica siga-me, possibilitando ao usuário o disparo de aplicações e o acesso a dados a partir de qualquer lugar, e a execução contínua da aplicação em face ao seu deslocamento.

4.5.1.2 Ambiente pervasivo

O ISAMpe (ISAM *pervasive environment*) corresponde ao ambiente computacional onde recursos e serviços são gerenciados pelo EXEHDA com o intuito de atender os requisitos da Computação Pervasiva. A composição deste ambiente envolve tanto os dispositivos dos usuários, como os equipamentos da infra-estrutura de suporte, to-

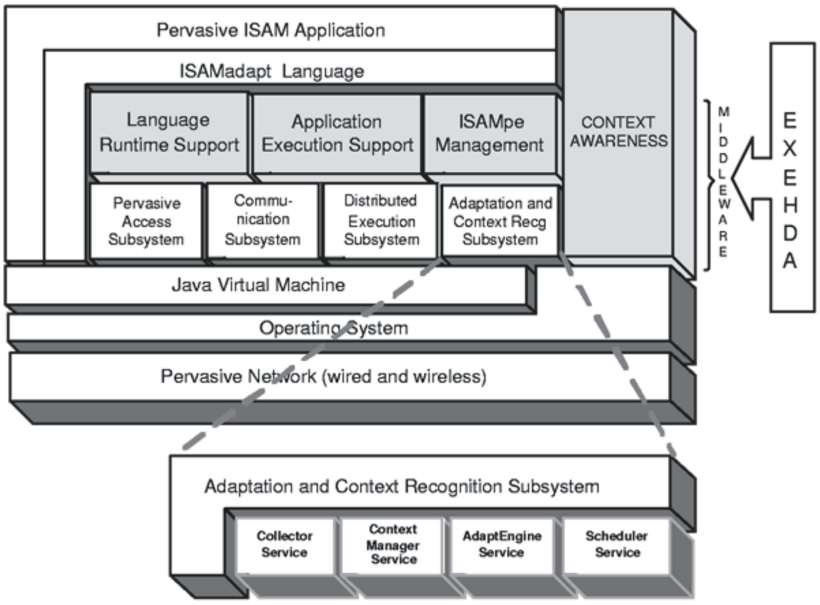

Figura 4.3: Subsistema de reconhecimento de contexto e adaptação do EXEHDA (YAMIN, 2004)

dos instanciados pelo seu respectivo perfil de execução do *middleware*. A integração dos cenários da computação em grade, da computação móvel e da computação sensível ao contexto, é mapeada em uma organização composta pela agregação de células de execução do EXEHDA, conforme pode ser visto na figura 4.4 (ISAM, 2007).

O meio físico sobre o qual o ISAMpe é definido constitui-se por uma rede infra-estruturada, cuja composição final pode ser alterada pela agregação dinâmica de nodos móveis. Os recursos da infra-estrutura física são mapeados para três abstrações básicas, as quais são utilizadas na composição do ISAMpe (YAMIN, 2004):

- **EXEHDAcels:** denota a área de atuação de uma EXEHDAbase, e é composta por esta e por EXEHDAnodos. Os principais aspectos considerados na definição da abrangência de uma célula são: o escopo institucional, a proximidade geográfica e o custo de comunicação;

- **EXEHDAbase:** é o ponto de contato para os EXEHDAnodos. É responsável por todos os serviços básicos do ISAMpe e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre vários equipamentos;

- **EXEHDAnodo:** são os equipamentos de processamento disponíveis no ISAMpe, sendo responsáveis pela execução das aplicações. Um subcaso deste tipo de recurso é o **EXEHDAnodo móvel**. São os nodos do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem fio e, neste caso, integram a célula a qual seu ponto-de-acesso está subordinado. São funcionalmente análogos aos EXEHDAnodos, porém eventualmente com uma capacidade mais restrita (por exemplo, PDAs).

O ISAMpe é formado por equipamentos multi-institucionais, o que gera a necessidade de adotar procedimentos de gerência iguais aos utilizados em ambientes de

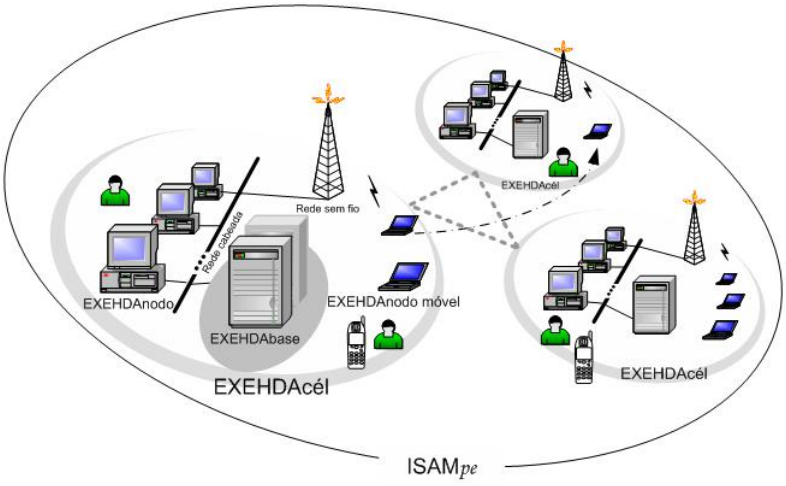

Figura 4.4: ISAM *Pervasive Environment*
(YAMIN, 2004)

Grade Computacional (*Grid Computing*) (GEYER et al., 2002) (YAMIN et al., 2003). O gerenciamento da organização celular do ISAMpe resguarda a autonomia das instituições envolvidas. Apesar de não contemplar mecanismos de gerência específicos para recursos especializados como impressoras, scanners, etc., o EXEHDA permite a catalogação de tais recursos como integrantes de uma determinada célula do ISAMpe, tornando-os, desta forma, passíveis de serem localizados dinamicamente e serem utilizados pelas aplicações pervasivas.

4.5.1.3 Composição baseada em serviços

O requisito de operação em um ambiente altamente heterogêneo, onde não só o hardware exibe capacidades variadas de processamento e memória, mas também as bibliotecas de software disponíveis em cada dispositivo, motivaram a adoção de uma abordagem na qual um núcleo mínimo do *middleware* tem suas funcionalidades estendidas por serviços carregados sob demanda. Esta organização reflete um padrão de projeto referenciado na literatura como micro-kernel (BUSCHMANN, 1996). Some-se a isto o fato de que esta carga sob demanda tem perfil adaptativo. Deste modo, poderá ser utilizada versão de um determinado serviço, melhor sintonizada às características do dispositivo em questão. Isto é possível porque, na modelagem do EXEHDA, os serviços estão definidos por sua interface, e não pela sua implementação propriamente dita.

A contra-proposta à estratégia micro-kernel de um único binário monolítico, cujas funcionalidades cobrissem todas as combinações de necessidades das aplicações e dispositivos, se mostra impraticável na Computação Pervasiva, cujo ambiente computacional apresenta elevada heterogeneidade de recursos de processamento.

Por sua vez, o requisito do *middleware* de manter-se operacional durante os períodos de desconexão planejada motivou, além da concepção de primitivas de comunicação adequadas a esta situação, a separação dos serviços que implementam operações de natureza distribuída em instâncias locais ao EXEHDAnodo (instância nodal), e instâncias locais a EXEHDAbase (instância celular). Neste sentido, o relacionamento entre instância de nodo e celular assemelha-se à estratégia de *Proxies*, enquanto que o relacionamento entre instâncias celulares assume um caráter P2P. A abordagem

P2P nas operações inter-celulares vai ao encontro do requisito de escalabilidade. Uma organização dos subsistemas do EXEHDA pode ser visto na figura 4.5.

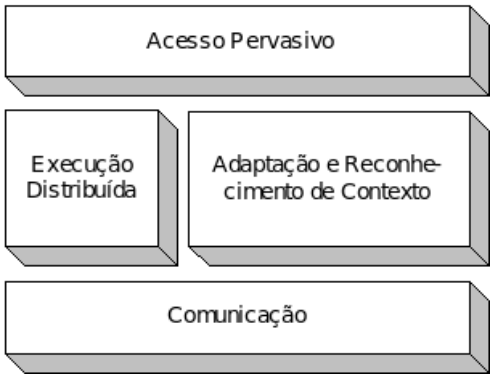

Figura 4.5: Organização dos subsistemas do EXEHDA
(YAMIN, 2004)

Com isso, os componentes da aplicação em execução em determinado dispositivo podem permanecer operacionais, desde que, para satisfação de uma dada requisição pelo *middleware*, o acesso a um recurso externo ao dispositivo seja prescindível. Por outro lado, a instância celular, em execução na base da célula, provê uma referência para os outros recursos, no caso da realização de operações que requeiram coordenação distribuída. Neste sentido, observe-se que a EXEHDAbase é, por definição, uma entidade estável dentro da EXEHDAcel, permitindo que os demais integrantes (recursos) da célula tenham um caráter mais dinâmico no que se refere a sua disponibilidade (presença efetiva) na célula.

4.5.1.4 O núcleo do EXEHDA

A funcionalidade provida pelo EXEHDA é personalizável no nível de nodo, sendo determinada pelo conjunto de serviços ativos e controlada por meio de perfis de execução. Um perfil de execução define um conjunto de serviços a ser ativado em um EXEHDAnodo, associando a cada serviço uma implementação específica dentre as disponíveis, bem como definindo parâmetros para sua execução. Adicionalmente, o perfil de execução também controla a política de carga a ser utilizada para um determinado serviço, a qual se traduz em duas opções: (*i*) quando da ativação do nodo (*bootstrap* do *middleware*) e (*ii*) sob demanda.

Desta maneira, a informação definida nos perfis de execução é também consultada quando da carga de serviços sob demanda, assim, a estratégia adaptativa para carga dos serviços acontece tanto na inicialização do nodo, quanto após este já estar em operação e precisar instalar um novo serviço. Esta política para carga dos serviços é disponibilizada por um núcleo mínimo do EXEHDA, o qual é instalado em todo EXEHDAnodo que for integrado ao ISAMpe. Este núcleo é formado por dois componentes, conforme figura 4.6:

- **ProfileManager:** interpreta a informação disponível nos perfis de execução e a disponibiliza aos outros serviços do *middleware*. Cada EXEHDAnodo tem um perfil de execução individualizado;
- **ServiceManager:** realiza a ativação dos serviços no EXEHDAnodo a partir das informações disponibilizadas pelo *ProfileManager*. Para isto, carrega sob demanda

o código dos serviços do *middleware*, a partir do repositório de serviços que pode ser local ou remoto, dependendo da capacidade de armazenamento do EXEHDAnodo e da natureza do serviço.

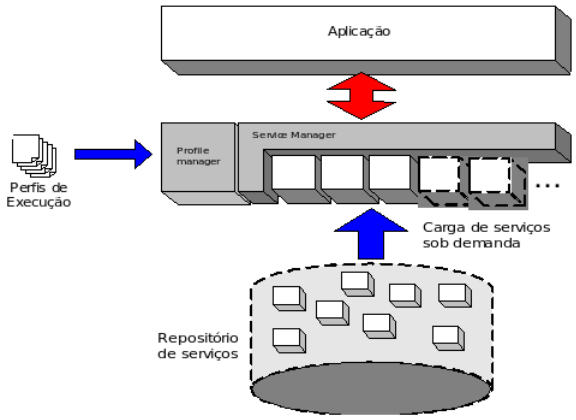

Figura 4.6: Organização do núcleo do EXEHDA
(YAMIN, 2004)

4.5.2 Subsistemas do EXEHDA

4.5.2.1 Subsistema de execução distribuída

O Subsistema de Execução Distribuída é responsável pelo suporte ao processamento distribuído no EXEHDA. No intuito de promover uma execução efetivamente pervasiva, este subsistema interage com outros subsistemas do EXEHDA. Em específico, interage com o subsistema de reconhecimento de contexto e adaptação, de forma a prover comportamento distribuído e adaptativo às aplicações da Computação Pervasiva. Este subsistema é constituído pelos seguintes componentes:

Executor

Serviço que acumula as funções de disparo de aplicações, e de criação e migração de seus objetos. Na implementação destas funções é empregada a instalação de código sob demanda. Para tal, interage com os serviços *CIB* e *BDA*, que podem ser vistos no itens seguintes.

Cell Information Base - CIB

Serviço que implementa a base de informações da célula. Sua principal funcionalidade está relacionada à manutenção da infra-estrutura distribuída que forma o ISAMpe.

OXManager

A abstração OX (Objeto eXehda (EXEHDA)), provida pelo *middleware* às aplicações, consiste em uma instância de objeto, criada por intermédio do serviço *Executor*, à qual pode ser associada meta-informação em tempo de execução. No caso da abstração OX, esta meta-informação ocorre na forma de atributos, i.e., pares <nome, valor>, sendo que "nome" é uma cadeia de caracteres ASCII que descreve o atributo, podendo "valor" ser um conteúdo genérico, inclusive de natureza binária. A gerência

e manutenção da meta-informação associada a um OX é atribuição do serviço *OXManager*. Este serviço confere às operações de consulta e atualização dos atributos do OX o necessário caráter pervasivo, permitindo que estes sejam acessados a partir de qualquer nodo do ISAMpe.

Discoverer

O serviço de descoberta de recursos responsável pela localização de recursos especializados no ISAMpe a partir de especificações abstratas dos mesmos. As especificações caracterizam o recurso a ser descoberto por meio de atributos e seus respectivos valores. Adicionalmente, a requisição de descoberta de recurso incorpora um parâmetro que define a amplitude da pesquisa. Nesse sentido, os seguintes valores estão definidos:

- 0: próprio nodo;
- 1: segmento local;
- 2: célula local;
- 3: vizinhança estática da célula local;
- 4: vizinhança completa da célula local, nível 1;
- 5: vizinhança completa da célula local, nível 2.

ResourceBroker

O controle da alocação de recursos às aplicações no EXEHDA é desempenhado pelo serviço *ResourceBroker*, o qual atende tanto requisições originárias da própria EXEHDAcel quanto oriundas de outras células do ISAMpe.

Gateway

Serviço que faz a intermediação das comunicações entre os nodos externos à célula e os recursos internos a ela. Da sua ação integrada com o *ResourceBroker* decorre o controle de acesso aos recursos de uma EXEHDAcel. Pode-se sintetizar que: *controle de acesso entre células = Gateway + ResourceBroker*.

StdStreams

Serviço que provê o suporte ao redirecionamento dos *streams* padrões de entrada, saída e erro. Sua funcionalidade se dá numa perspectiva por aplicação, sem a necessidade de modificação no código da mesma. Para isto, define que cada aplicação em execução em um determinado EXEHDAnodo possui um componente Console individualizado, o qual agrupa os três *streams* padrões.

Logger

A funcionalidade de registro de rastro de execução (logging) é amplamente empregada em sistemas computacionais. Na fase de desenvolvimento, pode ser empregada para depuração de um programa auxiliando a identificar comportamentos errôneos ou imprevistos (gargalos de execução - *bottlenecks*). Por sua vez, considerando a segurança dos sistemas computacionais, esta funcionalidade é freqüentemente empregada para registro

de operações importantes e/ou críticas realizadas, facilitando a identificação de situações de intrusão, ou de uso indevido do sistema. Para atendimento destes aspectos, é disponibilizado o serviço Logger

Dynamic Configurator - DC

Serviço que tem como objetivo realizar a configuração do perfil de execução do *middleware* em um determinado EXEHDAnodo de forma automatizada.

4.5.2.2 Subsistema de comunicação

A natureza da mobilidade do hardware e, na maioria das vezes, também a do software, não garante a interação contínua entre os componentes da aplicação distribuída. As desconexões são comuns, não somente devido à existência de alguns links sem fio, mas sobretudo como uma estratégia para economia de energia nos dispositivos móveis. O subsistema de comunicação do EXEHDA disponibiliza mecanismos que atendem estes aspectos da Computação Pervasiva. Integram este subsistema os serviços *Dispatcher, WORB, CCManager* os quais contemplam modelos com níveis diferenciados de abstração para as comunicações.

Dispatcher

Serviço que disponibiliza o modelo de comunicação mais elementar do EXEHDA: troca de mensagens ponto-a-ponto com garantia de entrega e ordenamento das mensagens, o qual é especializado para operação no ambiente pervasivo. As mensagens trafegam entre instâncias do *Dispatcher* localizadas em nodos diferentes através de estruturas denominadas canais. Nesse sentido, quando de sua inicialização, o *Dispatcher* atualiza a informação do EXEHDAnodo na CIB (Cell Information Base), em específico o atributo *contactAddress*, provendo uma lista de protocolos e endereços que podem ser utilizados para alcançar aquele EXEHDAnodo.

WORB

Serviço que tem o objetivo de simplificar a construção de serviços distribuídos, permitindo que os programadores focalizem esforços no refinamento da semântica distribuída associada ao serviço em desenvolvimento, abstraindo aspectos de baixo nível relativos ao tratamento das comunicações em rede. Para tanto, oferece um modelo de comunicação baseado em invocações remotas de método, similar ao RMI, porém sem exigir a manutenção da conexão durante toda a execução da chamada remota.

CCManager

Considerando a premissa da mobilidade lógica dos componentes que constituem as aplicações pervasivas, o suporte a um mecanismo de comunicação com característica de desacoplamento temporal e espacial é particularmente oportuno, à medida que este simplifica a construção de tais aplicações. Este serviço vem atender a esta demanda, disponibilizando um mecanismo baseado na abstração espaço de tuplas, o qual prescinde da coexistência temporal de emissor e receptor. Outro aspecto oportuno desta abstração é a facilidade com que podem ser implementados outros padrões de comunicação, além do ponto-a-ponto.

4.5.2.3 Subsistema de acesso pervasivo

A premissa de acesso em qualquer lugar, todo o tempo, a dados e código da Computação Pervasiva, requer um suporte do *middleware*. Os serviços que compõem este subsistema no EXEHDA são:

BDA-Base de Dados pervasiva das Aplicações

O ambiente de Computação Pervasiva tem por premissas (i) a possibilidade de o usuário disparar aplicações a partir de qualquer nodo integrante do sistema e, após o disparo, (ii) a mobilidade parcial ou integral de tais aplicações em resposta a modificações em seu contexto de execução, como, por exemplo, a movimentação do usuário ou alteração na condição de carga dos dispositivos atualmente em uso pela aplicação. Para tanto, a capacidade de instalação de código sob demanda é uma necessidade inerente à execução de aplicações na Computação Pervasiva. Seria impraticável manter todo o universo de software disponível instalado e atualizado (com coerência de versões) em todos os dispositivos do sistema. Por sua vez, a implementação do mecanismo de instalação sob demanda requer a existência de um repositório de código que forneça a mesma visão do software disponibilizado a partir de qualquer dispositivo do ISAMpe, mesmo após migrações. Outras funcionalidades para este repositório pervasivo também são desejáveis. Considerando a perspectiva de que as diversas aplicações disponibilizadas sigam linhas de evolução independentes, o suporte a controle de versões é oportuno na direção da manutenção da operacionalidade das diferentes aplicações.

AVU - Ambiente Virtual do Usuário

A premissa *siga-me* definida no ISAM reflete-se não só nas aplicações que um usuário atualmente executa, mas no seu ambiente computacional como um todo. Este engloba, além das aplicações em execução, as informações de personalização das aplicações definidas pelo usuário, o conjunto de aplicações instaladas (i.e. passíveis de serem disparadas por aquele usuário), como também seus arquivos privados. É atribuição do serviço *AVU* (Ambiente Virtual do Usuário) do EXEHDA a manutenção do acesso pervasivo a este ambiente virtual, da forma mais eficiente possível.

SessionManager

Serviço responsável pela gerência da sessão de trabalho do usuário, sendo definida pelo conjunto de aplicações correntemente em execução para aquele usuário. A informação que descreve o estado da sessão de trabalho é armazenada no *AVU*, estando portando disponível de forma pervasiva.

Gatekeeper

Serviço responsável por intermediar acessos entre as entidades externas à plataforma ISAM e os serviços do *middleware* de execução, conduzindo os procedimentos de autenticação necessários.

4.5.2.4 Subsistema de reconhecimento de contexto e adaptação

Integram este subsistema os serviços (*i*) Collector, (*ii*) Deflector, (*iii*) ContextManager, (*iv*) AdaptEngine e (*v*) Scheduler. Particularmente, *AdaptEngine* e *Scheduler* são responsáveis, respectivamente, pelo controle das adaptações de cunho funcional e não-

funcional. Entende-se por adaptação funcional aquela que implica a modificação do código sendo executado. Por sua vez, adaptação não-funcional, é aquela que atua sobre a gerência da execução distribuída, na qual podem ser identificados seus diversos serviços, os quais serão detalhados a seguir (YAMIN, 2004):

Collector

Responsável pela extração da informação bruta (diretamente dos recursos envolvidos) que, posteriormente refinada, dará origem aos elementos de contexto. Para isto, o serviço *Collector* aglutina a informação oriunda de vários componentes monitores *Monitor* e as repassa aos consumidores registrados *MonitoringConsumer*. Um componente *Monitor* gerencia um conjunto de sensores parametrizáveis. Por outro lado, entre os consumidores da informação extraída pela monitoração estão o serviços *ContextManager* e *Scheduler*.

Vale salientar que em decorrência da organização distribuída do EXEHDA o coletor de um EXEHDAnodo pode também, se necessário, registrar-se como consumidor perante o coletor de outro EXEHDAnodo.

Na arquitetura de monitoração, cada sensor contribui com a extração de um valor que descreve um aspecto específico, estático ou dinâmico, do recurso sendo monitorado. O conjunto dos sensores existentes em um dado EXEHDAnodo, assim como os parâmetros suportados por cada um destes sensores, integra a informação de descrição daquele nodo, disponibilizada no serviço *CIB*.

O serviço *Collector*, é responsável pela modificação dos parâmetros de operação dos sensores. A atuação do *Collector* ocorre por intermédio do Monitor associado ao sensor sendo re-parametrizado. É responsabilidade do *Collector* estabelecer parâmetros de operação que satisfaçam o caso mais exigente, considerando as necessidades dos consumidores registrados daquele sensor.

O controle da condição de publicação entre o *Collector* e os consumidores registrados é feito por meio do parâmetro *threashold*, que configura o limiar de variação do dado, de forma individualizada para cada sensor, acima do qual uma publicação do novo valor registrado pelo sensor é realizada. A extração da informação junto aos monitores não ocorre em momentos arbitrários, mas apenas em instantes discretos, múltiplos de um determinado *quantum* de tempo. O *quantum* é um dos parâmetros de configuração do serviço *Collector* em cada EXEHDAnodo, sendo utilizado para controlar o grau de intrusão do mecanismo de monitoração.

Transcorrido um quantum de tempo, o *Collector* notifica os monitores através do método *quantumExpired*, os quais então, executam uma operação de *polling* em cada um dos sensores ativos naquele momento no seu nodo. O critério de publicação *(treashold)* especificado para o sensor é aplicado, determinando a geração, ou não, de um evento para aquele sensor. Dessa forma, todos os eventos gerados dentro de um *quantum* são agrupados em uma única mensagem, reduzindo o tráfego de dados até os consumidores registrados.

Deflector

Disponibiliza abstração de canais de multicast para uso na disseminação das informações monitoradas. A presença do serviço *Deflector* é decorrência da busca de escalabilidade para a arquitetura de monitoração do EXEHDA.

ContextManager

Responsável pelo refinamento (tratamento) da informação bruta produzida pela monitoração para produção de informações abstratas referentes aos elementos de contexto *(valores contextualizados)*. É recebida como parâmetro uma descrição (XML) de como o dado referente àquele elemento de contexto deve ser produzido a partir da informação proveniente da monitoração.

AdaptEngine

Responsável pelo controle das adaptações de cunho funcional, este serviço provê facilidades para definição e gerência de comportamentos adaptativos por parte das aplicações. Deste modo, libera o programador de gerenciar os aspectos de mais baixo nível envolvidos na definição e liberação dos elementos de contexto junto ao *ContextManager*.

Outra função do serviço *AdaptEngine* é prover um mecanismo de carga de código contextualizado. Desta forma, com base na informação do estado do elemento de contexto fornecido pelo serviço *ContextManager*, seleciona e carrega o código correspondente dentre alternativas de código definidas na política de adaptação funcional vigente. Este serviço disponibiliza ainda métodos para a ativação de ações quando determinado estado de um elemento de contexto tornar-se ativo.

Scheduler

Serviço central na gerência das adaptações de cunho não-funcional no EXEHDA, isto é, que não implicam alteração de código. Nesse sentido, o *Scheduler* emprega a informação de monitoração, obtida junto ao serviço *Collector*, para orientar operações de mapeamento. Essas operações decorrem de instanciações remotas ou migrações realizadas pelo serviço *Executor*, ou quando de chamadas de re-escalonamento, originadas do estado atual de um recurso não satisfazer mais as necessidades de um objeto anteriormente a ele alocado.

4.6 Considerações finais

Neste capítulo foram apresentados os conceitos e as tecnologias relacionadas com ontologias. Também, foram discutidas as principais motivações para uso de ontologias, evidenciando os motivos para sua aplicação na área da Ciência da Computação. Quanto as linguagens utilizadas para construção de ontologias, destaca-se as destinadas a aplicações para *Web* Semântica, especialmente a OWL. Dentre as ferramentas de desenvolvimento, o editor Protégé, caracteriza-se como a ferramenta com maior capacidade de extensão e independência de plataforma.

Ainda neste capítulo foram exploradas as características e funcionalidades do *middleware* EXEHDA, direcionado para Computação Pervasiva. Considerando o foco do trabalho, destacou-se o subsistema de reconhecimento de contexto e adaptação, especialmente os serviços e componentes que fazem parte da arquitetura de software.

Com base nos princípios discutidos neste capítulo, no capítulo 5 será apresentada a modelagem proposta para o EXEHDA-ON, bem como os protótipos desenvolvidos e os testes realizados.

5 EXEHDA-ON: MODELAGEM E PROTOTIPAÇÃO

Neste capítulo são tratados os aspectos referentes a materialização dos esforços de pesquisa do EXEHDA-ON, sendo então discutidos aspectos pertinentes a concepção da sua arquitetura de software, bem como do modelo ontológico projetado para representação e processamento dos dados de contexto do ambiente pervasivo previsto. Também são discutidos aspectos relacionados à prototipação e testes do EXEHDA-ON, sendo apresentados alguns casos de emprego, e destacadas as tecnologias utilizadas.

5.1 Aspectos de modelagem do EXEHDA-ON

A modelagem do EXEHDA-ON contemplou dois principais esforços de concepção: (*i*) modelagem ontológica do ambiente pervasivo e do contexto de interesse das aplicações. Fundamentalmente, esta modelagem prevê o uso de ontologias implementadas em OWL (*Web Ontology Language*), sobre as quais o EXEHDA-ON realiza a representação e o processamento das informações de contexto; (*ii*) modelagem da arquitetura de software com a especificação dos diferentes serviços que a integram.

5.1.1 Considerações gerais

A construção de um suporte à sensibilidade ao contexto para as aplicações distribuídas em larga escala, apresenta inúmeros desafios, os quais se relacionam especialmente a obtenção, modelagem, armazenamento, processamento, distribuição e monitoramento do contexto. Dentre estes desafios, a modelagem e o processamento do contexto são preocupações centrais neste trabalho.

A estratégia inicialmente prevista para a produção, em nível abstrato, da informação de contexto, consiste no emprego de filtros denominados cadeias de detecção de contexto para cada contexto registrado, como ilustrado na figura 5.1. Estas cadeias são compostas por três elementos: (*i*) *aggregator*, (*ii*) *translator*, (*iii*) *notifier*. O *aggregator* é responsável pela composição dos dados de um ou mais sensores para a produção de valor dito agregado. O dado agregado é repassado ao componente tradutor para conversão do mesmo em um valor abstrato, conforme definido pelo programador da aplicação. O *notifier*, por sua vez, detecta alterações no dado abstrato gerado pelo tradutor *translator*, gerando um evento de modificação de contexto. Uma visão geral desta proposta foi apresentada em (AUGUSTIN, 2004) e (YAMIN, 2004).

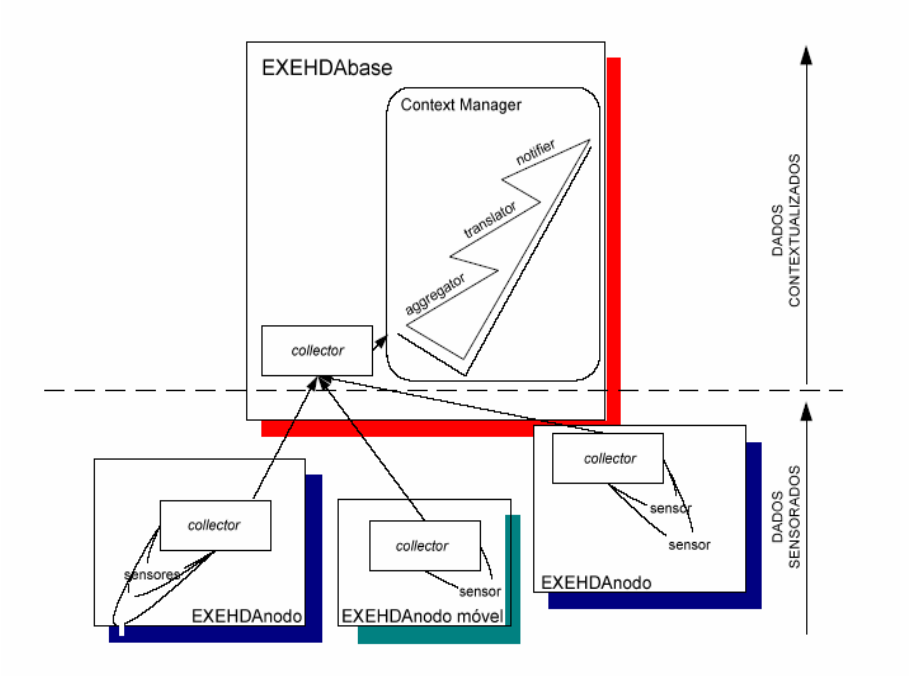

Figura 5.1: Construção da informação de contexto
(AUGUSTIN, 2004)

5.1.2 Modelagem do ambiente pervasivo tratado pelo EXEHDA-ON

O modelo ontológico para uso no EXEHDA-ON foi definido considerando aspectos que modelassem o domínio do ambiente pervasivo provido pelo EXEHDA. A perspectiva é que este modelo represente o estado atual do ambiente de execução pervasivo provido pelo EXEHDA (vide figura 4.4), gerando deste modo um conhecimento sobre o mesmo, possibilitando assim sua manipulação pelo servidor de contexto, o qual responde às demandas introduzidas pelas aplicações dos usuários.

A natureza da informação de contexto, nas células de execução, pode ser estática ou dinâmica. Um exemplo de informação estática são os atributos relativos ao tipo do dispositivo, os quais não se alteram com o passar do tempo. Informações dinâmicas traduzem aspectos do contexto que oscilam em uma determinada freqüência. Como exemplo de informações dinâmicas têm-se: a temperatura do processador, a ocupação de memória e a ocupação de disco. Estas informações são obtidas por meio de monitoramento periódico ou coletadas por procedimentos disparados por eventos (YAMIN, 2004).

Como decorrência das discussões apresentadas na seção 4.4.1 do capítulo 4, a metodologia utilizada para a construção da ontologia descritiva do ambiente pervasivo é baseada nas propostas de (FERNÁNDEZ; GÓMES-PÉREZ; JURISTO, 1997) e (GRUBER, 1993). A seguir são descritos os passos utilizados na definição da ontologia projetada para uso no EXEHDA-ON:

- definição do domínio e captura do conhecimento;
- conceituação do conhecimento capturado em um conjunto de representações;
- desenvolvimento do modelo conceitual em uma linguagem formal.

No desenvolvimento da ontologia para o ambiente pervasivo tratado pelo EXEHDA-ON buscou-se o conhecimento do domínio, caracterizando-se os tipos de dispositivos, os sensores, os componentes de software e a infra-estrutura das redes de interconexão.

Entre os resultados da etapa de "conceituação do conhecimento capturado em um conjunto de representações", destaca-se o desenvolvimento de um glossário de termos, contendo os conceitos pesquisados, sua descrição e atributos relacionados. Este glossário é apresentado nas tabelas 5.1 e 5.2. Também, foram elaboradas tabelas de atributos de classe. A tabela 5.3 mostra os atributos do conceito "Fixo".

Ainda nesta etapa foi construída uma árvore de classificação de conceitos, que permite visualizar as classes definidas para a ontologia do ambiente pervasivo tratado pelo EXEHDA-ON, mostrando sua hierarquia (vide figura 5.2).

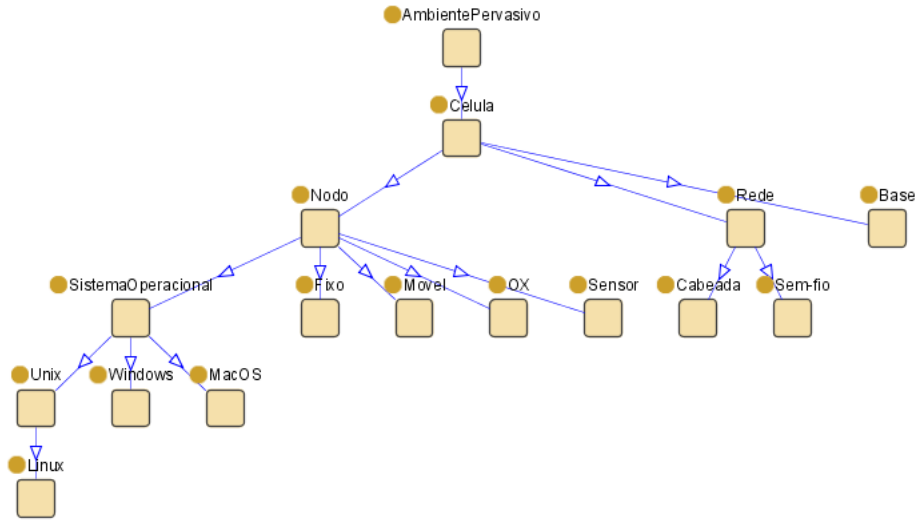

Figura 5.2: Árvore de conceitos da ontologia do ambiente pervasivo

Em função das discussões relacionadas com linguagens e ferramentas para construção de ontologias apresentadas no capítulo 4, o desenvolvimento do modelo conceitual foi feito na linguagem OWL-DL (OWL - *Description Logics*) com uso do editor Protégé versão 3.3.1.

5.1.3 Modelagem do contexto de interesse das aplicações

O modelo ontológico para uso no EXEHDA-ON também prevê a representação do contexto de interesse das aplicações através de uma ontologia construída com a mesma metodologia descrita na seção anterior (seção 5.1.2). O contexto de interesse da aplicação é um subconjunto do contexto geral do ambiente pervasivo, quando o contexto de interesse ocorre a aplicação é notificada. Na figura 5.3 é possível visualizar as classes definidas para a ontologia do contexto de interesse das aplicações, mostrando sua hierarquia.

As classes e atributos da ontologia do contexto de interesse representam as principais características dos nodos, com exceção dos atributos "TipoOperador" e "Extensao". O primeiro atributo permite estabelecer uma comparação entre os valores atribuídos aos termos na ontologia do contexto de interesse com os valores aos termos que represen-

Tabela 5.1: Glossário de termos da ontologia do ambiente pervasivo

Classes	Descrição	Atributos
Ambiente Pervasivo	*Representa o ambiente de execução pervasivo provido pelo EXEHDA*	—
Celula	*Especifica quais são as células do ambiente pervasivo. Denota a área de atuação de uma Base e é composta por esta e por Nodos*	CelulaID
Base	*Responsável por todos os serviços básicos do ambiente pervasivo e, embora constitua uma referência lógica única, seus serviços, sobretudo por aspectos de escalabilidade, poderão estar distribuídos entre vários equipamentos*	CelulaID
Nodo	*Equipamentos de processamento disponíveis no ambiente pervasivo, sendo responsáveis pela execução das aplicações. Abrange as sub-classes Fixo, Movel, Sensor e OX*	CelulaID, NodoID
Fixo	*Corresponde aos nodos interconectados, em geral, por meio de redes cabeadas*	CelulaID, NodoID, TipoNodo, NomeSO, NumProcessadores, NumNucleos, VelocidadeCPU, PoderComp, TotMemoria, TotDisco, AtivNodo, OcupMemoria, CargaCPU, DiscoDisponivel, TempProcessador
Movel	*Corresponde aos nodos do sistema com elevada portabilidade, tipicamente dotados de interface de rede para operação sem fio e, neste caso, integram a célula a qual seu ponto-de-acesso está subordinado*	CelulaID, NodoID, TipoNodo, NomeSO, NumNucleos, VelocidadeCPU, PoderComp, TotMemoria, TotDisco, AtivNodo, OcupMemoria, CargaCPU, DiscoDisponivel, TempProcessador
Sensor	*Corresponde aos sensores existentes nos nodos*	ValorSensor
OX	*Objeto eXehda (EXEHDA) corresponde aos componentes de software instalados nos nodos*	CompOX

Tabela 5.2: Continuação do glossário de termos da ontologia do ambiente pervasivo

Classes	Descrição	Atributos
SistemaOperacional	*Corresponde aos sistemas operacionais dos nodos*	CelulaID, NodoID
MacOS	*Representa o sistema operacional MacOS*	CelulaID, NodoID, TipoOS, VersaoOS
Unix	*Representa o sistema operacional Unix*	CelulaID, NodoID, TipoOS, VersaoOS
Windows	*Representa o sistema operacional Windows*	CelulaID, NodoID, TipoOS, VersaoOS
Linux	*Representa o sistema operacional Linux*	CelulaID, NodoID, TipoOS, VersaoOS
Rede	*Redes de interconexão disponíveis na célula. Abrange as sub-classes Cabeada e Sem-fio*	CelulaID
Cabeada	*Redes cabeadas, já existentes na célula, para interconexão dos nodos base e/ou nodos fixos ou móveis*	CelulaID, RedeID, BandaRede, LatenciaRede, TipoConexao
Sem-fio	*Redes sem-fio, já existentes na célula, para interconexão dos nodos base e/ou nodos fixos ou móveis*	CelulaID, RedeID, BandaRede, LatenciaRede, TipoConexao

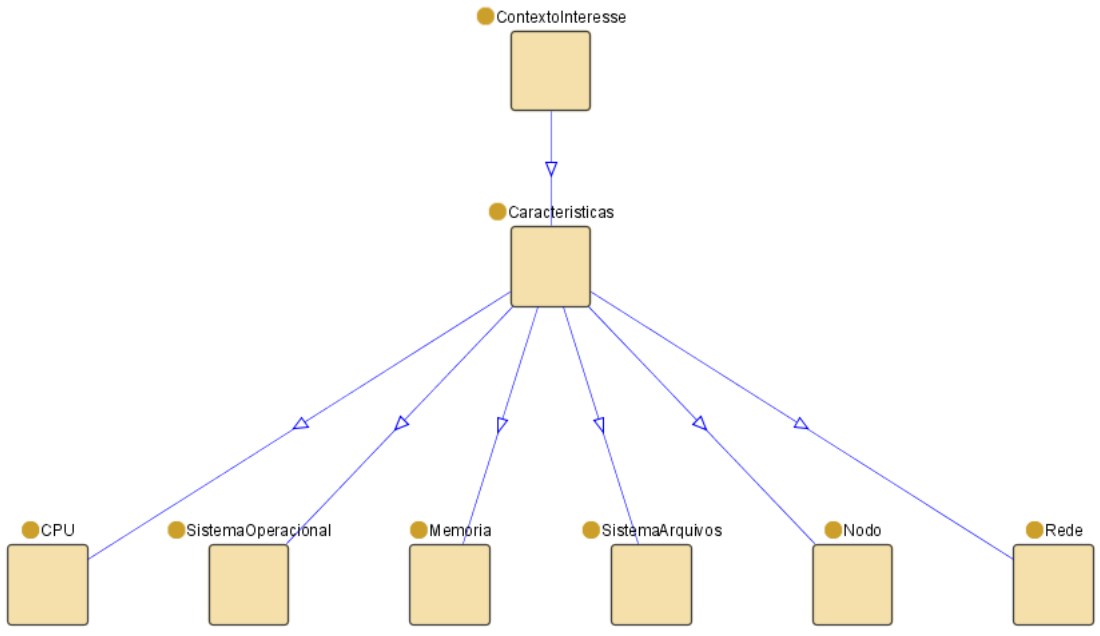

Figura 5.3: Árvore de conceitos da ontologia do contexto de interesse das aplicações

Tabela 5.3: Tabela de atributos da classe “Fixo”

Atributo	Descrição	Tipo	Unidade
CelulaID	*Representa a identificação da célula*	*string*	—
NodoID	*Representa a identificação do nodo*	*string*	—
TipoNodo	*Especifica o tipo de nodo (microcomputador, servidor, ...)*	*string*	—
NomeSO	*Identifica o nome do sistema operacional*	*string*	—
NumProcessadores	*Especifica o número de processadores do nodo*	*int*	—
NumNucleos	*Especifica o número de núcleos por processador*	*int*	—
VelocidadeCPU	*Contém a velocidade de operação da CPU*	*float*	MHz
PoderComp	*Identifica o poder computacional do nodo*	*float*	*BogoMips*
TotMemoria	*Especifica o total de memória disponível no nodo*	*float*	MB
SistArquivos	*Representa a parte do sistema operacional responsável pelo gerenciamento geral dos discos rígidos e pelo armazenamento lógico dos dados*	*float*	GB
TotDisco	*Representa a capacidade total de armazenamento em disco instalada no nodo*	*float*	GB
AtivNodo	*Identifica se existe algum usuário logado no nodo*	*int*	—
OcupMemoria	*Especifica o total de memória que está sendo utilizada no nodo, em um determinado instante de tempo*	*float*	MB
CargaCPU	*Representa a carga média que está sendo imposta à CPU em um intervalo de tempo*	*float*	%
DiscoDisponível	*Contém o total de espaço em disco disponível, em um determinado instante de tempo, no nodo*	*float*	GB
TempProcessador	*Representa a temperatura do processador, em um determinado instante de tempo*	*float*	^{o}C

Tabela 5.4: Glossário de termos da ontologia do contexto de interesse das aplicações

Classes	Descrição	Atributos
ContextoInteresse	*Identifica o contexto de interesse das aplicações*	IDAplicacao, IDContexto
Caracteristicas	*Corresponde às características dos nodos do ambiente pervasivo*	TipoOperador (valores permitidos: Maior, MaiorIgual, Menor, MenorIgual, IgualNumero, IgualString)
CPU	*Representa as características da CPU*	CargaCPU, NumProcessadores, VelocidadeCPU, Extensao
SistemaOperacional	*Representa as características do sistema operacional*	NomeSO, Extensao
Memoria	*Representa as características da memória*	TotMemoria, OcupMemoria, Extensao
SistemaArquivos	*Identifica as características do sistema de arquivos*	SistArquivos, TotDisco, DiscoDisponivel, Extensao
Nodo	*Caracteriza os equipamentos de processamento disponíveis no ambiente pervasivo*	TipoNodo, AtivNodo, Extensao
Rede	*Representa as características das redes de interconexão*	BandaRede, LatenciaRede, TipoConexao, Extensao

tam as características dos nodos na ontologia do ambiente pervasivo. Por sua vez, o atributo "Extensao" é utilizado nas regras de inferência para registrar eventuais extensões semânticas da ontologia do contexto de interesse (vide seção 5.2.2). A tabela 5.4 exibe as classes com suas respectivas descrições e atributos.

5.1.4 Modelagem da arquitetura de software proposta para o EXEHDA-ON

Nesta seção é descrita a proposta para arquitetura de software do EXEHDA-ON e a forma de adequação da mesma ao Subsistema de Reconhecimento de Contexto e Adaptação do EXEHDA, identificando seus pontos de integração e seus serviços.

De modo geral, uma arquitetura para sistemas sensíveis ao contexto envolveria uma série de sensores, de software e/ou de hardware, que monitoram os aspectos de interesse do ambiente computacional (*middleware* e aplicações). As informações colhidas por esses sensores são passadas a um conjunto de serviços de contexto, onde são processadas e/ou modificadas para que para que possam ser entregues aos consumidores das informações contextualizadas. Esta visão está retratada na modelagem conceitual do EXEHDA-ON apresentada na figura 5.4.

A arquitetura do EXEHDA-ON prevê a inclusão de serviços e componentes nos "EXEHDANodos" e no "EXEHDABase", os quais são abstrações do ambiente pervasivo provido pelo EXEHDA. A Figura 5.5 mostra as funcionalidades do EXEHDA-ON integradas ao Subsistema de Reconhecimento de Contexto e Adaptação do EXEHDA. Por questão de compatibilidade com a modelagem de EXEHDA alguns termos da figura 5.5

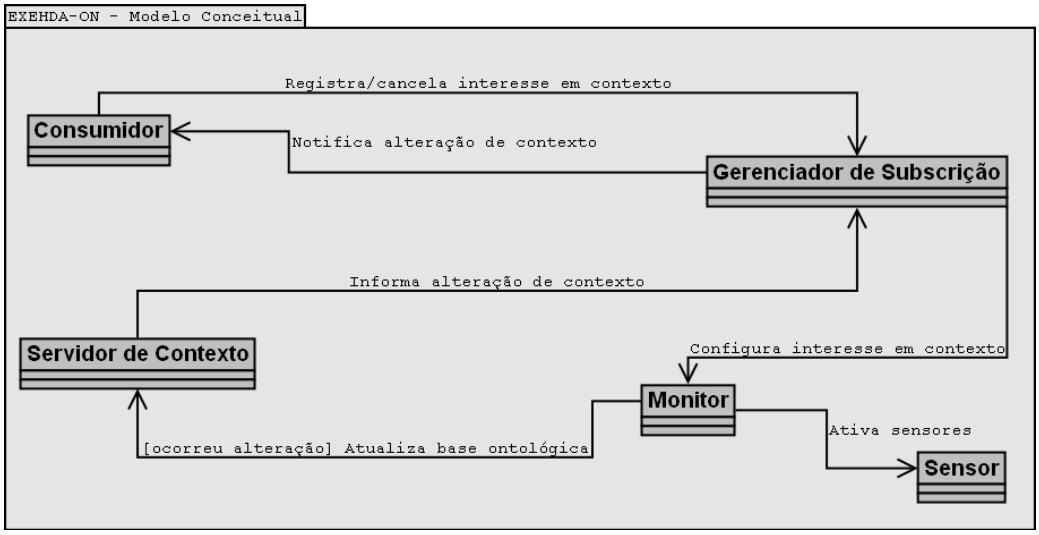

Figura 5.4: Modelo conceitual do EXEHDA-ON

foram mantidos em inglês.

Em cada EXEHDANodo, junto ao componente "Monitor", o serviço do EXEHDA-ON "Gerenciador de monitoramento do contexto" fica com a responsabilidade de receber os dados sensorados e gerar as instâncias da ontologia do ambiente pervasivo, representadas por *strings* em linguagem OWL. Por sua vez, o serviço "Collector" do EXEHDA é responsável pela gerência das comunicações entre os dados sensorados nos nodos e o servidor de contexto que aglutina os dados da célula como um todo.

A localização dos serviços do EXEHDA-ON dentro do EXEHDAbase é junto ao "ContextManager", que é serviço chave na construção de informações globais de contexto, mais especificamente dentro das estruturas de cadeias de detecção de contexto do "ContextManager" (*aggregator* e *translator*), que são empregadas, respectivamente, para composição dos dados de um ou mais sensores e para abstração da informação de contexto.

Assim, no EXEHDAbase atua o serviço responsável pela aglutinação das instâncias na ontologia do ambiente pervasivo, denominado "Gerenciador de atualização da base ontológica", e o serviço responsável pelas consultas e inferências sobre a base ontológica do EXEHDA-ON, denominado "Interpretador de contexto".

5.1.5 Descrição dos serviços do EXEHDA-ON

No EXEHDA-ON estão previstos diversos serviços que gerenciam a obtenção, processamento e disseminação das informações de contexto. A figura 5.6 apresenta uma visão geral destes serviços, os quais são descritos a seguir.

Gerenciador de subscrição

As subscrições correspondem ao meio pelo qual os consumidores têm a possibilidade de registrar suas requisições aos serviços de contexto providos pelo EXEHDA-ON. Assim, este serviço interage com os consumidores das informações de contexto possibilitando o registro de interesse em determinadas condições de contexto. Também, quando o consumidor não tiver mais interesse em alguma informação de contexto este serviço permite o cancelamento da subscrição. Cabe ao "Gerenciador de subscrição" interpretar e

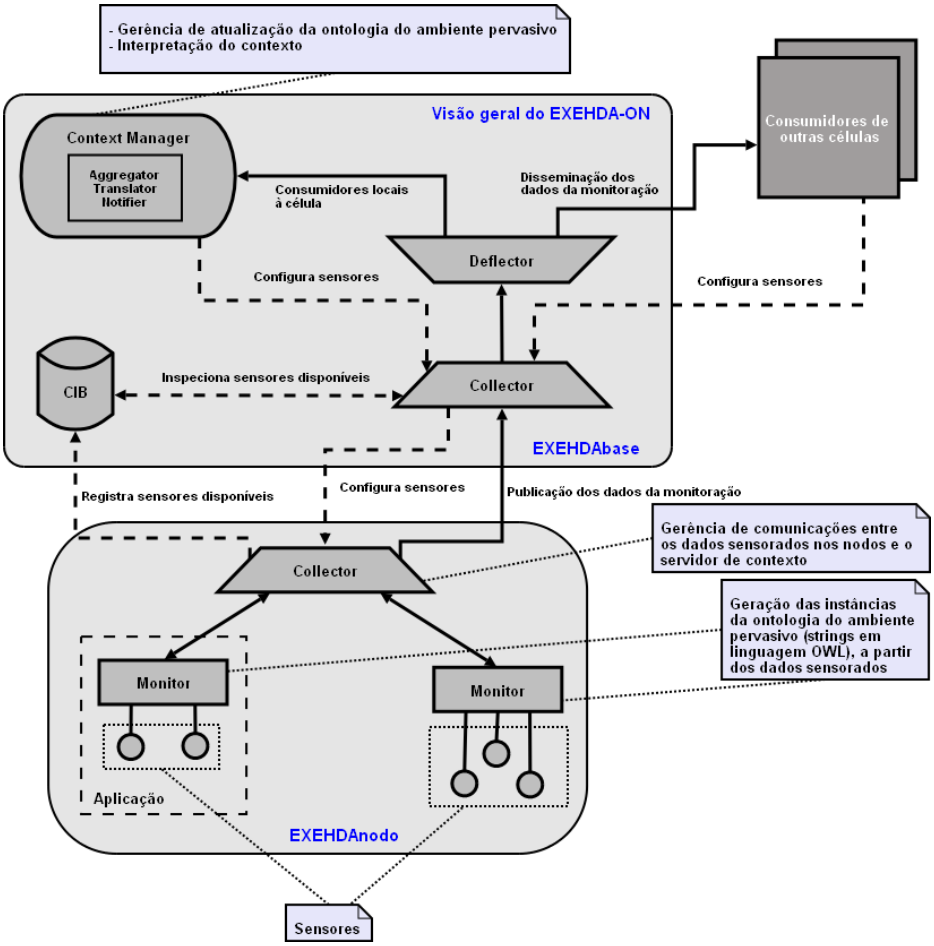

Figura 5.5: Integração das funcionalidades do EXEHDA-ON

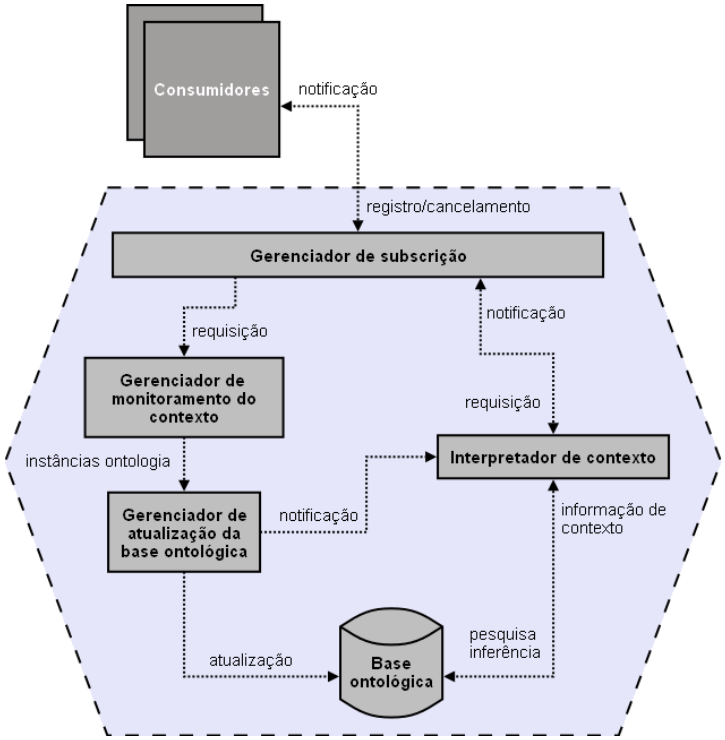

Figura 5.6: Visão geral dos serviços do EXEHDA-ON

gerenciar as solicitações enviadas pelos consumidores e distribuí-las aos demais serviços que formam a arquitetura do EXEHDA-ON, com a intenção de identificar a existência das condições de contexto de interesse do consumidor e notificá-lo de sua existência.

Interpretador de contexto

O "Gerenciador de subscrição" envia para o "Interpretador de contexto" as requisições dos consumidores registrados com o intuito de identificar a existência das condições de contexto de interesse do consumidor. Para executar essa tarefa, o serviço "Interpretador de contexto" pode realizar consultas e inferências sobre a base ontológica.

Gerenciador de monitoramento do contexto

Caso as informações de contexto existentes na base ontológica não atendam ao interesse de algum consumidor registrado, o "Gerenciador de subscrições" encaminha o pedido ao serviço "Gerenciador de monitoramento do contexto", o qual através de arquitetura de monitoração do EXEHDA (*i*) ativa ou desativa sensores em decorrência do interesse dos consumidores; (*ii*) recebe os dados sensorados e (*iii*) gera instâncias da ontologia no formato da linguagem OWL. Estas tarefas possibilitam a atualização da base ontológica.

Gerenciador de atualização da base ontológica

Este serviço recebe as instâncias da ontologia do ambiente pervasivo expressas em linguagem OWL geradas pelo serviço "Gerenciador de monitoramento do contexto" e, através da API Jena (vide seção 5.2.1), manipula a base ontológica realizando a atualização desta ontologia. Também, notifica o "Interpretador de contexto" quando

ocorre uma atualização nesta ontologia.

5.1.6 Detalhamento do serviço de interpretação de contexto do EXEHDA-ON

Dentre os serviços que constituem o EXEHDA-ON apresentados na seção 5.1.5, o serviço "Interpretador de contexto" é central para consecução dos objetivos propostos para o EXEHDA-ON. A figura 5.7 mostra este serviço e os componentes que o constituem.

O "Interpretador de contexto" realiza o processamento das informações de contexto com o intuito de identificar na base ontológica a existência das condições de contexto de interesse dos consumidores registrados. Este serviço recebe as solicitações dos consumidores a partir do "Gerenciador de subscrições" e realiza consultas e inferências sobre a base ontológica, utilizando o componente "Gerenciador de consultas e inferências".

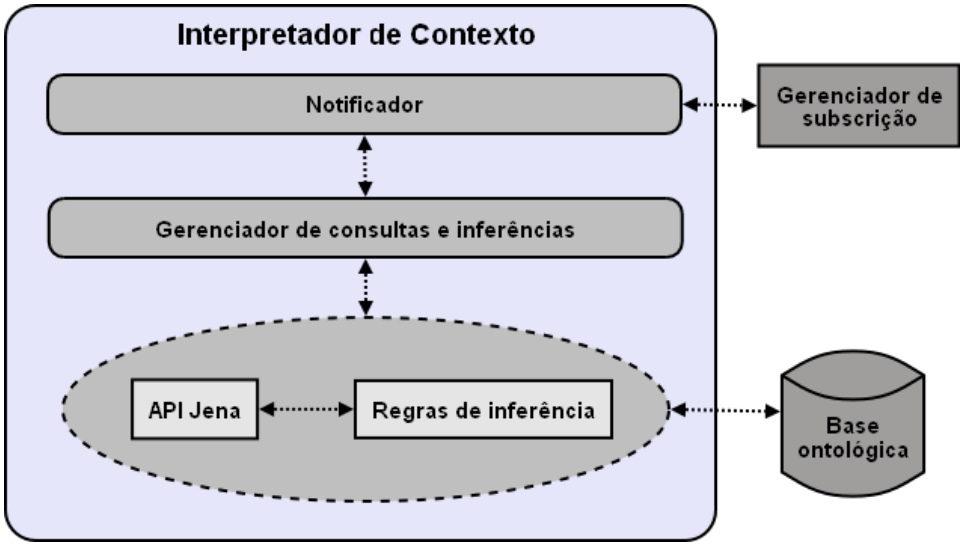

Figura 5.7: Interpretador de contexto do EXEHDA-ON

Gerenciador de consultas e inferências

O serviço de interpretação do contexto possui um componente que realiza consultas à base ontológica. Este componente constrói consultas com a linguagem SPARQL (vide seção 5.2.1), as quais são submetidas à base ontológica. Também, realiza serviços de inferências sobre a ontologia, identificando, por exemplo, eventuais inconsistências que possam existir nas consultas, bem como extraindo informações que não estejam explicitamente escritas na estrutura do modelo ontológico.

A arquitetura proposta para a execução das tarefas relativas ao componente "Gerenciador de consultas e inferências" é apresentada na figura 5.8.

Quando um consumidor registra-se no *middleware* EXEHDA, este deve informar seu contexto de interesse. As informações registradas são processadas pelo componente "Gerenciador de consultas e inferências" com base na ontologia de contexto de interesse da aplicação. Este componente, então, (*i*) analisa os requisitos especificados pelo consumidor, aplicando regras de inferência (vide seção 5.2.2) que permitem verificar a consistência das consultas e estendê-las semanticamente; (*ii*) gerencia a construção das con-

sultas; (*iii*) executa as consultas sobre a ontologia; e (*iv*) notifica as condições de contexto existentes, em função do interesse dos consumidores registrados.

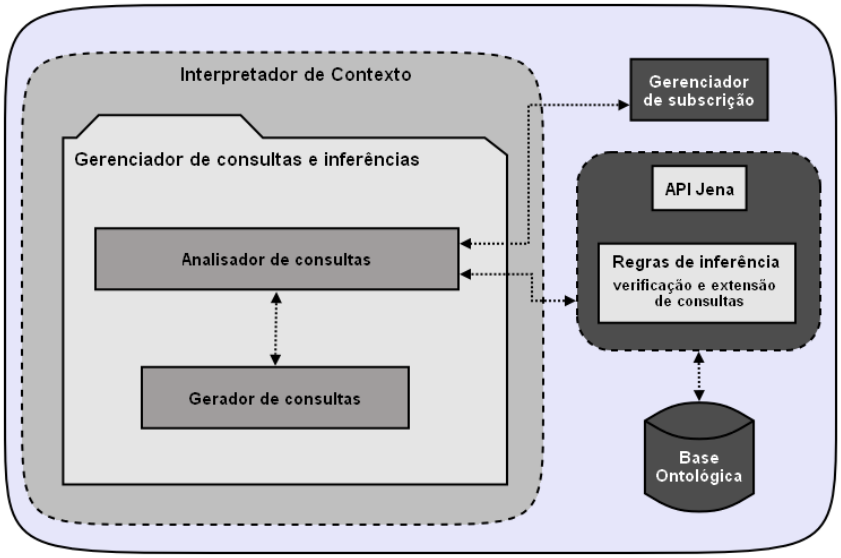

Figura 5.8: Componente Gerenciador de consultas e inferências

5.2 Prototipação e testes do EXEHDA-ON

Nesta seção estão descritos os aspectos de prototipação do EXEHDA-ON, bem como casos de estudo realizados. Também, são caracterizadas as principais tecnologias empregadas na sua implementação.

5.2.1 Tecnologias de Web Semântica utilizadas no EXEHDA-ON

Como um dos elementos básicos que constitui a Web Semântica, as ontologias proporcionam a interoperabilidade semântica entre os contextos das aplicações. Isto significa que é possível explorar aspectos além da interoperabilidade sintática e estrutural fornecida pelos padrões XML (*EXtensible Markup Language*), Esquema XML, RDF (*Resource Description Framework*) e Esquema RDF. Para isso, é necessário utilizar uma linguagem de ontologia: (*i*) compatível com padrões como XML, Esquema XML, RDF e Esquema RDF; (*ii*) com sintaxe rica para representar conhecimento e regras e permitir a inferência de novos dados; (*iii*) de fácil compreensão e extensão; e (*iv*) que tenha suporte eficiente de ferramentas para a manipulação de informações de contexto segundo o modelo subjacente às ontologias criadas nessa linguagem.

Desta forma, para construção e processamento da ontologia do EXEHDA-ON propõem-se a utilização de tecnologias da Web Semântica. Fundamentado nos requisitos expostos no parágrafo anterior foram escolhidas as tecnologias descritas nos próximos itens.

OWL

A linguagem escolhida para construção do modelo ontológico do EXEHDA-ON foi a OWL (BECHHOFER et al., 2004), padrão para a Web Semântica, descrita em seus conceitos básicos no capítulo 4. Como sub-linguagem foi adotada a OWL-DL. Esta sub-linguagem corresponde à lógica de descrição e provê um maior grau de expressividade onde todas as conclusões são computáveis e todas as computações terminam em tempo finito.

A OWL fornece suporte a metadados RDF, abstrações de classes, generalização, agregação, relações de transitividade, simetria e detecção de inconsistências.

Jena

No EXEHDA-ON propõem-se o uso da API Java do *toolkit* Jena (MCBRIDE, 2007), por oferecer: (*i*) mecanismos para manipulação de modelos RDF em memória, bases de dados relacionais e arquivos; (*ii*) suporte às mais difundidas linguagens de consulta de dados RDF, como: SPARQL (*SPARQL Protocol And RDF Query Language*) (SEABORNE, 2007b); (*iii*) um conjunto de APIs para manipulação de ontologias codificadas em OWL; (*iv*) e máquinas de inferência baseadas em ontologias e regras, bem como um mecanismo que permite integrar máquinas de inferência de terceiros.

A implementação da API Jena é código aberto e foi desenvolvida no projeto *HP Labs Semantic Web Programme* da *Hewlett-Packard Development Company*. A API está dividida essencialmente 5 áreas relativas ao processamento de ontologias:

- processamento e manipulação de modelos RDF;
- implementação da linguagem de consulta SPARQL;
- processamento e manipulação de ontologias descritas em OWL;
- inferência sobre OWL;
- persistência de modelos de ontologias sobre bases de dados.

A arquitetura base da API de ontologias do Jena é composta por 3 camadas, conforme figura 5.9.

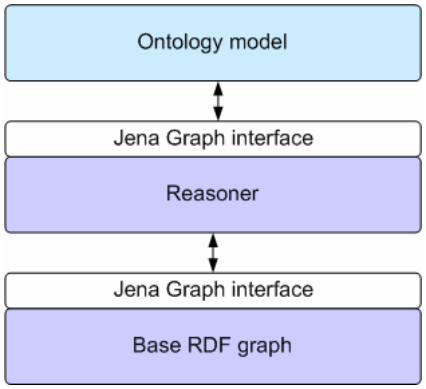

Figura 5.9: Arquitetura da API Jena
(MCBRIDE, 2007)

Existem 3 módulos base para o processamento de ontologias. Em todas as camadas existem *patterns* que são usados para permitir que a API seja mais genérica. O *Factory Pattern* é usado para a construção das classes principais para cada sub-módulo. O primeiro módulo é o *Ontology Model*, este contém todas as classes necessárias para trabalhar com ontologias descritas em OWL, DAML, OIL ou RDFS. Neste módulo a classe mais relevante é a *OntModel* que representa um modelo ontológico. Esta classe, no entanto, é uma interface, cabendo à *ModelFactory* encontrar a implementação apropriada para aquela interface. Neste caso é a *OntClassImp* que se encontra no *subpackage com.hp.hpl.jena.Ontology.Impl*. De uma forma genérica existe sempre (*i*) uma classe que representa um conceito (por exemplo: *OntModel*, *Reasoner*); (*ii*) uma *Factory* com métodos estáticos para construir instâncias dessa classe e (*iii*) um *Registry* que a *Factory* usa para localizar as implementação da superclasse que representa o conceito.

O outro módulo presente na arquitetura é o *Reasoner*. Este permite fazer inferências sobre modelos OWL. O uso das inferências sobre modelos semânticos permite obter informação adicional (inferida) sobre as ontologias. A estrutura geral deste módulo é apresentada na figura 5.10.

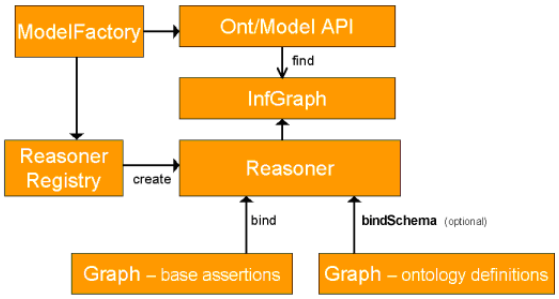

Figura 5.10: Arquitetura do *reasoner* da API Jena
(MCBRIDE, 2007)

A arquitetura do *Reasoner* possibilita que novos motores de inferência venham a ser adicionados, permitindo que estes motores possam ser tanto locais como remotos. A classe principal desta API é o *Reasoner*, a qual executa o processamento sobre as ontologias. Os *Reasoners* podem ser criados a partir da *ReasonerFactory*, que por sua vez usa o *ReasonerRegistry* para encontrar o *Reasoner* apropriado. É no *ReasonerRegistry* que novos motores de inferência podem ser adicionados. A operação mais básica que é usada no *Reasoner* é o *validate*, este método permite verificar se o modelo ontológico está de acordo com as regras de inferência especificadas no *Reasoner*.

Os *reasoners* de OWL da API Jena funcionam aplicando regras tipo *if-then-else* sobre instâncias OWL. O outro *reasoner* disponível é o de uso geral. O *Reasoner* Genérico Jena (RGJ) permite inferência sobre grafos RDF através de regras que podem ser definidas externamente. Estas regras são formadas basicamente por uma lista de termos de corpo (premissas), uma lista de termos de cabeça (conclusões) e um nome opcional que identifica a regra. Cada termo pode ser uma tripla padrão, uma tripla estendida padrão ou uma chamada a uma primitiva embutida (REYNOLDS, 2007). A sintaxe da linguagem de regra fornecida pela API Jena é mostrada na listagem 5.1.

Listagem 5.1: Sintaxe da linguagem de regra da API Jena (REYNOLDS, 2007)

```
Rule      :=   bare-rule .
          or   [ bare-rule ]
          or   [ ruleName : bare-rule ]

bare-rule :=   term, ... term -> hterm, ... hterm     // forward rule
          or   term, ... term <- term, ... term       // backward rule

hterm     :=   term
          or   [ bare-rule ]

term      :=   (node, node, node)           // triple pattern
          or   (node, node, functor)        // extended triple pattern
          or   builtin(node, ... node)      // invoke procedural primitive

functor   :=   functorName(node, ... node)  // structured literal

node      :=   uri-ref                          // e.g. http://foo.com/eg
          or   prefix:localname                 // e.g. rdf:type
          or   <uri-ref>                              // e.g. <myscheme:myuri>
          or   ?varname                       // variable
          or   'a literal'                    // a plain string literal
          or   'lex'^^typeURI                 // a typed literal, xsd:* type names
               supported
          or   number                         // e.g. 42 or 25.5
```

A classe principal do motor genérico é o *GenericRuleReasoner*. A comunicação entre os módulos é feita pela Jena *Graph Interface* que representa um modelo genérico de dados para comunicação entre módulos.

O terceiro módulo, *Base RDF Graph*, se justifica pelo fato do OWL estar definido sobre RDFS, logo o Jena usa as suas APIs de RDF para poder manipular as ontologias.

Utilizando a API Jena também é possível executar consultas sobre modelos RDF, através de linguagens de consulta. No EXEHDA-ON é utilizada a linguagem SPARQL, a qual é descrita na seção a seguir. As classes e métodos para manipulação de consultas fornecidas pela Jena API estão no *engine* de consultas *ARQ*.

SPARQL

SPARQL (SEABORNE, 2007b) é uma linguagem recomendada pelo W3C para realização de consultas em ontologias descritas em RDF e linguagens derivadas. É uma linguagem "orientada a dados", ou seja, permite apenas extrair dados de documentos RDF disponíveis em rede ou armazenados em um meio físico qualquer, não possuindo mecanismos de inferência.

Para que as consultas possam ser realizadas e as informações possam ser extraídas, a linguagem SPARQL disponibiliza uma sintaxe que, embora com algumas particularidades, funciona de maneira similar à linguagem SQL (*Structured Query Language*). Abaixo são apresentadas as principais cláusulas que a compõem:

- **prefix**: pode-se associar uma descrição para uma determinada URI (*Uniform Resource Identifier*). Deve-se colocar a *string* que define o prefixo, seguida de dois pontos ":" e da URI que deve ser mapeada a partir no prefixo criado;

- **select**: essa cláusula permite selecionar quais informações serão retornadas como resultado da consulta. As informações são armazenadas em variáveis que são identificadas pelo sinal de interrogação (?);

- **from**: essa cláusula é implícita, visto que a consulta é realizada sobre a ontologia ou modelo no qual se está trabalhando;
- **where**: com essa cláusula especifica-se as restrições que serão feitas para a realização da consulta. As restrições devem seguir o formato de tripla, que são formadas por um sujeito, um predicado e um objeto. Essas triplas podem ser formadas tanto por um objeto ou por um valor literal. Para os predicados, como forma de abreviação ou simplificação, pode ser utilizado o prefixo determinado na cláusula *prefix*;
- **filter**: possibilita a restrição do conjunto de soluções de acordo com a definição de expressões. As expressões podem ser funções e operações, sendo que os operandos dessas funções e operadores são um subconjunto dos tipos de dados do XML Schema (xsd:string, xsd:decimal, xsd:double, xsd:dateTime) e tipos derivados de xsd:decimal;
- **order by**: essa cláusula aplica condições de ordenação sobre o resultado da consulta. Uma condição de ordenação pode ser uma variável ou a chamada a uma função. Pode-se informar a direção da ordenação em ascendente ou decrescente, utilizando Asc e Desc;
- **limit**: com essa cláusula pode-se limitar o número de soluções retornadas com a execução da consulta.

Na seção 5.2.5 são apresentados exemplos de utilização de consultas SPARQL em casos de emprego do EXEHDA-ON.

5.2.2 Regras de inferência definidas para uso no EXEHDA-ON

A ontologia do contexto de interesse das aplicações (vide seção 5.1.3) é processada pelo serviço "Interpretador de contexto" do EXEHDA-ON (vide seção 5.1.6). Este processamento consiste na aplicação de regras de inferência que verificam a consistência das instâncias geradas a partir da subscrição de uma aplicação e, caso não existam inconsistências, permitem estender semanticamente a consulta construída. Esta consulta é então executada sobre a ontologia do ambiente pervasivo.

Assim, nesta seção é apresentado o processo de construção de regras de inferência previstas para utilização pelo EXEHDA-ON. Estas regras, escritas na linguagem de regras da API Jena, são avaliadas pelo *Reasoner* Genérico (vide item referente à API Jena na seção 5.2.1).

No EXEHDA-ON propõem-se a utilização de dois tipos de regras que são baseados nas informações descritas para construção de uma consulta e na estrutura do conhecimento modelado na ontologia do ambiente pervasivo. O primeiro tipo verifica a consistência dos dados que fazem parte do contexto de interesse registrado por uma aplicação, evitando a construção e execução de consultas sem coerência. Por sua vez, o segundo tipo de regras permite capturar o conhecimento modelado na ontologia do ambiente pervasivo para estender semanticamente a consulta. A criação destes tipos de regras está baseada nos trabalhos de (TANGMUNARUNKIT; DECKER; KESSELMAN, 2003) e (SILVA, 2006).

A seguir é apresentado um exemplo de cada tipo de regra. Nestes exemplos, os prefixos "onto-ci" e "onto-ap" correspondem, respectivamente, a ontologia do contexto

de interesse das aplicações e a ontologia do ambiente pervasivo, as quais estão descritas nas seções 5.1.2 e 5.1.3.

Exemplo de regra de consistência

Com relação ao primeiro tipo de regra, um exemplo de contexto de interesse informado por uma aplicação que seria inconsistente para construção de uma consulta, envolveria as seguintes propriedades da classe "Nodo" da ontologia do ambiente pervasivo: sistema de arquivos (SistArquivos) igual a "EXT3" e nome do sistema operacional (NomeSO) igual a "Windows". O conhecimento prévio a respeito destes requisitos indica a existência de uma inconsistência, visto que, os sistemas operacionais da família Windows não suportam o sistema de arquivos EXT3. Neste caso, o conhecimento prévio é expresso na regra de inferência apresentada na listagem 5.2, a qual é avaliada pelo *reasoner* da API Jena. Caso as premissas que expressam uma inconsistência na consulta sejam verdadeiras, é gerado, como conclusão da regra, uma mensagem informando a inconsistência. Assim, é evitada a continuidade da construção da consulta, sendo enviado ao consumidor uma notificação de inconsistência na solicitação.

Listagem 5.2: Regra para verificação de consistência

```
@prefix onto-ci: <http://exehda.ucpel.tche.br/exehda-on/ontoci.owl#>.
@prefix onto-ap: <http://exehda.ucpel.tche.br/exehda-on/percom.owl#>.

[ConsistenciaSisArqSisOp: (?A onto-ci:SistArquivos ?SA) (?B onto-ci:NomeSO ?SO) (?C rdf:
    type owl:Class) equal(?SA 'EXT3') equalExt(?C 'Windows') (?D rdf:type ?C) (?D onto-
    ap:NomeSO ?SO1) equalExt(?SO ?SO1) -> (?D 'Sistema Operacional e Sistema de Arquivos
     são incompatíveis' ) ]
```

Exemplo de regra de extensão

Um exemplo relacionado ao segundo tipo de regra, envolveria o interesse de uma aplicação em nodos com sistema operacional "Unix". Neste caso, a consulta derivada deste contexto de interesse poderia ser estendida semanticamente abrangendo também sistemas operacionais "Linux". O conceito de sistema operacional é representado na ontologia do ambiente pervasivo pela classe "SistemaOperacional". Esta classe é especializada nos seguintes tipos de sistemas operacionais: "Windows", "Unix" e "MacOS", sendo a classe "Unix" especializada na classe "Linux". Baseado no conhecimento modelado pelas relações de subclasse que os tipos de sistemas operacionais possuem, é possível estender essa consulta, retornando também nodos que tenham sistema operacional igual a "Linux". A listagem 5.3 mostra a regra de inferência que possibilita a extensão semântica da consulta referida neste exemplo.

Listagem 5.3: Regra para extensão semântica

```
@prefix onto-ci: <http://exehda.ucpel.tche.br/exehda-on/ontoci.owl#>.
@prefix onto-ap: <http://exehda.ucpel.tche.br/exehda-on/percom.owl#>.

[ExtensaoUnix: (?A onto-ci:TipoSO ?SO)(?B rdf:type owl:Class) equalExt(?B 'Unix')
    equalExt(?B ?SO) (?D rdf:type ?B) (?D onto-ap:TipoOS ?SO1) (?E rdfs:subClassOf ?B)
    equalExt(?E 'Linux') (?F rdf:type ?E) (?F onto-ap:TipoOS ?SO2) ->(?G onto-ci:
    Extensao ?SO1) (?H onto-ci:Extensao ?SO2) (?I onto-ci:Extensao 'Unix') (?F onto-ci:
    Extensao 'Linux') ]
```

5.2.3 Prototipação do servidor de contexto do EXEHDA-ON

Um protótipo do servidor de contexto para o ambiente pervasivo foi modelado e desenvolvido com o uso da API Jena e a linguagem de programação Java. Para testar o protótipo foram criados sensores para dados estáticos (arquitetura, número de núcleos, tipo de processador, número de interfaces de rede, memória física, disco instalado) e dinâmicos (processador disponível, memória disponível, disco disponível, tráfego de entrada na rede, tráfego de saída da rede).

Os dados sensorados são compostos em instâncias da ontologia do ambiente pervasivo no nodo (processamento de *strings*), as quais são enviadas ao servidor de contexto, localizado no EXEHDABase. Estas instâncias contêm as informações de identificação do nodo e os dados sensorados. No servidor de contexto é feita a aglutinação destas instâncias na ontologia do ambiente pervasivo na célula com a utilização da API Jena.

Com o intuito de abstrair aspectos de baixo nível relativos ao tratamento das comunicações em rede, na implementação do servidor de contexto foram utilizados os serviços WORB e CIB do EXEHDA. A classe principal do servidor de contexto dispara uma *thread* responsável por utilizar esses serviços de comunicação para aguardar as atualizações de contexto dos nodos.

O servidor de contexto mantém uma base de dados ontológica com informações sobre o estado do ambiente pervasivo. As classes construídas para o servidor de contexto são apresentadas na figura 5.11.

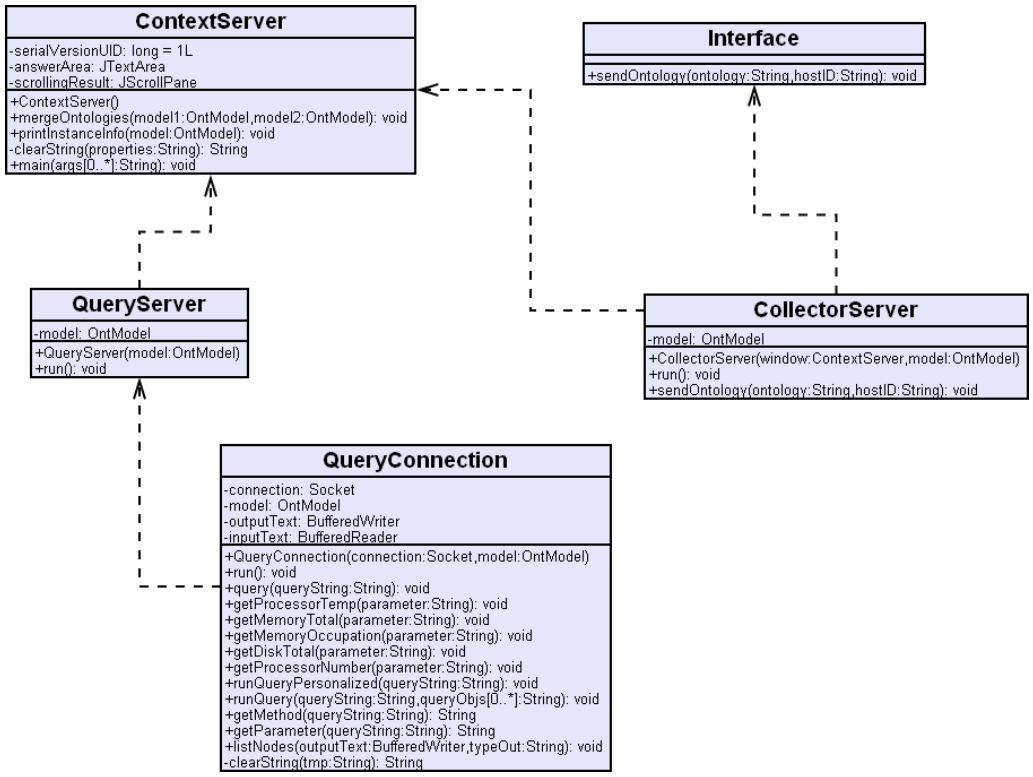

Figura 5.11: Classes do servidor de contexto do EXEHDA-ON

Todos os procedimentos relativos à manipulação da ontologia são realizados com o uso da API Jena. Quando a *thread* responsável pela comunicação recebe a ontologia do ambiente pervasivo atualizada de algum nodo ela a repassa para a classe principal

"*ContextServer*", a qual possui o método "*mergeOntologies*" responsável por verificar a ontologia recebida e realizar as atualizações necessárias na base de dados ontológica. O servidor é executado na base da célula, gravando as alterações ocorridas na ontologia dentro do arquivo *log* do EXEHDA.

Também, um serviço de consultas foi desenvolvido junto ao servidor de contexto. Este serviço possibilita a realização de pesquisas na ontologia do ambiente pervasivo utililizando a linguagem SPARQL. A linguagem de consulta SPARQL é utilizada através da biblioteca ARQ da API Jena. Os métodos da classe *QueryConnection* (vide figura 5.11) implementam as consultas em SPARQL que pesquisam propriedades dos nodos, tais como: temperatura do processador, memória total, tamanho do disco e número de processadores.

5.2.4 Definição de componentes de software do EXEHDA-ON

Os componentes de software foram definidos levando em consideração: (*i*) os serviços previstos para o EXEHDA-ON; (*ii*) arquitetura de software proposta para o EXEHDA-ON e (*iii*) os requisitos do servidor de contexto. Assim, foram projetados componentes de software que possuem funcionalidades que envolvem aspectos relacionados com a coleta das informações de contexto nos nodos, o envio das mesmas para atualização da base ontológica junto ao servidor de contexto e a execução de consultas à ontologia. A figura 5.12 mostra estes componentes e suas interações.

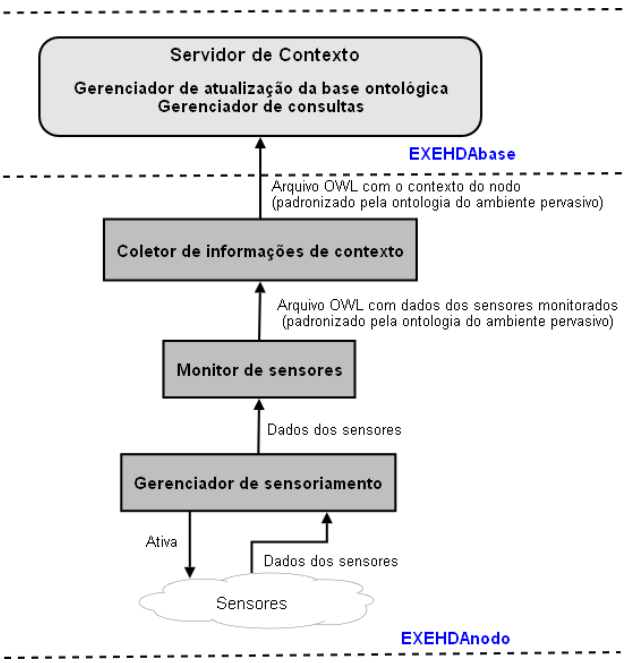

Figura 5.12: Gerenciamento das informações de contexto no EXEHDA-ON

Gerenciador de sensoriamento

Componente que realiza a ativação dos sensores responsáveis pela obtenção dos dados de contexto. Para execução de suas funcionalidades, este componente considera a divisão dos sensores em dois tipos: os que monitoram dados estáticos e os que monitoram dados dinâmicos.

Monitor de sensores

Os monitores são responsáveis por agrupar as informações de seus sensores, gerando *strings* em linguagem OWL no mesmo formato da ontologia do ambiente pervasivo. Este componente gera um arquivo OWL com as informações dos sensores e o envia para o componente "Coletor de informações de contexto". A listagem 5.4 mostra um exemplo do tratamento de *strings* realizado por este componente resultando em parte de um arquivo OWL.

Listagem 5.4: *Strings* em OWL no formato da ontologia do ambiente pervasivo

```
<CargaCPU rdf:datatype="http://www.w3.org/2001/XMLSchema#float">39.0</CargaCPU>
<TempProcessador rdf:datatype="http://www.w3.org/2001/XMLSchema#float">32.0</
    TempProcessador>
<OcupMemoria rdf:datatype="http://www.w3.org/2001/XMLSchema#float">415.0</
    OcupMemoria>
```

Coletor de informações de contexto

O componente "Coletor de informações de contexto" (vide 5.13) é responsável por enviar as instâncias da ontologia do ambiente pervasivo que representam o estado atual do nodo para o servidor de contexto. Ele agrupa as informações recebidas do componente "Monitor de sensores" em um determinado instante de tempo e as trata utilizando os serviços WORB e CIB do EXEHDA. A listagem 5.5 apresenta parte da ontologia de um nodo, mostrando os sensores agrupados e seus valores coletadas.

Listagem 5.5: Propriedades da ontologia do ambiente pervasivo, representando informações dos sensores de um nodo

```
<Fixo rdf:ID="Fixo_1">
  <NodoID rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</NodoID>
  <NumProcessadores rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</
      NumProcessadores>
  <TotDisco rdf:datatype="http://www.w3.org/2001/XMLSchema#float">80.0</TotDisco>
  <TotMemoria rdf:datatype="http://www.w3.org/2001/XMLSchema#int">512.0</TotMemoria>
  <CargaCPU rdf:datatype="http://www.w3.org/2001/XMLSchema#float">80.0</CargaCPU>
  <TempProcessador rdf:datatype="http://www.w3.org/2001/XMLSchema#float">23.0</
      TempProcessador>
  <OcupMemoria rdf:datatype="http://www.w3.org/2001/XMLSchema#float">415.0</
      OcupMemoria>
</Fixo>
```

Gerenciador de atualização da base ontológica

Este componente, integrado ao servidor de contexto junto ao EXEHDAbase, é responsável: (*i*) pelo recebimento das instâncias da ontologia do ambiente pervasivo que descrevem o contexto dos nodos, e (*ii*) pela atualização da ontologia que representa o contexto de uma célula pertencente ao ambiente pervasivo. A ontologia constitui a base ontológica da célula, a qual é acessada pelos serviços do EXEHDA-ON responsáveis por realizar a interpretação das informações de contexto representadas pela ontologia.

A atualização da base ontológica é decorrente de alterações nas informações de contexto nos nodos ativos na célula do ambiente pervasivo em um determinado instante de tempo, bem como ocorre em função da ativação e desativação de nodos na célula. Em ambas situações existe a necessidade de manipular a ontologia do ambiente pervasivo, seja pela alteração de valores em instâncias de nodos já existentes ou pela

agregação/desagregação de instâncias da ontologia que representam nodos que estejam ativos ou não no contexto celular.

O método "*mergeOntologies*" da classe "*ContextServer*" (vide figura 5.11) é responsável por este processo de atualização da ontologia, utilizando métodos da API Jena.

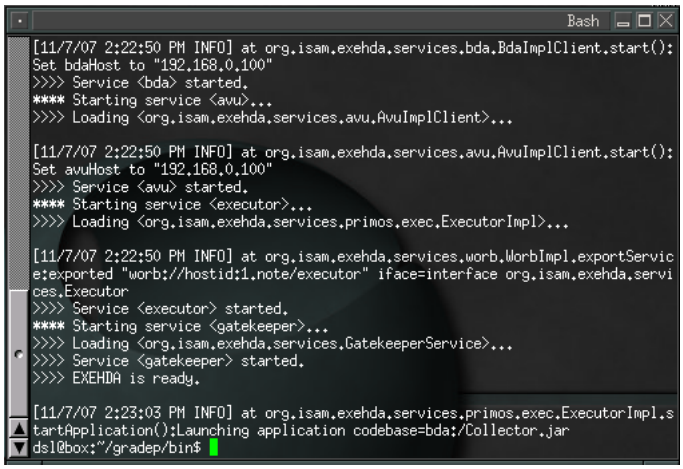

Figura 5.13: Coletor de informações de contexto em operação

Gerenciador de consultas

A ontologia do ambiente pervasivo descreve a infra-estrutura computacional que constitui uma célula e possui as instâncias que representam todos os nodos atualmente existentes no contexto da célula. Desta forma, a execução de consultas escritas em linguagem SPARQL sobre esta ontologia permite retornar informações de contexto que estão representadas pelas instâncias do modelo ontológico. A figura 5.14 mostra o Gerenciador de consultas do servidor de contexto em operação.

Assim, este componente do EXEHDA-ON, integrado ao servidor de contexto localizado no EXEHDAbase, tem as seguintes funcionalidades: (*i*) receber os parâmetros para as consultas; (*ii*) construir as consultas em linguagem SPARQL; (*iii*) executar as consultas sobre a ontologia do ambiente pervasivo e (*iv*) gerenciar o recebimento dos resultados.

Com exceção do "Gerenciador de atualização da base ontológica" e do "Gerenciador de consultas" , os demais componentes do EXEHDA-ON, acima descritos, estão integrados ao serviço Collector do EXEHDA instalado em cada nodo. Para obter as configurações da célula, tais como: endereço da base, nome do nodo, identificação do nodo e nome da célula, necessárias para o funcionamento do Collector, é utilizado o parser XML DOM (*Document Object Model*) com o intuito de acessar o arquivo XML que armazena as configurações do nodo.

5.2.5 Casos de emprego do EXEHDA-ON

Nesta seção são apresentados quatro casos de utilização do EXEHDA-ON. Neles são destacados o fluxo de processamento e os mecanismos envolvidos na operacionalização do EXEHDA-ON, conforme mostra a figura 5.15.

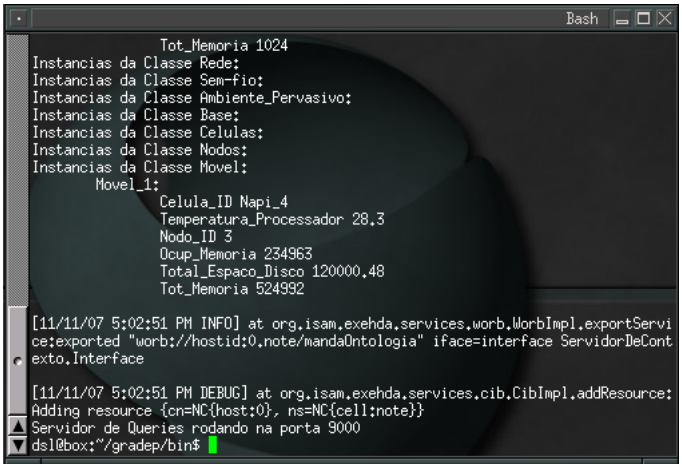

Figura 5.14: Gerenciador de consultas em operação

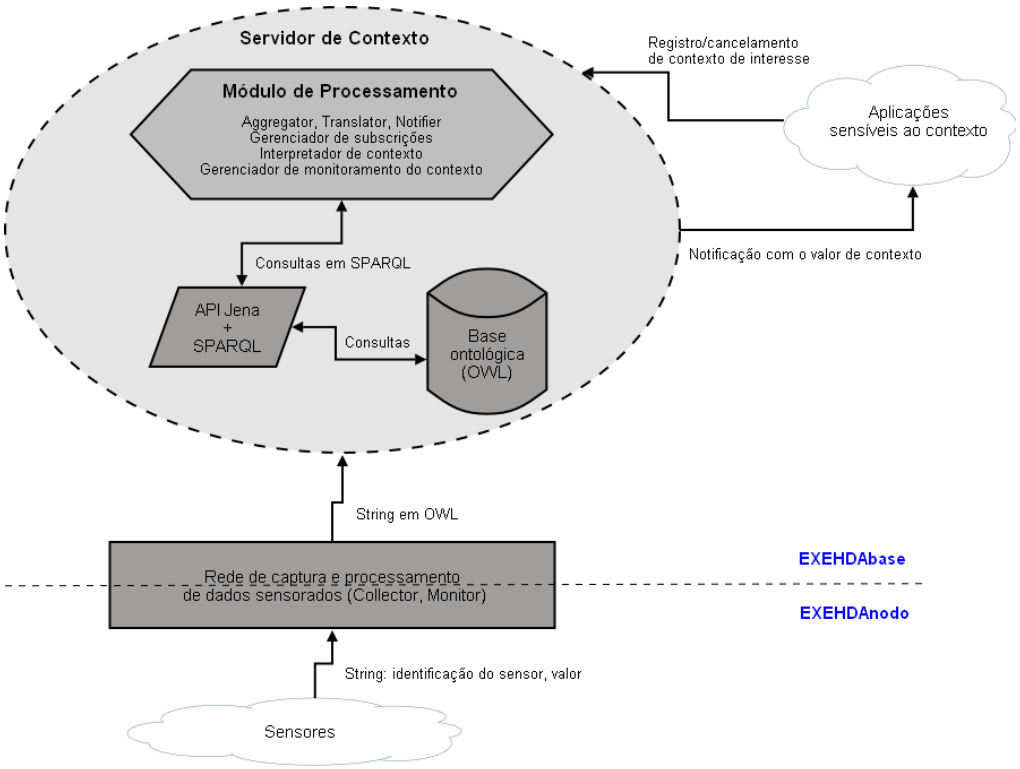

Figura 5.15: Fluxos de processamento e tecnologias associadas no EXEHDA-ON

5.2.5.1 Caso 1: Processamento de contexto baseado em arquivos OWL

Esta seção apresenta a manipulação de arquivos OWL gerados a partir do modelo ontológico proposto para o EXEHDA-ON. A implementação é baseada na parte do modelo que representa os sensores e os valores de seus objetos (memória total, memória livre, capacidade do disco rígido e usuário logado), relacionando esta classe aos nodos e à célula como um todo. Deste modo, o modelo ontológico pode ser tratado de forma parcial, destacando o aspecto de obtenção e disseminação das informações de contexto monitoradas pelos sensores. Com isso, demonstra-se a flexibilidade que o uso do modelo ontológico construído para o EXEHDA-ON pode proporcionar.

Com o intuito de representar as células do ambiente pervasivo, foram desenvolvidos componentes de software que geram e compõem arquivos OWL formados por diversas classes como pode ser visto na figura 5.2 e tabelas 5.1 e 5.2. Para demonstrar a possibilidade de manipulação parcial, os exemplos neste "Caso 1" consideram instâncias das classes "Nodo" e "Sensor" do modelo ontológico. Estes componentes de software, descritos nas seções seguintes, estão integrados tanto aos EXEHDAnodos, como ao EXEHDAbase.

Gerador de instâncias da ontologia

Este componente, integrado ao EXEHDAnodo, é responsável pela geração de instâncias da ontologia em linguagem OWL a partir de dados recebidos dos sensores. A instância correspondente a cada sensor contém as seguintes informações: identificação do sensor, identificação do nodo a que pertence e o valor sensoriado, ou seja, o valor absoluto enviado pelo sensor.

Um exemplo de ontologia gerada por este componente pode ser visualizada na listagem 5.6. O arquivo é gerado a partir de informações enviadas individualmente pelos sensores, sendo constituído pela classe *"Sensor"* e pela propriedade *"ValorSensor"*, além dos dois *prefix mappings*, respectivamente:

- *xmlns:NodoA="http://exehda.ucpel.tche.br/exehda-on/NodoA#"*, e
- *xmlns:exehda-on="http://exehda.ucpel.tche.br/exehda-on#"*

Listagem 5.6: Instância da ontologia que representa o sensor de memória do "NodoA"

```
<rdf:RDF
    xmlns:NodoA="http://exehda.ucpel.tche.br/exehda-on/NodoA#"
    xmlns:exehda-on="http://exehda.ucpel.tche.br/exehda-on#"
    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:daml="http://www.daml.org/2001/03/daml+oil#">
  <owl:Class rdf:about="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
  <owl:DatatypeProperty rdf:about="http://exehda.ucpel.tche.br/exehda-on#ValorSensor">
    <rdfs:domain rdf:resource="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  </owl:DatatypeProperty>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#MEM">
    <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
  </exehda-on:Sensor>
</rdf:RDF>
```

No exemplo acima, o *prefix mapping* denominado *"NodoA"*, será variável de acordo com o nome do nodo em que está sendo gerada a instância da ontologia. Por sua

vez, o segundo *prefix mapping* denominado *"exehda-on"*, será genérico, sendo utilizado por todas as instâncias geradas nos diversos nodos que constituem uma célula, indicando que todos os objetos (indivíduos) do tipo *"Sensor"* referem-se a uma mesma classe e a uma mesma propriedade. As instâncias da classe "Sensor" que correspondem aos indivíduos, serão nomeadas dinamicamente de acordo com o tipo de sensor que estarão descrevendo, assim como o valor a ser atribuído a este sensor.

Na linha 9 da listagem 5.6 pode-se observar a definição da classe "Sensor". Entre as linhas 10 e 13 é definida a propriedade "ValorSensor", que está sob domínio da classe "Sensor" e seu alcance configurado para o tipo "integer". Entre as linhas 14 e 16 é criado o objeto "MEM" do tipo "Sensor" (identificado pelo prefixo *"exehda-on:Sensor"*), com valor igual a "512". Assim, esta ontologia representa um sensor de memória (MEM), correspondente a uma instância da classe "Sensor", associada ao nodo denominado "NodoA", com valor 512 para a propriedade "ValorSensor".

Compositor de instâncias da ontologia do nodo

Este componente, integrado ao EXEHDAnodo, executa uma operação de leitura nos arquivos em formato OWL que contêm as instâncias da ontologia que descreve os sensores, aglutinando-as em uma única ontologia que representa o contexto do nodo em um determinado instante. Um exemplo de ontologia gerada por este componente pode ser visualizado na listagem 5.7.

Listagem 5.7: Ontologia que representa o contexto do "NodoA"

```
<rdf:RDF
    xmlns:NodoA="http://exehda.ucpel.tche.br/exehda-on/NodoA#"
    xmlns:exehda-on="http://exehda.ucpel.tche.br/exehda-on#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#">
  <owl:Class rdf:about="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
  <owl:DatatypeProperty rdf:about="http://exehda.ucpel.tche.br/exehda-on#ValorSensor">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
  </owl:DatatypeProperty>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#MEM_FREE">
    <exehda-on:ValorSensor>256</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#HD_FREE">
    <exehda-on:ValorSensor>30</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#MEM'>
    <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#USER">
    <exehda-on:ValorSensor>1</exehda-on:ValorSensor>
  </exehda-on:Sensor>
</rdf:RDF>
```

A estrutura básica deste arquivo OWL é igual a dos arquivos que representam a instância de um sensor, diferenciando-se pela união das várias informações dos sensores do nodo e seus respectivos valores. Assim, há a definição da classe "Sensor" e da propriedade "ValorSensor", bem como dos dois *prefix mappings (exehda-on: e NodoA:)*, seguida de várias instâncias da classe sensor, conforme pode-se ver na listagem 5.7.

Pode-se observar os vários objetos do tipo "Sensor" a partir da linha 13, cada um descrevendo um sensor distinto:

- ***MEM -*** descreve o total de memória disponível (valores expressos em MB);

- ***MEM_FREE -*** descreve o total de memória livre disponível (valores expressos em MB);

- ***HD_FREE -*** descreve o total de espaço livre no disco rígido (valores expressos em GB);

- ***USER -*** descreve a presença de usuário(s) no nodo (0 = não há usuário(s) logado(s), 1 = há usuário(s) logado(s)).

Desta forma, este documento OWL descreve um nodo com 512MB de memória, 256MB de memória livre, 30GB de disco rígido livre e a presença de usuário(s) logado(s).

Compositor da ontologia do ambiente pervasivo

Este componente, integrado ao EXEHDAbase, é responsável pelo recebimento dos arquivos OWL que descrevem o contexto dos nodos, aglutinando-os em uma ontologia que representa o contexto de uma célula pertencente ao ambiente pervasivo. Esta ontologia constitui a base ontológica da célula, servindo de base para os serviços do EXEHDA-ON responsáveis por realizar pesquisa sobre as informações de contexto representadas pela ontologia.

Um exemplo da ontologia do ambiente pervasivo gerada por este componente pode ser visualizada na listagem 5.8. Neste arquivo OWL pode-se identificar a descrição dos recursos computacionais dos vários nodos que compõem uma célula do ambiente pervasivo.

Listagem 5.8: Ontologia do ambiente pervasivo, representando uma célula

```
<rdf:RDF
    xmlns:NodoA="http://exehda.ucpel.tche.br/exehda-on/NodoA#"
    xmlns:exehda-on="http://exehda.ucpel.tche.br/exehda-on#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:NodoC="http://exehda.ucpel.tche.br/exehda-on/NodoC#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
    xmlns:NodoB="http://exehda.ucpel.tche.br/exehda-on/NodoB#">
  <owl:Class rdf:about="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
  <owl:DatatypeProperty rdf:about="http://exehda.ucpel.tche.br/exehda-on#ValorSensor">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
    <rdfs:domain rdf:resource="http://exehda.ucpel.tche.br/exehda-on#Sensor"/>
  </owl:DatatypeProperty>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoB#HD_FREE">
    <exehda-on:ValorSensor>20</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoB#MEM">
    <exehda-on:ValorSensor>1000</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#HD_FREE">
    <exehda-on:ValorSensor>30</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoB#USER">
    <exehda-on:ValorSensor>0</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoC#MEM_FREE">
    <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoC#HD_FREE">
    <exehda-on:ValorSensor>10</exehda-on:ValorSensor>
  </exehda-on:Sensor>
  <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#MEM_FREE">
    <exehda-on:ValorSensor>256</exehda-on:ValorSensor>
  </exehda-on:Sensor>
```

```
36    <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoB#MEM_FREE">
        <exehda-on:ValorSensor>610</exehda-on:ValorSensor>
38    </exehda-on:Sensor>
      <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoC#MEM">
40      <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
      </exehda-on:Sensor>
42    <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#MEM">
        <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
44    </exehda-on:Sensor>
      <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoC#USER">
46      <exehda-on:ValorSensor>0</exehda-on:ValorSensor>
      </exehda-on:Sensor>
48    <exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoA#USER">
        <exehda-on:ValorSensor>1</exehda-on:ValorSensor>
50    </exehda-on:Sensor>
    </rdf:RDF>
```

Pode-se observar que a estrutura básica dos arquivos OWL exibidos nas listagens anteriores (sensor e nodo) é mantida, sendo que a partir da linha 15 iniciam as instanciações de sensores com seus respectivos valores, cada qual com seu nodo de origem identificado com sua *URI* correspondente. Na listagem 5.9 encontra-se um exemplo:

Listagem 5.9: NodoC - Sensor de memória

```
<exehda-on:Sensor rdf:about="http://exehda.ucpel.tche.br/exehda-on/NodoC#MEM">
        <exehda-on:ValorSensor>512</exehda-on:ValorSensor>
</exehda-on:Sensor>
```

Observando a listagem 5.9 pode-se concluir que o indivíduo "MEM" do tipo "Sensor", pertence ao "NodoC", considerando a *URI* deste indivíduo "http://exehda.ucpel.tche.br/exehda-on/**NodoC**#MEM".

5.2.5.2 Caso 2: Processamento de contexto baseados em consultas SPARQL

Esta seção caracteriza consultas passíveis de serem executadas sobre a base ontológica gerada no "Caso 1" (vide listagem 5.8), com seus respectivos resultados.

Com o uso pelo EXEHDA-ON de uma linguagem para realização de consultas ao modelo ontológico, o processo de tradução dos dados sensorados para contextualizados no EXEHDA deixa de ser feito por algoritmos e estruturas de dados particulares para cada tipo de aplicação e passa a ser executado por uma linguagem de alto nível que independe da aplicação.

Consulta 1 - Exibir todos os recursos do NodoC

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER regex(str(?SUJEITO), "NodoC") }
```

Resultado da Consulta 1

```
SUJEITO                 | VALOR
============================
<NodoC#MEM_FREE>        | "512"
<NodoC#HD_FREE>         | "10"
<NodoC#MEM>             | "512"
<NodoC#USER>            | "0"
```

Consulta 2 - Nodos que possuem mais de 512MB de memória

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER (regex(str(?SUJEITO), "MEM$") && (?VALOR > 512)) }
```

Resultado da Consulta 2

SUJEITO	VALOR
<NodoB#MEM>	"1000"

Consulta 3 - Nodos que possuem mais de 256MB de memória livre

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER (regex(str(?SUJEITO), "MEM_FREE") && (?VALOR > 256))}
```

Resultado da Consulta 3

SUJEITO	VALOR
<NodoB#MEM_FREE>	"610"
<NodoC#MEM_FREE>	"512"

Consulta 4 - Nodos que possuem mais de 15GB de Disco Rígido Livre

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER (regex(str(?SUJEITO), "HD_FREE$") && (?VALOR > 15))}
```

Resultado da Consulta 4

SUJEITO	VALOR
<NodoA#HD_FREE>	"30"
<NodoB#HD_FREE>	"20"

Consulta 5 - Listar os nodos que possuem usuários logados

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER (regex(str(?SUJEITO), "USER$") && (?VALOR = 1))}
```

Resultado da Consulta 5

SUJEITO	VALOR
<NodoA#USER>	"1"

Consulta 6 - Listar os nodos que não possuem usuários logados

```
SELECT ?SUJEITO ?VALOR
WHERE { ?SUJEITO <http://exehda.ucpel.tche.br/exehda-on#ValorSensor> ?VALOR
FILTER (regex(str(?SUJEITO), "USER$") && (?VALOR = 0))}
```

Resultado da Consulta 6

```
SUJEITO          | VALOR
=====================
<NodoB#USER>  |  "0"
<NodoC#USER>  |  "0"
```

5.2.5.3 Caso 3: Inspeção do estado do contexto celular com o EXEHDA-ON

Para avaliação da integração das tecnologias no atendimento às demandas de prototipação do modelo foi desenvolvida uma aplicação que se vale do EXEHDA-ON para inspecionar o estado atual do contexto celular. Esta aplicação foi desenvolvida utilizando a linguagem Java e a API Jena.

O Inspetor disponibiliza um resumo do estado celular, indicando o número de nodos móveis e fixos que estão ativos. Também, permite ver informações detalhadas a respeito de um nodo selecionado. Estas informações abrangem:

- dados estáticos: arquitetura, disco instalado, memória física, número de interfaces de rede, número de núcleos;
- dados dinâmicos: ocupação de CPU, ocupação de memória, disco disponível, temperatura processador.

Este aplicativo utiliza as consultas SPARQL previamente definidas no componente de software "Gerenciador de consultas" do "Servidor de contexto do EXEHDA-ON", sendo possível visualizar o estado geral da célula (vide figura 5.16) e o estado específico de cada nodo (vide figura 5.17). A listagem 5.10 exibe um trecho de código Java relativo ao "Gerenciador de consultas" que destaca as consultas SPARQL previamente construídas para uso do Inspetor.

O fato de utilizar a linguagem SPARQL permite que as informações disponibilizadas pelo Inspetor possam ser ampliadas pelo acréscimo de novas consultas ao componente "Gerenciador de consultas", extraindo outras informações da base ontológica.

Listagem 5.10: Exemplos de consultas previamente definidas para uso do Inspetor

```
public void getTotalMemoria(String parametro){
  String queryString = "SELECT ?TotMemoria WHERE { ?nodo <http://exehda.ucpel.tche.br/
      exehda-on#NodoID> ?NodoID; <http://exehda.ucpel.tche.br/exehda-on#TotMemoria> ?
      TotMemoria FILTER (?NodoID = "+parametro+")}";
  String queryObjs[] = {"TotMemoria"};
  executorDeQuery(queryString, queryObjs);
}


public void getTotalDisco(String parametro){
  String queryString = "SELECT ?TotDisco WHERE { ?nodo <http://exehda.ucpel.tche.br/
      exehda-on#Nodo_ID> ?NodoID; <http://exehda.ucpel.tche.br/exehda-on#TotDisco> ?
      TotDisco FILTER (?Nodo_ID = "+parametro+")}";
  String queryObjs[] = {"TotDisco"};
  executorDeQuery(queryString, queryObjs);
}
```

5.2.5.4 Caso 4: Suporte à decisão para o escalonamento de recursos com o EXEHDA-ON

Na perspectiva deste "Caso 4", os nodos existentes em uma célula, em função do seu estado, são selecionados para computações paralelas numericamente intensivas.

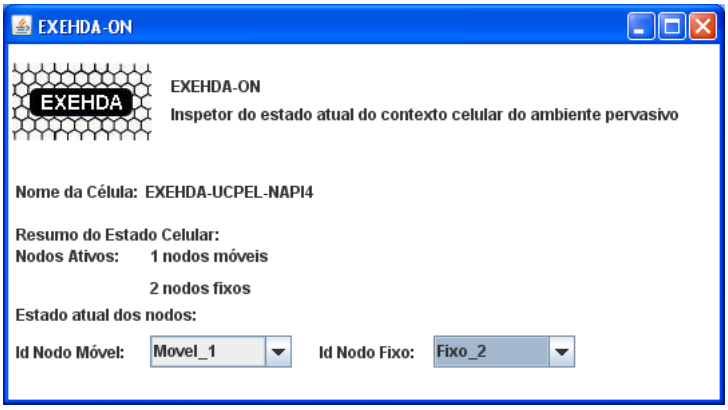

Figura 5.16: Interface principal da aplicação Inspetor

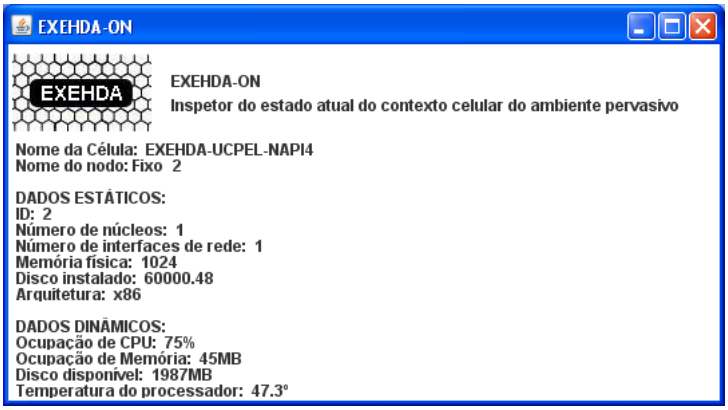

Figura 5.17: Detalhes de um nodo na aplicação Inspetor

Nesta seção é caracterizado o uso do EXEHDA-ON enquanto provedor de informações para a tomada de decisões por parte de um escalonador de recursos quando da gerência de uma execução paralela.

No EXEHDA, a escolha do nodo onde é instalado um componente de software (vide seção 4.5.2.1) denomina-se adaptação não-funcional. Neste tipo de adaptação a natureza da computação a ser realizada é mantida, independentemente do equipamento aonde o processamento será disparado.

Geração de contexto: carga sintética computacional

Para as avaliações neste "Caso 4", foi utilizado o "CpuSteal" (REAL, 2004), um gerador de cargas baseado em uma distribuição de probabilidades exponencial.

O "CpuSteal" opera em ciclos de ativação e desativação. Quando ativo, ele gera operações de ponto flutuante ocupando o processador, quando inativo ele entra em repouso. O controle do nível de ocupação média do processador é feito através da determinação de quanto tempo ele permanece ativo e quanto tempo ele fica em repouso em cada um dos ciclos.

Quando o "CpuSteal" é ativado, ele recebe como parâmetro o nome de um arquivo que descreve a carga a ser gerada, indicando os tempos de ativação e os tempos de repouso para cada ciclo de operação. A parametrização através de arquivo de dados permite que os experimentos sejam repetidos várias vezes, mantendo-se as mesmas características de carga. Nos experimentos apresentados nesta seção, o arquivo descritor de carga é gerado pelo programa "LoadGen", também desenvolvido para realizar a avaliação do EXEHDA-ON. O "LoadGen" gera os tempos de ativação e desativação baseado em uma distribuição de probabilidades exponencial.

Caracterização do hardware e software utilizado

Na avaliação do suporte do EXEHDA-ON à tomada de decisão para o escalonamento de recursos foi utilizado o *cluster* H3P da Escola de Informática da UCPEL (http://h3p.g3pd.ucpel.tche.br), composto de 10 nodos com sistema operacional Linux, cuja configuração parcial de hardware é apresentada na tabela 5.5. Estes computadores são interligados por uma rede *FastEthernet*.

As cargas computacionais ficaram distribuídas conforme tabela 5.6 e foram produzidas pelo gerador de cargas "CPUSteal" (vide seção 5.2.5.4).

O nodo "H3P" é o gerenciador do sistema, assim não será utilizado para o processamento. Nos nodos "H1", "H2" e "H3" não foi imposta carga gerada pelo "CPUSteal".

Construindo a tomada de decisão com o EXEHDA-ON

O processo de tomada de decisão é baseado no provimento de informações de contexto para o escalonador por parte do "Servidor de contexto do EXEHDA-ON". As consultas, realizadas na base ontológica quando ocorrem alterações nas informações de contexto, geram uma lista de nodos com as propriedades "poder computacional", "carga da CPU" e "componente OX (Objeto EXEHDA)" (YAMIN, 2004) organizadas de forma a atender a seleção prioritária dos nodos com (*i*) menor carga computacional, (*ii*) maior poder computacional e (*iii*) componente de software para cálculo já instalado pelo nodo mestre. Em relação a este último critério de seleção, o mesmo indica que este nodo trabalhador já foi utilizado previamente, portanto não há necessidade de instalação do

Tabela 5.5: Composição do *Cluster* H3P

Nodo	Freqüência(MHz)	Memória(MB)
H3P	1680	185
H1	2020	185
H2	1680	185
H3	2020	185
H4	2020	185
H5	2020	185
H6	2020	185
H7	2020	185
H8	1680	185
H9	1680	185

Tabela 5.6: Carga média dos nodos de processamento

Nodo	Carga (%)
H4	18.48
H5	18.82
H6	39.30
H7	40.26
H8	59.95
H9	61.45

código, apenas ativação da tarefa. A figura 5.18 ilustra o processo de tomada de decisão com o EXEHDA-ON.

Um cenário possível para uma consulta realizada sobre o modelo ontológico envolveria a definição de limites para a carga da CPU e poder computacional. Assim, na listagem 5.11 é exibida a consulta SPARQL que lista os nodos com carga igual ou inferior a 20% e com poder computacional maior ou igual a 3000 bogomips, também, esta consulta ordena os nodos em ordem crescente de carga, decrescente de poder computacional e decrescente da. A propriedade "CompOX" tem como objetivo indicar ao escalonador que o componente de software responsável pelo processamento já está instalado no nodo, logo em condições iguais de poder computacional e carga, o escalonador seleciona prioritariamente os nodos que já possuem instalado o componente de software responsável associado ao processamento desejado, neste "Caso 4" o "*WorkerPI*" refere-se ao componente de software que realiza computação do cálculo do π.

Listagem 5.11: Consulta 1: Nodos selecionados por carga e poder computacional e existência do OX

```
SELECT ?NODO ?CARGACPU ?PODERCOMP ?COMPOX
WHERE { ?NODO <http://exehda.ucpel.tche.br/exehda-on#PoderComp> ?PODERCOMP;
              <http://exehda.ucpel.tche.br/exehda-on#CargaCPU> ?CARGACPU;
              <http://exehda.ucpel.tche.br/exehda-on#CompOX> ?COMPOX
FILTER (?PODERCOMP >= 3000.0 && ?CARGACPU <= 20.0) }
ORDER BY ?CARGACPU DESC(?PODERCOMP) DESC(?COMPOX)
```

Resultado da Consulta 1

```
Nodo  | CargaCPU  | PoderComp  |  CompOX
=========================================
H3    |   0.00    | 3997.69    |  WorkerPI
H1    |   0.00    | 3997.69    |
H2    |   0.00    | 3325.95    |
H4    |  18.48    | 3997.69    |  WorkerPI
H5    |  18.82    | 3997.69    |
=========================================
```

Outro cenário para tomada de decisão envolveria uma consulta que retorna todos os nodos disponíveis organizados de forma crescente da carga da CPU e decrescente do seu poder computacional. Além de exibir o componente OX instalado, pelos motivos expostos no parágrafo anterior.

Listagem 5.12: Consulta 2: Todos os nodos disponíveis ordenados

```
SELECT ?NODO ?CARGACPU ?PODERCOMP ?COMPOX
WHERE { ?NODO <http://exehda.ucpel.tche.br/exehda-on#CargaCPU> ?CARGACPU;
              <http://exehda.ucpel.tche.br/exehda-on#PoderComp> ?PODERCOMP;
              <http://exehda.ucpel.tche.br/exehda-on#CompOX> ?COMPOX }
ORDER BY ?CARGACPU DESC(?PODERCOMP) DESC(?COMPOX)
```

Resultado da Consulta 2

```
Nodo  | CargaCPU  | PoderComp  | CompOX
=========================================
H3    |   0.00    | 3997.69    | WorkerPI
H1    |   0.00    | 3997.69    |
H2    |   0.00    | 3325.95    |
H4    |  18.48    | 3997.69    | WorkerPI
H5    |  18.82    | 3997.69    |
H6    |  39.30    | 3997.69    |
H7    |  40.26    | 3997.69    |
H8    |  59.95    | 3325.95    |
H9    |  61.45    | 3325.95    |
=========================================
```

Nas listagens anteriores o valor “0.00” na coluna “CargaCPU” indica que o nodo está livre, não tendo sido imposta uma carga gerada pelo “CPUSteal” (vide seção 5.2.5.4) a estes nodos.

Aplicação de teste desenvolvida

A aplicação prevista instala computações remotas em função da disponibilidade de recursos computacionais na célula de execução. A medida que surjam recursos que atendam os requisitos mínimos estabelecidos pelo desenvolvedor no âmbito do contexto celular o componente da aplicação responsável pela gerência da execução é notificado para que execute os procedimentos pertinentes.

A aplicação desenvolvida é do tipo sintético, e tem por objetivo central permitir a exploração das funcionalidades do EXEHDA-ON. Sua finalidade matemática é calcular o número π utilizando o método de Monte Carlo (PRESS et al., 1992), do ponto de vista computacional se trata de uma computação paralela do tipo *bag-of-tasks* cujo número de computações paralelas é determinado pelo programador. A seguir é apresentada uma descrição do método empregado nesta aplicação.

Considerando que A é a área de $\frac{1}{4}$ de círculo representada na figura 5.19, tem-se que:

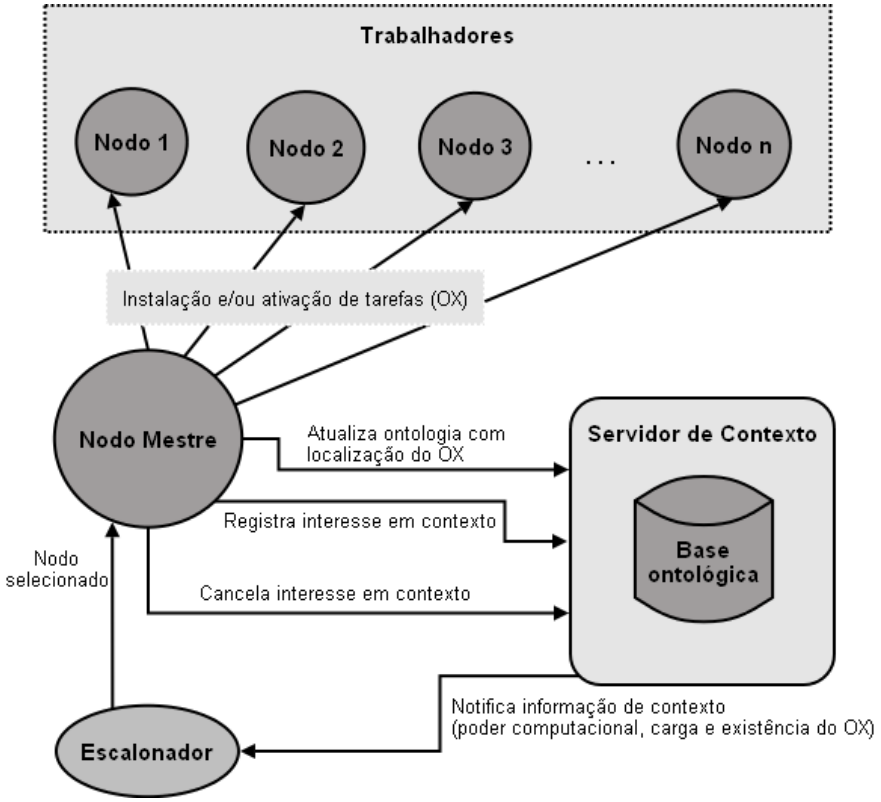

Figura 5.18: Tomada de decisão com EXEHDA-ON

$$A = \frac{1}{4} * \pi * r^2 \tag{5.1}$$

Pelo Método de Monte Carlo,

$$A = l^2 * \frac{P_{in}}{P_{total}} \tag{5.2}$$

sendo P_{in} os pontos lançados aleatoriamente que ficaram dentro da área em cinza, e P_{total} o número total de pontos gerados.

Unindo as equações 5.1, e 5.2 obtém-se:

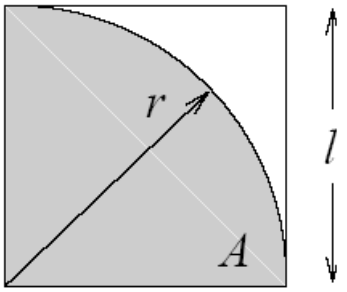

Figura 5.19: Área de $\frac{1}{4}$ de círculo

$$\frac{1}{4} * \pi * r^2 = l^2 * \frac{Pin}{Ptotal}$$
$$\pi = \frac{l^2 * \frac{P_{in}}{P_{total}}}{\frac{1}{4} * r^2}$$

Considerando-se $l = r = 1$ obtém-se:

$$\pi = 4 * \frac{P_{in}}{P_{total}} \tag{5.3}$$

O algoritmo de cálculo consiste em gerar pontos $P(x, y)$ sendo $0 \leq x \leq 1$ e $0 \leq y \leq 1$ (vide figura 5.20). Identificar e contar os pontos P_{in} e P_{total}, e aplicar a equação 5.3.

Entrada: número de iterações da execução em n
Resultado: valor estimado do π em PI

```
1: for P_total = 1 to n do
2:     x ← número aleatório entre 0 e 1
3:     y ← número aleatório entre 0 e 1
4:     if (x^2 + y^2 ≤ 1) then
5:         P_in ← P_in + 1
6:     end if
7: end for
8: PI ← 4 * P_in / P_total
```

Figura 5.20: Algoritmo de cálculo do π

Considerando que o processamento de cada uma das iterações pode ocorrer de forma independente, uma das maneiras de particionar o problema é dividir o número total de iterações desejadas entre vários nodos.

Resultados obtidos

O suporte à decisão para o escalonamento de recursos com o EXEHDA-ON foi avaliado através de quatro execuções da aplicação nos equipamentos do *cluster* H3P descritos na tabela 5.5. Parte dos nodos foi submetido a diferentes cargas sintéticas, através de um gerador programado de contexto de carga de processador, conforme mostra a tabela 5.6.

Em cada uma das quatro execuções foram selecionados cinco nodos, sendo que em três destas execuções os nodos foram escolhidos de forma aleatória (vide listagem 5.13). Em uma das execuções os nodos foram selecionados a partir de consulta SPARQL construída pelo EXEHDA-ON, considerando um critério de carga da CPU inferior a 70%, bem como uma ordem dos nodos crescente de carga da CPU e decrescente de poder computacional (vide listagem 5.14).

Observa-se que em função do resultado ordenado da consulta torna-se possível a seleção por parte do módulo Escalonador do número necessário para o processamento do cálculo, considerando as tarefas que faltam. Cabe ressaltar que caso o número de nodos atendendo ao critério fosse inferior a cinco, estes seriam selecionados e, o escalonador aguardaria novos nodos para dar continuidade ao restante do processamento. Quando o

número de nodos necessário para o processamento é atingido o nodo mestre cancela a subscrição do contexto de interesse.

Listagem 5.13: Nodos selecionados aleatoriamente

```
Execução 1: h1, h2, h4, h5, h7
Execução 2: h3, h5, h7, h8, h9
Execução 3: h1, h2, h3, h6, h8
```

Listagem 5.14: Nodos selecionados pelo EXEHDA-ON

```
Consulta SPARQL executada:

        SELECT ?NODO ?CARGACPU ?PODERCOMP
        WHERE { ?NODO <http://exehda.ucpel.tche.br/exehda-on#CargaCPU> ?CARGACPU;
                      <http://exehda.ucpel.tche.br/exehda-on#PoderComp> ?PODERCOMP
        FILTER (?CARGACPU <= 70.00)}
        ORDER BY ?CARGACPU DESC(?PODERCOMP)
_____________________________________________________________________

Resultado da consulta:

        Nodo   | CargaCPU   | PoderComp
        ==============================
        H1     |   0.00     | 3997.69
        H3     |   0.00     | 3997.69
        H2     |   0.00     | 3325.95
        H4     |  18.48     | 3997.69
        H5     |  18.82     | 3997.69
        H6     |  39.30     | 3997.69
        H7     |  40.26     | 3997.69
        H8     |  59.95     | 3325.95
        H9     |  61.45     | 3325.95
        ==============================

_____________________________________________________________________

Nodos selecionados (considerando que nenhuma tarefa foi iniciada):

        h1, h2, h3, h4, h5
```

O cálculo do π pelo método de Monte Carlo foi realizado com o lançamento de 10 bilhões de pontos. Este número total de iterações foi dividido entre os cinco nodos selecionados em cada uma das execuções. Na listagem 5.15 são mostrados os resultados obtidos. A forma de apresentação é a mesma registrada no *log* do EXEHDA.

Observando os resultados na listagem 5.15 pode-se verificar a precisão do impacto do gerador de carga no tempo de execução do módulo de cálculo. Por exemplo, os nodos h1 e h5 possuem o mesmo poder computacional, tendo sido imposto ao nodo h5 uma carga média de 18.82%. Analisando os tempos destes nodos, obtidos nas diferentes execuções apresentadas na listagem 5.15, pode-se observar diferenças em torno de 20%.

Os valores registrados no *log* tem a precedência definida pela ordem que ocorrem, o último a concluir a computação determina o tempo total de execução.

O principal objetivo destes casos de emprego foi exercitar o uso da arquitetura de software do EXEHDA-ON. Foram explorados os seguintes aspectos: (*i*) integração dos componentes de coleta de dados de contexto e de processamento das informações contextuais, (*ii*) emprego de uma linguagem de alto nível (SPARQL), (*iii*) atuação do servidor de contexto do EXEHDA-ON, o qual tem a API Jena como suporte para processamento de ontologias, e (*iv*) consistência do modelo ontológico concebido, enquanto representação do ambiente pervasivo.

Listagem 5.15: Resultados obtidos

```
Execução com nodos selecionados pelo EXEHDA-ON
...
O resultado de hostid:3.exehda-h3p foi 1570799271 em 812.632 segundos
O resultado de hostid:1.exehda-h3p foi 1570789020 em 813.025 segundos
O resultado de hostid:2.exehda-h3p foi 1570817787 em 983.396 segundos
O resultado de hostid:4.exehda-h3p foi 1570801460 em 991.103 segundos
O resultado de hostid:5.exehda-h3p foi 1570780264 em 995.219 segundos <-- Tempo total
Resultado do calculo do PI: 3.1415951208
...

Execução 1: nodos aleatórios
...
O resultado de hostid:1.exehda-h3p foi 1570801127 em 811.151 segundos
O resultado de hostid:4.exehda-h3p foi 1570781893 em 935.631 segundos
O resultado de hostid:2.exehda-h3p foi 1570806011 em 976.104 segundos
O resultado de hostid:5.exehda-h3p foi 1570804099 em 997.109 segundos
O resultado de hostid:7.exehda-h3p foi 1570797361 em 1221.303 segundos <-- Tempo total
Resultado do calculo do PI: 3.1415961964
...

Execução 2: nodos aleatórios
...
O resultado de hostid:3.exehda-h3p foi 1570827840 em 813.745 segundos
O resultado de hostid:5.exehda-h3p foi 1570820120 em 957.303 segundos
O resultado de hostid:7.exehda-h3p foi 1570771285 em 1192.303 segundos
O resultado de hostid:9.exehda-h3p foi 1570764426 em 1818.579 segundos
O resultado de hostid:8.exehda-h3p foi 1570813229 em 1850.741 segundos <-- Tempo total
Resultado do calculo do PI: 3.14159876
...

Execução 3: nodos aleatórios
...
O resultado de hostid:1.exehda-h3p foi 1570800575 em 812.077 segundos
O resultado de hostid:3.exehda-h3p foi 1570802723 em 814.551 segundos
O resultado de hostid:2.exehda-h3p foi 1570791423 em 971.851 segundos
O resultado de hostid:6.exehda-h3p foi 1570793860 em 1084.87 segundos
O resultado de hostid:8.exehda-h3p foi 1570801037 em 1830.695 segundos <-- Tempo total
Resultado do calculo do PI: 3.1415958472
...
```

5.3 Considerações finais

Neste capítulo foram apresentados aspectos de modelagem e prototipação do EXEHDA-ON, embasados nas discussões de temas centrais para a proposta. Estas discussões apresentadas nos capítulos 3 e 4, exploraram fundamentalmente a correlação existente entre Computação Pervasiva, Computação Sensível ao Contexto e ontologias. O capítulo 6 contém as principais conclusões deste trabalho, as contribuições da pesquisa, publicações realizadas e trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

A Computação Pervasiva é um novo paradigma que permite ao usuário o acesso a seu ambiente computacional independente de localização, tempo e equipamento. A Sociedade Brasileira de Computação a destaca como um dos grandes desafios da pesquisa na área da Ciência da Computação para os próximos dez anos. A sistematização dos esforços de pesquisa na área (vide capítulo 2) apontaram que este paradigma computacional 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. O 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 revisão dos projetos de pesquisa em Computação Pervasiva aponta que 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. Quanto maior a expressividade do modelo de informação do contexto, maior é a capacidade de representar a estrutura e a semântica dos conceitos. A análise dos métodos e tecnologias de *Web* Semântica (vide capítulo 4) caracterizam que quanto mais formal o modelo para descrição do contexto, maior é a capacidade de realização de pesquisa e inferência sobre o mesmo. Neste sentido, o uso de ontologias contribui para qualificar os mecanismos de sensibilidade ao contexto, em função da elevada expressividade que o uso destas pode propiciar.

A utilização de tecnologias de *Web* Semântica, em especial de ontologias, permite criar modelos formais, semânticos e extensíveis que descrevam o domínio das aplicações a partir da especificação de conceitos, relações e axiomas. O compartilhamento desses modelos promove a interoperabilidade semântica entre as aplicações e o EXEHDA-ON. A extensibilidade proporcionada pelo uso de ontologias permite ainda que mudanças no modelo acarretem alterações mínimas nas aplicações desenvolvidas. A especificação semântica das informações contextuais e do domínio das aplicações possibilita ainda a interpretação e inferência de novos fatos, baseados nas informações semânticas descritas nos modelos. Considerando essa premissa, desenvolvida ao longo do trabalho, a pesquisa realizada (vide capítulo 5) teve como foco explorar a correlação entre Computação Pervasiva, Computação Sensível ao Contexto e ontologias, contemplando uma proposta de

emprego de ontologias na busca da qualificação dos mecanismos utilizados para expressar informações de contexto. Como resultado, foram definidos os princípios e a concepção do EXEHDA-ON, um mecanismo para sensibilidade ao contexto, baseado em ontologias, direcionado ao *middleware* EXEHDA.

Atualmente, o processo de tradução dos dados sensorados para contextualizados no EXEHDA é feito por algoritmos e estruturas de dados particulares para cada tipo de aplicação. Neste sentido, a contribuição central da pesquisa EXEHDA-ON é minimizar a gerência desta personalização por parte do programador, tendo proposto a construção de um modelo ontológico que descreva semanticamente o estado atual do ambiente, utilizando um vocabulário comum e interpretável pelo servidor de contexto. Além disso, o fato de ser utilizado um modelo ontológico também torna possível a realização de pesquisas e inferências sobre o estado do ambiente pervasivo, com a utilização de uma linguagem de alto nível.

6.1 Principais conclusões

As atividades desenvolvidas ao longo deste trabalho, especialmente os estudos realizados, a construção do modelo ontológico e a prototipação e testes do EXEHDA-ON, permitiram a obtenção das seguintes conclusões:

- pesquisas envolvendo aspectos da sensibilidade ao contexto se mostram presentes nas diferentes propostas para Computação Pervasiva;
- os projetos de pesquisa em sensibilidade ao contexto mais atuais, discutem o uso de ontologias;
- o uso de ontologias se mostra oportuno para sensibilidade ao contexto, pois:
  - permite o compartilhamento e a representação do conhecimento em sistemas distribuídos dinâmicos e abertos;
  - provê significados para as informações contextuais, com sua semântica declarativa;
  - potencializa a interoperabilidade das entidades computacionais com o servidor de contexto;
  - possibilita que a interpretação de contexto seja realizada em alto nível.

Ainda, foi possível identificar uma qualificação dos mecanismos usados para obtenção e processamento de informações de contexto a partir dos estudos realizados, bem como do desenvolvimento da proposição do modelo ontológico e da arquitetura de software para o EXEHDA-ON.

Assim, o EXEHDA-ON atingiu seu objetivo de prover melhores níveis de descrição nas informações que caracterizam o contexto do ambiente pervasivo provido pelo EXEHDA, ficando indicado que o uso de ontologias e suas tecnologias associadas possibilita o emprego de uma semântica de maior expressividade que a usualmente praticada na coleta e no tratamento dos dados sensorados.

Na tabela 6.1, é feita uma comparação entre os principais projetos em Computação Sensível ao Contexto (vide capítulo 3) utilizados como referência para a definição deste trabalho, e a solução adotada no EXEHDA-ON.

Tabela 6.1: Comparação do EXEHDA-ON com projetos em Computação Sensível ao Contexto

Características	Outros projetos	EXEHDA-ON
Arquitetura	*CASS* tem uma arquitetura baseada em *middleware* centralizado. *CoBra* possui arquitetura baseada em agentes. A arquitetura da plataforma *Infraware* é constituída por um *middleware* baseado em *WebServices*	Considerando que o EXEHDA-ON foi concebido como extensão do *middleware* EXEHDA, ele mantém os aspectos arquiteturais do mesmo. Assim, sua arquitetura é baseada em serviços, cuja integração visa fornecer a infra-estrutura necessária para suporte à sensibilidade ao contexto em um ambiente pervasivo.
Modelagem de contexto	*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. A estrutura e o vocabulário da ontologia aplicada no *Context Managing Toolkit* são descritos em RDF. No Gaia o contexto é representado através de predicados escritos em DAML+OIL. Abordagens baseadas em ontologias escritas em OWL para modelagem de contexto são encontradas nos projetos *SOCAM*, *CoBrA* e *Infraware*.	A solução adotada no EXEHDA-ON para modelagem do contexto consiste no uso de ontologias desenvolvidas em OWL. O modelo baseado em ontologias é o mais promissor para a modelagem de contexto em ambientes pervasivos, conforme foi discutido no capítulo 3. As ontologias do EXEHDA-ON são próprias e descrevem o contexto de interesse das aplicações e o ambiente pervasivo. A linguagem OWL, escolhida para especificação das ontologias do EXEHDA-ON, é recomendada como um padrão de linguagem pela W3C, visto que, revisa e incorpora melhoramentos às características das demais linguagens, tais como: RDF e RDF *Schema* e DAML-OIL.
Processamento de contexto	O processamento do contexto nos projetos *CASS*, *CoBrA*, *Context Toolkit*, *CORTEX*, *SOCAM* e *Infraware* é baseado em motores de inferência que realizam a interpretação do contexto. Modelos de contexto não baseados em ontologias apresentam menor expressividade o que reduz a possibilidade de realização de inferências, sendo esta uma das limitações dos projetos CASS, *Context Toolkit* e CORTEX.	No EXEHDA-ON o processamento do contexto é realizado através de serviços e componentes de software que manipulam a base ontológica. As tarefas de interpretação de contexto incluem pesquisa e inferência sobre o modelo ontológico, utilizando regras de inferência e a linguagem de consulta SPARQL. O "Interpretador de contexto" é implementado com o uso da API Jena.

6.2 Contribuições da pesquisa

Considerando o foco da pesquisa EXEHDA-ON, destacam-se as seguintes contribuições:

- identificação das principais pesquisas relacionadas à sensibilidade ao contexto. Este estudo está no capítulo 3;

- sistematização das motivações para uso de ontologias no atendimento das demandas de processamento de contexto no cenário da Computação Pervasiva. Esta sistematização é apresentada no capítulo 4;

- concepção de um modelo ontológico do ambiente pervasivo para uso do EXEHDA-ON, a qual é tratada no capítulo 5;

- especificação da arquitetura de software do EXEHDA-ON, caracterizando os diversos serviços que compõem o servidor de contexto baseado em ontologias. A arquitetura de software do EXEHDA-ON é apresentada no capítulo 5;

- modelagem e desenvolvimento de um protótipo do servidor de contexto para o ambiente pervasivo. A prototipação deste servidor de contexto é apresentada no capítulo 5;

- desenvolvimento de casos de utilização do EXEHDA-ON, envolvendo (*i*) processamento de contexto baseado em arquivos OWL; (*ii*) processamento de contexto baseado em SPARQL; e (*iii*) suporte à decisão para o escalonamento de recursos. Estes casos de emprego do EXEHDA-ON são apresentados no capítulo 5;

- especificação e implementação de uma ferramenta gráfica para inspeção dos equipamentos que compõem as células de execução do EXEHDA. Esta ferramenta é descrita no capítulo 5;

- colaboração nos esforços de desenvolvimento do Projeto OPEN (*Ontology-Driven Pervasive Environment*) que transcorre no consórcio de pesquisa dos grupos G3PD e GPIA da UCPEL;

- divulgação dos resultados dos trabalhos realizados durante o desenvolvimento da dissertação através de publicações. As principais estão relacionadas na seção 6.3.

- disseminação no âmbito da UCPEL e da UFPel de conhecimento pertinente à área de sensibilidade ao contexto na Computação Pervasiva; pela participação na orientação de três trabalhos de conclusão de curso de graduação relacionados com a pesquisa desenvolvida para concepção do EXEHDA-ON;

- repasse do conhecimento e das tecnologias associadas ao EXEHDA-ON através de um *site* na Internet. A URL do site é http://exehda.ucpel.tche.br/exehda-on, sendo disponibilizado no mesmo, além de documentação pertinente, ferramentas de software, bem como códigos desenvolvidos.

6.3 Publicações realizadas

Ao longo de seu desenvolvimento, este trabalho teve seus resultados divulgados através de publicações. As principais estão relacionadas a seguir:

- **ERAD 2007**: LOPES, João L. B., YAMIN, Adenauer C., PILLA, Mauricio L. EXEHDA-ON: Uma Proposta Baseada em Ontologias Para a Sensibilidade ao Contexto na Computação Pervasiva In: 7ª Escola Regional de Alto Desempenho, 2007, Porto Alegre. ERAD 2007 - 7ª Escola Regional de Alto Desempenho. Porto Alegre: Evangraf, 2007. p.57 - 58

- **Iberian-American Conference on Technology Innovation and Strategic Areas - 2007**: LOPES, João L. B., PILLA, Mauricio L., YAMIN, Adenauer C. EXEHDA: a Middleware for Complex Heterogeneous and Distributed Applications In: Iberian-American Conference on Technology Innovation and Strategic Areas, 2007, Rio de Janeiro. Iberian-American Conference on Technology Innovation and Strategic Areas, 2007.

- **WPUC 2007**: LOPES, João L. B., PERNAS, Ana M., AFONSO, F., LIBRELOTTO, G., AUGUSTIN, Iara, PALAZZO, Luiz A. M., YAMIN, Adenauer C. Uma Abordagem Baseada em Ontologias para Sensibilidade ao Contexto na Computação Pervasiva In: I Workshop on Pervasive and Ubiquitous Computing, 2007, Gramado, RS. I Workshop on Pervasive and Ubiquitous Computing, 2007.

- **CLEI 2007**: FRAINER, Gustavo, SILVA, Luciano, AUGUSTIN, Iara, LOPES, João, YAMIN, Adenauer, REISER, Renata, GEYER, Cláudio. Towards Pervasive Applications in a Grid Computing Environment In: XXXIII Conferencia Latinoamericana de Informatica - CLEI 2007, 2007, San Jose. XXXIII Conferencia Latinoamericana de Informática, 2007.

- **CIC-UCPEL 2007**: LOPES, João L. B., PERNAS, Ana M., AFONSO, F., PALAZZO, Luiz A. M., YAMIN, Adenauer C. EXEHDA-ON: Uma Abordagem Baseada em Ontologias para Sensibilidade ao Contexto na Computação Pervasiva. Congresso de Iniciação Científica da Universidade Católica de Pelotas. Pelotas, RS. Mostra de Pós-Graduação, 2007.

- **Capítulo de Livro: Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications - 2008**: LOPES, João L. B., PALAZZO, Luiz A. M., YAMIN, Adenauer C., PILLA, Mauricio L., AUGUSTIN, Iara, GEYER, Claudio F. R. An Approach Based on Ontologies for Improving Context-Aware Usability in Pervasive Computing. Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications. IGI Global, 2008. (Aceito para publicação).

- **ERAD 2008**: LOPES, João L. B., AFONSO, Fernando A., PERNAS, Ana M., PALAZZO, Luiz A. M., YAMIN, Adenauer C. EXEHDA-ON: Uma Abordagem Baseada em Ontologias para Sensibilidade ao Contexto na Computação Pervasiva. In: 8ª Escola Regional de Alto Desempenho, 2008, Santa Cruz-RS. ERAD 2008 - Fórum de Pós-Graduação da 8ª Escola Regional de Alto Desempenho. (Aceito para publicação).

- **ERAD 2008**: AFONSO, Fernando A., LOPES, João L. B., PALAZZO, Luiz A. M., YAMIN, Adenauer C. Gradep-SC: Sensibilidade ao Contexto no Middleware GRADEp. In: 8ª Escola Regional de Alto Desempenho, 2008, Santa Cruz-RS. ERAD 2008 - Fórum de Iniciação Científica da 8ª Escola Regional de Alto Desempenho. (Aceito para publicação).

- **ERAD 2008**: NUNES, Lucas D., PERNAS, Ana M., LOPES, João L. B. Modelagem de um Sistema de Consulta de Informações de Contexto para Apoio a Adaptação de Aplicações Pervasivas. In: 8ª Escola Regional de Alto Desempenho, 2008, Santa Cruz-RS. ERAD 2008 - Fórum de Iniciação Científica da 8ª Escola Regional de Alto Desempenho. (Aceito para publicação).

6.4 Trabalhos futuros

Na perspectiva de dar continuidade a pesquisa desenvolvida durante a concepção do EXEHDA-ON, são apresentados a seguir aspectos a serem explorados em trabalhos futuros:

- analisar o desempenho do EXEHDA-ON considerando as diferentes possibilidades de modelagem ontológica do contexto;

- construir diferentes modelos ontológicos em função do domínio a ser tratado;

- expandir o mecanismo para construção de contextos que englobem várias células;

- ampliar a abrangência das regras de inferência utilizadas pelo serviço de interpretação de contexto.

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>. Acesso em junho de 2007.

AUGUSTIN, I. **Abstrações para uma Linguagem de Programação Visando Aplicações Móveis Conscientes do Contexto em um Ambiente de Pervasive Computing**. 2004. 193p. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre, RS.

AUGUSTIN, I.; YAMIN, A. C.; SILVA, L. C. da; REAL, R. A.; FRAINER, G.; GEYER, C. F. R. ISAMadapt: abstractions and tools for designing general-purpose pervasive applications: Experiences with Auto-adaptive and Reconfigurable Systems. **Softw. Pract. Exper.**, New York, NY, USA, v.36, n.11‐12, p.1231–1256, 2006.

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.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A Survey on Context-Aware Systems. **International Journal of Ad Hoc and Ubiquitous Computing**, [S.l.], v.2, n.4, p.263–277, 2007.

BECHHOFER, S.; HARMELEN, F. van; HENDLER, J.; HORROCKS, I.; MCGUINNESS, D. L.; PATEL-SCHNEIDER, P. F.; STEIN, L. A. **OWL Web Ontology Language Reference**. Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em junho de 2007.

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.

BRICKLEY, D.; GUHA, R. **RDF Vocabulary Description Language 1.0**: RDF Schema. Disponível em: <http://www.w3.org/TR/rdf-schema/>. Acesso em janeiro de 2008.

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.

BUSCHMANN, F. **Pattern-oriented software arquitecture** : a system of patterns. New York: [s.n.], 1996. John Wiley.

CHEN, G.; KOTZ, D. A Survey of Context-Aware Mobile Computing Research. **Technical Report TR2000-381**, 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.

CHEVERST, K.; MITCHELL, K.; DAVIES, N. Design of an object model for a context sensitive tourist GUIDE. **Computers and Graphics**, [S.l.], v.23, n.6, p.883–891, 1999.

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

CORCHO, O.; GÓMEZ-PÉRES, A. A roadmap to ontology specification languages. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE ENGINEERING AND KNOWLEDGE MANAGEMENT, 12., 2000. **Anais...** Springer-Verlag, 2000. p.80–96.

COSTA, P. D. **Towards a Services Platform for Context-Aware Applications**. Enschede, Netherlands: [s.n.], 2003.

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.

ENDEAVOUR. **The Endeavour Expedition**: charting the fluid information utility. Disponível em: <http://endeavour.cs.berkeley.edu/proposal/>. Acesso em outubro de 2007.

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).

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.

GEYER, C.; YAMIN, A.; SILVA, L. d.; VARGAS, P.; DUTRA, I.; PETEK, M.; ADAMATTI, D.; AUGUSTIN, I.; BARBOSA, J. Regional Center And Grid Development In Brazil. In: LAFEX INTERNATIONAL SCHOOL ON HICH ENERGY PHYSICS, LISHEP2002, 2002, Rio de Janeiro. **Anais...** Mérida: Universidad de Los Andes, 2002. p.[S.l.:s.n.].

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

GRADEL, 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.

GRUNINGER, M.; FOX, M. S. Methodology for the Design and Evaluation of Ontologies. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 14, 1995, Montreal, Quebec, Canada. **Anais...** [S.l.: s.n.], 1995. (IJCAI Proceedings 1995).

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.

HELAL, S. Programming Pervasive Spaces. **IEEE Pervasive Computing**, [S.l.], v.4, n.1, p.84–87, 2005.

HENRICKSEN, K.; INDULSKA, J. Developing context-aware pervasive computing applications: Models and approach. **Pervasive and Mobile Computing**, [S.l.], v.2, n.2, p.37–64, 2006.

HENRICKSEN, K.; INDULSKA, J.; RAKOTONIRAINY, A. Generating Context Management Infrastructure from Context Models. In: INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT (MDM), INDUSTRIAL TRACK PROCEEDINGS, 4., 2003, Melbourne. **Anais...** [S.l.: s.n.], 2003. p.1–6.

HESS, C. K.; ROMAN, M.; CAMPBELL, R. Building Applications for Ubiquitous Computing Environments. In: CONFERENCE ON PERVASIVE COMPUTING, 2002, Zurich, Switzerland. **Anais...** [S.l.: s.n.], 2002.

HIRTLE, D.; BOLEY, H.; GROSOF, B.; KIFER, M.; SINTEK, M.; TABET, S.; WAGNER, G. **Schema Specification of RuleML 0.91**. Disponível em: <http://www.ruleml.org/0.91/>. Acesso em janeiro de 2008.

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

HORROCKS, I.; PARSIA, B.; PATEL-SCHNEIDER, P.; HENDLER, J. SemanticWeb Architecture: Stack or Two Towers? In: PRINCIPLES AND PRACTICES OF SEMANTIC WEB REASONING, 14TH INTERNATIONAL CONFERENCE, 2005. **Anais...** Springer-Verlag, 2005. p.37–41.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; BOLEY, H.; TABET, S.; GROSOF, B.; DEAN, M. **SWRL**: A Semantic Web Rule Language Combining OWL and RuleML. Disponível em: <http://www.w3.org/submission/swrl/>. Acesso em janeiro de 2008.

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.

IBM. **IBM System Journal - Pervasive Computing**. New York: [s.n.], 1999. v.38, n.4.

INDULSKA, J.; ROBINSON, R.; RAKOTONIRAINY, A.; HENRICKSEN, K. Experiences in Using CC/PP in Context-Aware Systems. In: INTERNATIONAL CONFERENCE ON MOBILE DATA MANAGEMENT (MDM), 4., 2003. **Anais...** Springer, 2003. p.247–261. (Lecture Notes in Computer Science, v.2574).

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

ISAM. **Projeto ISAM - Infra-Estrutura de Suporte às Aplicações Móveis**. Disponível em: <http://www.inf.ufrgs.br/isam>. Acesso em maio de 2007.

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

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

KLYNE, G.; CARROLL, J. **Resource Description Framework (RDF)**: Concepts and Abstract Syntax. Disponível em: <http://www.w3.org/TR/rdf-concepts/>. Acesso em janeiro de 2008.

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

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 novembro de 2006.

LOPES, J. L.; PILLA, M. L.; YAMIN, A. C. EXEHDA: a Middleware for Complex, Heterogeneous and Distributed Applications. **Iberian-American Conference on Technology Innovation and Strategic Areas**, Rio de Janeiro, Brazil, Maio 2007.

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

MANOLA, F.; MILLER, E. **RDF Primer**. Disponível em: <http://www.w3.org/TR/REC-rdf-syntax/>. Acesso em janeiro de 2008.

MCBRIDE, B. **Jena API - A Semantic Web Framework for Java**. Disponível em: <http://jena.sourceforge.net/ontology/>. Acesso em novembro de 2007.

MCCARTHY, J.; BUVAC, S. Formalizing Context (Expanded Notes). In: WORKING PAPERS OF THE AAAI FALL SYMPOSIUM ON CONTEXT IN KNOWLEDGE REPRESENTATION AND NATURAL LANGUAGE, 1997, Menlo Park, California. **Anais...** American Association for Artificial Intelligence, 1997. p.99–135.

OPEN. **Projeto OPEN - Ontology-Driven Pervasive Environment**. Projeto submetido à PROPGE-UCPEL.

OXYGEN. **OXYGEN Project Home Page**. Disponível em: <http://oxygen.lcs.mit.edu>. Acesso em novembro de 2006.

PEREIRA FILHO, J. G.; PESSOA, R. M.; CALVI, C. Z.; OLIVEIRA, N. Q.; CARMO, R. R. M.; BARBOSA, A. C. P.; FARIAS, C. R. G.; LEITE, M. M. Infraware: um Middleware de Suporte a Aplicações Móveis Sensíveis ao Contexto. In: SBRC - SIMPóSIO BRASILEIRO DE REDES DE COMPUTADORES, 2006, Curitiba. **Anais...** [S.l.: s.n.], 2006.

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

PRESS, W. H.; FLANNERY, B. P.; TEUKOLSKY, S. A.; VETTERLING, W. T. **Numerical Recipes in C**: The Art of Scientific Computing. 2nd ed. Cambridge, UK: Cambridge University Press, 1992.

RAENTO, M.; OULASVIRTA, A.; PETIT, R.; TOIVONEN, H. ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications. **IEEE Pervasive Computing**, [S.l.], v.4, n.2, p.51–59, April-June 2005.

REAL, R. A. **TiPS, uma proposta de escalonamento direcionada à computação pervasiva**. Porto Alegre, RS: [s.n.], 2004. 115p.

REAL, R.; YAMIN, A.; AUGUSTIN, I.; BARBOSA, J.; SILVA, L.; FRAINER, G.; GEYER, C. Tratamento da incerteza no escalonamento de recursos em Pervasive Computing. In: CONFERÊNCIA IADIS IBERO-AMERICANA WWW/INTERNET, 2003, Algarve, Portugal. **Anais...** IADIS Press, 2003. p.167–170.

REYNOLDS, D. **Jena 2 Inference support**. Disponível em: <http://jena.sourceforge.net/inference/index.html>. Acesso em janeiro de 2008.

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.

SBC. **SBC - Grandes Desafios da Pesquisa em Computação no Brasil - 2006 - 2016**. Disponível em: http://www.sbc.org.br.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, 1994, Santa Cruz, CA, USA. **Anais...** [S.l.: s.n.], 1994. p.85–90.

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

SEABORNE, A. **RDQL - A Query Language for RDF**. Disponível em: <http://www.w3.org/Submission/RDQL/>. Acesso em junho de 2007.

SEABORNE, A. **SPARQL - A Query Language for RDF**. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. Acesso em novembro de 2007.

SILVA, A. P. C. **Mecanismo de Matching Semântico de Recursos Computacionais de Grids baseado na Integração Semântica de Múltiplas Ontologias**. Florianópolis, SC: [s.n.], 2006. 161p.

STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: WORKSHOP ON ADVANCED CONTEXT MODELLING, REASONING AND MANAGEMENT, UBICOMP 2004 - THE SIXTH INTERNATIONAL CONFERENCE ON UBIQUITOUS COMPUTING, 2004, Nottingham, England. **Anais...** UbiComp, 2004.

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

SU, X.; ILEBREKKE, L. A comparative study of ontology languages and tools. In: ADVANCED INFORMATION SYSTEMS ENGINEERING, 14TH INTERNATIONAL CONFERENCE, 2002. **Anais...** Springer-Verlag, 2002. p.761–765.

TANGMUNARUNKIT, H.; DECKER, S.; KESSELMAN, C. **Ontology-based Resource Matching in the Grid—The Grid meets the Semantic Web**.

USCHOLD, M.; KING, M. Towards a Methodology for Building Ontologies. In: INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 1995, Montréal, Québec Canada. **Anais...** [S.l.: s.n.], 1995.

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.

YAMIN, A. C.; AUGUSTIN, I.; BARBOSA, J.; SILVA, L. C. da; REAL, R. A.; FILHO, A. S.; GEYER, C. F. R. EXEHDA: Adaptive Middleware for Building a Pervasive Grid Environment. **Frontiers in Artificial Intelligence and Applications - Self-Organization and Autonomic Informatics**, [S.l.], v.135, p.203–219, 2005.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo