PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

INTRODUZINDO LOCALIZAÇÃO NA PLATAFORMA SEMANTICORE PARA A CRIAÇÃO DE APLICAÇÕES PERVASIVAS BASEADAS EM AGENTES DE SOFTWARE

LUCIANO ZAMPERETTI WOLSKI

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Marcelo Blois Ribeiro

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

W867i Wolski, Luciano Zamperetti
Introduzindo localização na plataforma SemantiCore para a criação de aplicações pervasivas baseadas em agentes de software / Luciano Zamperetti Wolski. – Porto Alegre, 2009.
84 p.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Marcelo Blois Ribeiro.

1. Informática. 2. Sistemas Multiagentes. 3. Computação Pervasiva. 4. Agentes de Software. I. Ribeiro, Marcelo Blois. II. Título.

CDD 006.39

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Introduzindo Localização na Plataforma SemantiCore para a Criação de Aplicações Pervasivas Baseadas em Agentes de Software", apresentada por Luciano Zamperetti Wolski, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sisitemas de Informação, aprovada em 21/12/09 pela Comissão Examinadora:

Prof. Dr. Marcelo Blois Ribeiro – Orientador — PPGCC/PUCRS

Prof. Dr. Fabiano Passuelo Hessel – — PPGCC/PUCRS

Prof. Dr. Rafael Heitor Bordini — UFRGS

Homologada em 27/04/10, conforme Ata No. 007 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Este trabalho é dedicado, minha mãe Alvenir, ao meu pai Luiz ao meu filho Lucas, minha irmã Elisane, ao João Vitor e a minha companheira de todas as horas Gilvana pelo carinho, companheirismo, e compreensão nesse trabalho.

AGRADECIMENTOS

Ao professor e orientador deste trabalho, Dr. Marcelo Blois Ribeiro, pela paciência, compreensão, contribuições e ensinamentos para o desenvolvimento deste trabalho que contribuíram muito para o meu crescimento profissional. Grato pela confiança depositada.

Aos professores membros da banca examinadora por terem aceitado o convite e por ajudar a enriquecer este trabalho com suas contribuições. Em especial ao professor Dr. Fabiano Passuelo Hessel pelas contribuições nas fases anteriores ao desenvolvimento deste trabalho.

Aos colegas do grupo de pesquisa ISEG que não mediram esforços em ajudar quando preciso, em especial aos colegas: Ana Paula Lemke, pelas correções e sugestões, ao Luís Henrique Leal Ries, pelas discussões, correções e material bibliográfico e ao Mauricio da Silva Escobar, pelas ferramentas e disposição em responder sobre os questionamentos sobre o SemantiCore

Aos professores do Programa de Pós-Graduação em Ciência da Computação da PUCRS: Avelino Francisco Zorzo; César De Rose, Duncan Dubugras Alcoba Ruiz; Fabiano Passuelo Hessel; Fernando Luís Dotti; João Batista Souza de Oliveira; Jorge Luis Nicolas Audy; Marcelo Blois Ribeiro; Milene Selbach Silveira; Ney Laert Vilar Calazans; Paulo Henrique Lemelle Fernandes; Toacy Cavalcante de Oliveira; Vera Lúcia Strube de Lima. Um salve especial ao Xiru Avelino, agradeço por ter acreditado no MINTER e se dedicado desde o inicio das negociações.

Aos funcionários do Programa de Pós-Graduação em Ciência da Computação da PUCRS, em especial ao Thiago Lingener sempre disposto e prestativo em atender os Minterianos.

A todos os colegas do MINTER, em especial aos Xirus: Armando, Alexandre, Diógenes, Everton, Fernando, Ivan, José Fernandes, Rodrigo e Uelinton. Espero, em breve, participar de novos desafios juntos.

Ao professor Dr. Flavio Teles, Coordenador do MINTER na UNEMAT, pelo incentivo na qualificação docente e apoio na execução do mestrado.

Ao funcionário da UNEMAT e também acadêmico do curso de Ciência da Computação da UNEMAT, Campus de Barra do Bugres, Marcio Gouvea Silva pela contribuição e tempo para realizar os inúmeros testes realizados neste trabalho.

À Fundação de Amparo a Pesquisa do Estado de Mato Grosso – FAPEMAT, financiadora deste MINTER, em especial ao presidente Antonio Carlos Camacho por ter acreditado e incentivado a qualificação dos profissionais da área tecnológica no Estado de Mato Grosso.

A tia Vera Wolski de Oliveira e primos Mario Vicente, Cristina, Rafael e Jalise, pela logística e acolhimento nesses dois anos de estudos em Porto Alegre. A todos "dziękuję bardzo".

A Deus, fonte de sustentação, minha eterna gratidão.

INTRODUZINDO LOCALIZAÇÃO NA PLATAFORMA SEMANTICORE PARA A CRIAÇÃO DE APLICAÇÕES PERVASIVAS BASEADAS EM AGENTES DE SOFTWARE

RESUMO

Este trabalho aborda duas tecnologias emergentes na área computacional: agentes de software e computação pervasiva. A computação pervasiva prevê que a computação esteja sempre disponível, em qualquer lugar, a qualquer tempo, e que o usuário possa usar qualquer dispositivo para ter acesso ao seu ambiente computacional. O crescente interesse da comunidade científica no uso da tecnologia de agentes para o desenvolvimento de aplicações para ambientes pervasivos é motivado por algumas propriedades dos agentes como autonomia, mobilidade e pró-atividade. Existem várias plataformas para o desenvolvimento de sistemas multiagentes (SMAs) como JADE [BEL07], Jason [BOR07] e SemantiCore [RIB04] e, nenhuma delas possui características para geração de aplicações pervasivas orientadas a agentes. A partir disso, faz sentido estendermos as plataformas para que incorporem características pervasivas as plataformas multiagentes usadas hoje em dia. Neste trabalho, veremos uma proposta de como tornar um ambiente de desenvolvimento multiagentes apto a criar aplicações pervasivas.

Palavras Chave: Agentes de Software, Sistemas Multiagentes, Computação Pervasiva.

INTRODUCING LOCATION IN SEMANTICORE PLATFORM FOR THE CREATION OF PERVASIVE APPLICATIONS BASED ON SOFTWARE AGENTS

ABSTRACT

This work presents two emergent technologies in the computational area: software agents and pervasive computing. Pervasive computing proposes that the computation is always available, at any place, at any time, and that the user could use any device to have access to his computational environment. The growing interest of the scientific community in the use of software agents for application development pervasive environments is related to the agent properties such as: autonomy, mobility and pro-activity. There are several platforms for the development of multi-agents systems (MAS) such as JADE [BEL07], Jason [BOR07] and SemantiCore [RIB04], but none of them provides the necessary infrastructure for pervasive agent-based application development, none of them has characteristics for generation of pervasive applications orientated to agents. In this sense, it is necessary to integrate pervasive characteristics in the agent platforms used nowadays. In this work, we propose the adaptation of a MAS environment order to create pervasive applications.

Keywords: Software agents, Multi-agents Systems, Pervasive Computing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AA – Advertisement Agent
ACA – Attention CockpitAgent
ACL – Agent Communicaton Language
AgDL – Agent Definition Language
AgML – Agent Modeling Language
AP – Access Point
API – Application Programming Interface
ASCII – American Standard Code for Information Interchange
AUML – Agent-based Unified Modeling Language
BDI – Belief-Desire-Intention
CA – Coordinator Agent
CA – Connector Agent
CAA – Context-Aware Agent
CIS – Context Information Service
CMA – Context Management Agent
CPA – Agente de Fornecimento de Contexto
DS – Discovery Service
FIPA – Foundation For Intelligent Physical Agents
GUI – Graphical User Interface
IP – Internet Protocol
ISEG – Intelligent Systems Engineering Group
JVM – Java Virtual Machine
LAC – Laboratory for Advanced Collaboration
LIS – Location Inference Service
LMA – Location Management Agent
MAC – Media Access Control
MART – MappedAgentRoutingTable
MIT – Massachusetts Institute of Technology
MJ – Memory Jog
MJA – Memory Jog Agent
OA – Ontology Agent
SCPe – SemantiCore Pervasivo
PDA – Personal Digital Assistent
PPGCC – Programa de Pós-Graduação em Ciência da Computação
PUCRS – Pontifícia Universidde Católica do Rio Grande do Sul
RA – Reasoner Agent

RDF – Resource Description Framework
RMI – Remote Method Invocation
RSSI – Received Signal Strength Indication
SKBA – System Knowledge Base Agent
SCPe – SemantiCore Pervasivo

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, percebe-se um considerável esforço na pesquisa em direção à computação pervasiva. Computação pervasiva é um termo praticamente novo na área da computação. Surgiu a partir de um artigo publicado por Mark Weiser em 1991, “The Computer for the 21st Century”, onde ele afirma que: “As mais impactantes tecnologias são aquelas que desaparecem”. A idéia que as tecnologias desaparecem vem da criação de ambientes com computação e comunicação de maneira integrada aos seres humanos, onde a percepção de se estar lidando com computadores seria mínima. Mark Weiser vislumbrou que, no futuro, computadores habitariam os mais triviais objetos (etiquetas de roupas, xícaras de café, interruptores de luz, canetas, etc) de forma invisível ao usuário. Neste mundo de Weiser, devemos aprender a conviver com computadores, e não apenas interagir com eles [WEI91]. Atualmente, vários *middlewares* para ambientes pervasivos que criam uma infraestrutura para o desenvolvimento de aplicações pervasivas, como MoCA [SAC04], Infraware [PES06] e One.World [GRI04].

Para que a computação seja pervasiva é preciso que uma aplicação tenha inteligência para processar a informação do contexto sobre o usuário e o ambiente, a fim de proporcionar ao usuário a informação certa no momento certo, e, possivelmente, agir em nome do usuário. Agentes de software podem auxiliar na construção destas aplicações proporcionando meios de controlar a distribuição inerente ao ambiente e, ao mesmo tempo, permitir um adequado mapeamento do conhecimento humano em aplicações de computador [MAE97].

Um agente de software é uma entidade de software que, a partir de informações percebidas no ambiente, captadas através da interação direta com outros agentes de software ou humanos, ou geradas a partir dos mecanismos dedutivos internos ao agente, atua em um ambiente buscando o alcance de seus objetivos [RIB02].

Para Pattie Maes [MAE97], agentes de software e computação pervasiva são tecnologias complementares. Estudos realizados em computação pervasiva pelo Media Lab do MIT (Massachusetts Institute of Technology) trabalham na fusão dessas duas tecnologias, no sentido de incorporar sensores, computação e habilidade de comunicação a objetos (celular, PDA, redes sem fio, etc) do nosso cotidiano. A computação pervasiva possibilita aos agentes ajudar os usuários com tarefas no mundo físico, assim como tarefas no mundo digital. Os agentes precisam dispor de informações do usuário, como preferências de cor de roupa, gastronomia, músicas, sites mais acessados, programas mais utilizados entre outras para ajudá-lo com uma série de tarefas. A fusão

dessas duas tecnologias surge como um modelo computacional para integrar de forma transparente os componentes de hardware e software para auxiliar o usuário em um ambiente.

Este trabalho propõe a análise e adaptação da arquitetura do SemantiCore [RIB04] para que ele possa ser usado para a construção de aplicações pervasivas baseadas em agentes juntamente com o *middleware* MoCA [SAC04]. O SemantiCore é um framework que visa promover uma camada de abstração sobre plataformas ou serviços de distribuição de computação que facilite a implementação de sistemas multiagentes, que atuem na Web [RIB04]. Já o *middleware* MoCA oferece suporte ao desenvolvimento de aplicações distribuídas sensíveis ao contexto que envolvem dispositivos móveis. A ideia central é a adaptação do SemantiCore com o *middleware* MoCA, para que ele proporcione a noção de localidade, permitindo que os agentes possam ter acesso a informações do ambiente físico em que estão situados.

1.1 Questão de pesquisa

Um SMA pode auxiliar na construção de aplicações pervasivas proporcionando meios de controlar a distribuição inerente ao ambiente e ao mesmo tempo permitir o fácil mapeamento do conhecimento humano em aplicações de computador.

Neste sentido, emerge a questão de pesquisa deste estudo: “É possível introduzir o gerenciamento de localização a plataforma SemantiCore de forma a permitir a criação de SMAs que utilizam-se de agentes rodando em diversos dispositivos com gerenciamento uniforme e transparente ao usuário?”.

1.2 Objetivos

Uma vez definida a questão de pesquisa, definiu-se o objetivo geral e os objetivos específicos deste trabalho, os quais são apresentados a seguir.

1.2.1 Objetivo Geral

Propor e aplicar um modelo que integre um middleware pervasivo sobre a plataforma SemantiCore de forma a permitir a criação de SMAs que utilizam-se de agentes rodando em diversos dispositivos com gerenciamento uniforme e transparente ao usuário.

1.2.2 Objetivos Específicos

- Aprofundar o estudo teórico sobre trabalhos relacionados ao problema abordado.

- Levantar a arquitetura atual do SemantiCore.

- Analisar a possibilidade de uso dos middlewares disponíveis para agregar serviços pervasivos ao SemantiCore.

- Propor a adaptação da arquitetura do SemantiCore para trabalhar com aplicações pervasivas.

- Implementar as alterações, gerando um novo protótipo do SemantiCore com serviços pervasivos.

- Avaliar os resultados.

1.3 Estrutura da Dissertação

O presente trabalho encontra-se estruturado em três partes: fundamentação teórica, a apresentação da proposta e a demonstração da aplicação desenvolvida. O Capítulo 2 corresponde à fundamentação teórica do trabalho, fazendo abordagem as duas tecnologias utilizadas para o desenvolvimento deste trabalho, os agentes de software e a computação pervasiva.

No Capítulo 3 apresentamos alguns trabalhos encontrados na literatura que utilizam *middlewares* pervasivos juntamente com plataforma multiagentes para o desenvolvimento de aplicações. Destaque para o desenvolvimento de aplicações sensíveis ao contexto utilizando sistemas multiagentes.

No capítulo 4 descrevemos sobre o *middleware* pervasivo e a plataforma multiagente utilizadas para o desenvolvimento de uma plataforma que integre essas duas tecnologias. Com destaque para a principal abordagem do trabalho, localização dos agentes levando em consideração as informações de contexto dos dispositivos móveis.

O Capítulo 5 descreve a implementação realizada sobre a plataforma do SemantiCore para a inserção de localização aos agentes inseridos em um determinado ambiente. Desenvolvemos um cenário para testarmos o protótipo criado juntamente com um aplicativo simples como exemplo de uso.

Por fim, no Capítulo 6 apresentamos as conclusões e os trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Agentes de software

Weiss [WEI99] define um agente de software como um sistema de computador situado em um ambiente e capaz de agir de forma autônoma para atingir um objetivo, onde a autonomia refere-se à capacidade de agir de acordo com sua própria linha de controle. Segundo Russel e Norvig [RUS04], um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores.

Wooldridge e Jennings [WOO95] afirmam que agentes são sistemas que apresentam um comportamento determinado por um processo de raciocínio baseado na representação de suas atitudes, tais como crenças, comprometimentos e desejos. Eles acreditam que um sistema pode ser visto como um agente se ele possuir as seguintes propriedades:

- Autonomia: o agente deve funcionar sem intervenção humana, baseando suas ações em seu conhecimento armazenado sobre o ambiente;

- Habilidade social: o agente interage com outros agentes através de linguagem comum;

- Reatividade: o agente deve ser capaz de perceber mudanças em seu ambiente e atuar de acordo com essas mudanças;

- Pró-atividade: o agente não deve apenas atuar por percepção, mas deve procurar alcançar uma meta, apresentando iniciativa.

Maes [MAE95] afirma que um agente ideal sabe qual é o seu objetivo e irá se esforçar para alcançá-lo. Um agente também deve ser robusto e adaptável, capaz de aprender com a experiência e reagir a situações imprevistas com um repertório de diferentes métodos. Finalmente, deve ser autônomo de forma que possa sentir o estado atual de seu ambiente e agir de forma independente para avançar em direção ao seu objetivo.

Para a construção de sistemas complexos é interessante considerar a utilização de vários agentes que desempenham tarefas voltadas à obtenção de seus objetivos e que estão de acordo com os objetivos de todo o sistema. Um sistema que possui vários agentes atuando em um ambiente em busca de seus objetivos é denominado sistema multiagentes [WEI99].

2.2 Sistemas Multiagentes

Geralmente, um agente de software, não é encontrado sozinho em um sistema ou aplicação, mas em conjunto com outros agentes. Esses agentes podem ser de mesmo tipo ou de tipos diferentes, formando uma sociedade denominada de sistema multiagentes.

Para [SYC98], o estudo de sistemas multiagentes (SMA) centra-se em sistemas nos quais muitos agentes inteligentes interagem uns com os outros. Os agentes são considerados como entidades autônomas, tais como programas ou robôs. Suas interações podem ser cooperativas ou competitivas. Isto é, os agentes podem compartilhar um objetivo comum (por exemplo, como numa colônia de formiga), ou eles podem perseguir os seus próprios interesses (como na economia de livre mercado).

Segundo [SYC98], as características em SMAs são: (1) cada agente tem informação incompleta ou capacidades para resolver o problema e, portanto, tem uma visão limitada; (2) não existe qualquer sistema de controle global; (3) os dados são descentralizados; (4) computação é assíncrona.

Para [BRE98], SMA caracterizam-se pela existência de uma certa quantidade de agentes autônomos, heterogêneos e independentes, trabalhando em conjunto para resolver um problema. Esses agentes são aptos a se adaptarem ao ambiente em que atuam, reagirem a ele e provocarem mudanças neste meio. Pode-se dizer então que um agente de software funciona continuamente, e é capaz de comunicar-se com outros agentes e cooperar com eles, e ainda mover-se de um ambiente para outro.

As características dos SMA se encaixam perfeitamente às necessidades encontradas no desenvolvimento de sistemas complexos. Primeiro, a autonomia dos componentes da aplicação (ou seja, a habilidade de um agente em decidir que ação irá tomar em determinado momento) reflete a natureza descentralizada dos sistemas distribuídos modernos. Segundo, o modo flexível com o qual os agentes operam (tendo um comportamento diferenciado a cada resposta do ambiente, por exemplo) se encaixa nas situações dinâmicas e imprevisíveis no que se espera atualmente em um sistema. Finalmente, a natureza dinâmica presente nas interações de um SMA é apropriada para sistemas abertos, onde seus componentes e suas interações estão em constante mudança [ZAM03].

2.2.1 Linguagens de modelagem

Os SMAs, tanto no meio acadêmico, quanto na engenharia de software, ganham cada vez mais espaço como um paradigma para o desenvolvimento e criação de sistemas de software [LIN01]. Com isso, novas tecnologias, métodos, linguagens de modelagem, plataformas de desenvolvimento, ferramentas e linguagens de programação estão sendo propostas. Para o desenvolvimento de sistemas baseados em agentes, é preciso técnicas adequadas que explorem seus benefícios e características próprias [SIL03].

Para que um novo paradigma tenha sucesso e seja difundido, é preciso uma linguagem de modelagem, aliada a outras tecnologias baseadas em agentes, que explorem o uso de abstrações relacionadas a agentes e promovam o refinamento dos modelos de design para código [SIL04]. A linguagem UML (*Unified Modeling Language*), utilizada para modelar sistemas orientados a objetos, pode ser utilizada como base para o desenvolvimento de SMAs, já que em SMAs agentes e objetos coexistem [CAI02]. No entanto, a UML não possui suporte suficiente para modelar SMAs, pois não possui suporte à modelagem de agentes, papéis de agentes e organização [SIL04].

Destacamos aqui duas linguagens utilizadas para a modelagem de agentes, a AUML [ODE00] e o ANote [CHO04].

A AUML (*Agent Unified Modeling Language*) [ODE00] é uma linguagem estendida da UML para modelar sistemas orientados a agentes. A AUML propõe e utiliza para representar as interações entre agentes os diagramas de sequência, colaboração, atividades e estados.

O ANote [CHO04] é uma linguagem de modelagem visual que tem como finalidade descrever os conceitos relacionados ao processo de modelagem de SMAs. Os desenvolvedores podem especificar objetivos, agentes, contextos, ontologias e comportamento que engloba soluções com agentes [CHO04]. O agente é o principal elemento de modelagem do ANote, fornecendo um conjunto de visões para especificação de um SMA.

2.2.2 Metodologias de desenvolvimento de agentes

A construção de sistemas multiagentes não é uma tarefa fácil, é preciso uma metodologia para gerenciar a complexidade de desenvolvimento, manutenção e distribuição desses sistemas.

Entende-se por metodologia uma série recomendada de passos e procedimentos, que compõem os métodos que devem ser seguidos durante o processo de concepção do software.

As metodologias orientadas a agentes devem prover abstrações adequadas e ferramentas para modelar tarefas individuais e sociais dos agentes. Várias metodologias para o desenvolvimento de sistemas multiagentes tem sido criadas como: Tropos [MYL01], Prometheus [PAD02] e MaSE [DEL99].

2.2.2.1 Tropos

O Tropos (*Requirement-Driven Development for Agent Software*) é uma metodologia que parte da premissa que, para se construir um software que opera dentro de um ambiente dinâmico, é preciso analisar e modelar o ambiente levando-se em consideração os atores, os objetivos e as dependências com outros atores [MYL01]. Segundo Mylopoulos, Tropos suporta quatro fases de desenvolvimento de software:

- Requisitos iniciais: compreensão do problema através do estudo da estrutura organizacional (atores organizacionais).

- Requisitos finais: modelo de dependência estratégica e o modelo de razão estratégica do sistema (ator do sistema).

- Projeto arquitetônico: é definida a arquitetura global do sistema em termos de subsistemas, interconectados através de fluxos de controle de dados (atores subsistema).

- Projeto detalhado: usa versões estendidas de UML, como a linguagem AUML (agentes).

2.2.2.2 Prometheus

A metodologia Prometheus [PAD02] suporta o desenvolvimento de agentes inteligentes com base na arquitetura BDI (crenças, desejos e intenções). Tem sido utilizada tanto no meio acadêmico quanto no meio industrial, pois suporta todo o processo de desenvolvimento desde a especificação até o teste.

Segundo Padgham, Prometheus consiste de três fases:

- Fase de Especificação do Sistema: identifica as funcionalidades básicas do sistema juntamente com os dados de entrada (percepções), de saída (ações) e qualquer informação importante de troca de dados.

- Fase de Design Arquitetural: utiliza as saídas da fase anterior para identificar os agentes e como eles vão interagir.

- Fase de Design Detalhado: Detalha a parte interna do agente e define como o agente irá cumprir suas tarefas.

2.2.2.3 MaSE

A metodologia MaSE (*Multiagent Systems Engineering*) [DEL99] possui como base o paradigma da orientação a objetos, com modelos semelhantes a UML. Utiliza duas linguagens para descrever agentes e sistemas multiagentes: a AgML (*Agent Modeling Language*) e AgDL (*Agent Definition Language*). A AgML é uma linguagem gráfica que define os tipos de agentes no sistema e suas ligações com outros agentes. Enquanto que a linguagem AgDL tem como base a lógica de predicados de primeira ordem para descrever o comportamento interno de cada agente individualmente.

A metodologia está estruturada em duas fases de desenvolvimento, a fase da Análise produz um conjunto de papéis cujas tarefas descrevem o que o sistema deve fazer para satisfazer seus requisitos e a fase do Projeto cria as classes de agentes, constrói conversas entre os agentes, agrupa classes de agentes e projeto do sistema.

2.2.3 Plataformas de desenvolvimento

Atualmente, existem diversas plataformas de desenvolvimento de SMAs que auxiliam na construção de aplicações que utilizam agentes, podemos destacar as plataformas: Jason [BOR07], JADE [JAD08] e SemantiCore [RIB04].

O SemantiCore é a plataforma escolhida para o desenvolvimento do nosso trabalho, será visto mais detalhadamente no capítulo 4. Para que nossa proposta tenha sucesso precisamos

incorporar características pervasivas ao SemantiCore para que o desenvolvedor consiga criar aplicações multiagentes com características pervasivas.

2.2.3.1 Jason

O Jason [BOR07] é um interpretador multi-plataforma que implementa *AgentSpeak*, utilizando uma linguagem abstrata baseada na arquitetura BDI (*beliefs, desires and intentions*), traduzindo para o português, crenças, desejos e intenções.

Segundo [BOR03], a linguagem *AgentSpeak* foi projetada para a programação de agentes BDI na forma de sistemas de planejamento reativos (*reactive planning systems*). Sistemas de planejamento reativos são sistemas que estão permanentemente em execução, reagindo a eventos que acontecem no ambiente em que estão situados através da execução de planos que se encontram em uma biblioteca de planos parcialmente instanciados. Um SMA desenvolvido no interpretador Jason possui suporte para comunicação entre agentes distribuídos utilizando a teoria de atos de fala.

De acordo com [BOR07], além de interpretar a linguagem *AgentSpeak*, possui outros recursos:

- Negação forte (*strong negation*), portanto tanto sistemas que consideram mundo-fechado (*closed-world*) quanto mundo-aberto (*open-world*) são possíveis;

- Tratamento de falhas em planos;

- Comunicação baseada em atos de fala (incluindo informações de fontes como anotações de crenças);

- Anotações em identificadores de planos, que podem ser utilizadas na elaboração de funções personalizadas para seleção de planos;

- Suporte para o desenvolvimento de ambientes (que normalmente não é programada em AgentSpeak; no Jason o ambiente é programado em Java);

- A possibilidade de executar o SMA distribuidamente em uma rede (usando o SACI);

- Possibilidade de especializar (em Java) as funções de seleção de planos, as funções de confiança e toda a arquitetura do agente (percepção, revisão de crenças, comunicação e atuação);

- Possui uma biblioteca básica de “ações internas”;

- Possibilita a extensão da biblioteca de ações internas.

2.2.3.1 JADE

Segundo [JAD08], o JADE, *Java Agent DEvelopment Framework*, é um *framework* para o desenvolvimento de sistemas multiagentes que segue os padrões estabelecidos pela FIPA (*Foundation For Intelligent Physical Agents*) [FIP08], totalmente implementado em Java e fornece uma biblioteca de classes para a implementação de agentes. Além disso, JADE provê uma plataforma distribuída sobre a qual os agentes serão executados. Por ser uma plataforma distribuída, pode-se ter agentes e SMAs em vários computadores de uma rede, e agentes migrando de um computador para o outro.

O JADE é composto por um ambiente de execução, uma biblioteca de classes e um pacote de ferramentas de suporte. Segundo [ELI05] o JADE utiliza os seguintes princípios:

- Interoperabilidade: seguindo as especificações FIPA, os agentes do JADE podem interoperar com outros agentes;

- Uniformidade e portabilidade: o JADE fornece APIs homogêneas independentes da versão Java e da rede utilizada;

- Facilidade de uso: a complexidade do middleware é escondida por um conjunto simples e intuitivo de APIs;

- Filosofia do pay-as-you-go: os programadores não precisam utilizar todas as funções fornecidas pelo middleware.

De acordo com [ELI05] o JADE oferece as seguintes funções aos programadores de agentes:

- Plataforma de agentes distribuída. Uma plataforma pode ser dividida em diversos hosts, conectados via RMI. Cada host executa apenas um JVM (Java Virtual Machine). Os agentes são implementados como threads Java e estão contidos em Agent Containers.

- Interface gráfica com usuário para controlar remotamente os agentes e o Agent Containers;

- Ferramentas para correção de erros;

- Mobilidade de agentes entre plataformas. Inclui transparência do estado e código dos agentes;

- Suporte à execução de múltiplas, paralelas e concorrentes atividades de agentes, por meio de um modelo de comportamento. O JADE gerencia os comportamentos do agente em um formato não-preemptivo;

- Plataforma de agentes adere ao FIPA, incluindo o AMS (Sistema de Gerenciamento de Agentes), o DF (Facilitador de Diretório) e o ACC (Canal de Comunicação de Agentes). Estes três componentes são ativados automaticamente na inicialização da plataforma;

- Diversos DFs podem ser iniciados em tempo de execução para implementação de aplicações com diversos domínios. Cada domínio é um conjunto lógico de agentes, e cada conjunto possui serviços que são publicados por um facilitador comum. Os DFs herdam uma GUI e todas as capacidades padrões definidas pela FIPA;

- Transporte eficiente de mensagens ACL (linguagem baseada em ações de fala) dentro da mesma plataforma de agentes. Internamente, as mensagens são transferidas como objetos Java. Quando atravessam os limites da plataforma, são automaticamente convertidas na sintaxe da FIPA.

- Biblioteca com os protocolos de interação da FIPA;

- Cadastramento e descadastramento automático de agentes com o AMS (*Agent Management System*);

- Serviço de nomeação dentro dos padrões da FIPA. Quando o agente é criado, cada um obtém seu GUID (Identificador Único Global), fornecido pela plataforma;

- Suporte a ontologias e a linguagens de conteúdo definidas por aplicações;

- Interface *InProcess*, possibilitando que aplicações externas executem agentes autônomos.

JADE disponibiliza um conjunto de ferramentas gráficas de suporte, que simplificam a administração da plataforma e o desenvolvimento de aplicações. Em [ELI05] encontramos as seguintes definições:

- *Remote Monitoring Agent* (RMA): atua como console gráfico para administração e controle da plataforma;

- Directory Facilitador (DF): é uma interface gráfica completa que permite, de maneira simples e intuitiva, controlar a base de conhecimento de um DF;

- *Dummy Agent*: é uma ferramenta que possibilita o monitoramento e correção de erros, construída como uma interface gráfica e com um agente JADE. Mensagens ACL podem ser compostas e enviadas a outros agentes;

- *Sniffer Agent*: é um agente que pode interceptar mensagens ACL enquanto elas trafegam, sendo apresentadas graficamente por meio de uma notação similar aos Diagramas de Sequência da UML;

- *Introspector Agent*: é um agente que possibilita a monitoração do ciclo de vida de um agente, suas trocas de mensagens ACL e os comportamentos em execução;

- *LogManagerAgent*: é um agente que permite a configuração, em tempo de execução, das informações de logging;

- *SocketProxyAgent*: é um agente simples, atuando como um gateway bidirecional entre uma plataforma JADE e uma conexão TCP/IP normal. As mensagens ACL são convertidas em strings ASCII, ou vice-versa, e enviadas por uma conexão de *socket*.

Na próxima seção, apresentamos o conceito e características da computação pervasiva.

2.3 Computação pervasiva

Em 1991 Mark Weiser utilizou pela primeira vez o termo computação ubíqua. Atualmente, a computação ubíqua também é chamada de computação pervasiva e constitui na criação de ambientes com vários dispositivos computacionais e comunicação que interagem de maneira integrada aos seres humanos, onde a percepção de se estar lidando com computadores é mínima, de tal forma que estes passam a fazer parte deste ambiente.

Neste novo ambiente computacional, o poder computacional pode estar disponível em qualquer lugar (salas de aula, residências, escritórios, hospitais, automóveis, etc) onde os dispositivos de rede embutidos do ambiente fornecem conexão discreta aos usuários a todo o tempo, sem que o usuário perceba.

Na computação pervasiva a comunicação é centrada no usuário e não no dispositivo. Existem dois tipos de comunicação em sistemas pervasivos: comunicação fixa (onde o dispositivo não precisa de recursos móveis para atender o usuário) e a comunicação móvel (é a comunicação na qual o dispositivo precisa de uma série de recursos de mobilidade para atender o usuário) [SAT01].

Para [SAH03] o computador ainda é visto como uma ferramenta que executa programas em um mundo virtual, onde o usuário executa uma tarefa e ao terminar encerra o programa. A computação pervasiva prevê uma visão bem diferente. Ao invés do usuário ter de entrar no mundo virtual para usar a ferramenta, a ferramenta é que se integra ao mundo do usuário, fazendo com que o mesmo não precise de conhecimento específico para operá-la.

Desde que Weiser concebeu sua visão de pervasividade, evoluções importantes no hardware podem ser constatadas. Esta evolução permitiu a criação de dispositivos menores e mais portáteis, bem como sensores e dispositivos de controle com crescente poder de processamento. É importante citar ainda as tecnologias para comunicação sem fio, como o *Bluetooth* e o IEEE 802.11, criam a possibilidade de acesso à informação a qualquer hora em qualquer lugar, o que não era possível com as redes cabeadas [BON05].

Para [SAH03] a computação pervasiva se divide em quatro grandes áreas: dispositivos, rede, *middleware* e aplicações.

2.3.1 Dispositivos

Em um ambiente inteligente, podemos ter diferentes dispositivos, dos mais simples aos mais sofisticados como: dispositivos tradicionais (mouses, teclados, *speakers*, etc), dispositivos móveis sem fios (*pagers*, PDA, telefones celulares, *palmtops*, etc) e dispositivos inteligentes (eletrodomésticos inteligentes, pisos com sensores embutidos, sensores biológicos, etc).

Para que a computação pervasiva esteja presente nesse ambiente com uma variedade de dispositivos, é necessário que todos os dispositivos possuam inteligência, ou seja, permitam a interação com o ambiente (através de entradas e saídas utilizando sensores e atuadores), possibilitando a comunicação com outros dispositivos. Por exemplo, em uma casa, podemos ter uma geladeira que se comunica com o telefone celular através do *Bluetooth*.

2.3.2 Rede

Como vimos anteriormente, a gama de diferentes dispositivos em um ambiente cresce cada vez mais. Para que esses dispositivos se comuniquem através de uma rede pervasiva é preciso de um canal de comunicação entre esses dispositivos. Para satisfazer a ideia de onipresença da computação pervasiva é preciso o uso de redes de curto, médio e longo alcances como as utilizadas pela telefonia celular, e de redes sem fio como *wi-fi* e *Bluetooth*.

Como conseqüência desta proliferação, muitas das tecnologias atuais devem ser revistas. Além de estender a infraestrutura do *backbone* para antecipar-se a demanda futura, as redes globais como a Internet também devem modificar aplicações existentes para integrar completamente estes dispositivos [SAH03].

2.3.3 Middleware

A computação pervasiva exige um *middleware* capaz de mediar interações entre usuários e o núcleo do sistema, tornando a computação pervasiva invisível ao usuário (mascarando a heterogeneidade). O *middleware* pervasivo deverá fornecer suporte tanto no modo cliente-servidor como no ponto-a-ponto.

Podemos destacar os *middlewares* para ambientes pervasivos que criam uma infraestrutura para o desenvolvimento de aplicações pervasivas, como: MoCA [SAC04], Infraware [PES06] e One.World [GRI04]. O *middleware* MoCA oferece suporte ao desenvolvimento de aplicações distribuídas sensíveis ao contexto que envolvem dispositivos móveis. Já o Infraware é uma plataforma orientada a serviços, com suporte a aplicações móveis sensíveis ao contexto. Enquanto que o One.World procura explorar as limitações dos sistemas distribuídos da atualidade e levanta a questão de como estruturar sistema de suporte para aplicações pervasivas.

2.3.4 Aplicações

Aplicações pervasiva são mais centradas no ambiente do que aplicações para web ou para sistemas móveis. Isto significa que as aplicações sofrem grande influência do *middleware* e da rede. Os usuários precisam das aplicações para facilitar suas tarefas e explorar melhor os recursos do

ambiente pervasivo. Por exemplo, em uma casa inteligente poderíamos ter uma aplicação para gerenciar as preferências dos moradores que, ao detectar a presença de um morador no interior de um determinado cômodo da casa, ajusta a intensidade de luz conforme sua preferência.

Sendo assim, para atingir o conceito da computação pervasiva proposta por Weiser [WEI91], é necessário resolver várias questões e desafios em aberto como: escalabilidade, heterogeneidade, integração, invisibilidade, percepção e gerenciamento de contexto.

2.4 Desafios

O grande desafio da computação pervasiva é prover informação transparente ao usuário integrando vários componentes de hardware e de software sem que o usuário perceba. De acordo com [SAH03], os grandes desafios da computação pervasiva são: escalabilidade, heterogeneidade, integração, invisibilidade, percepção e gerenciamento de contexto.

2.4.1 Escalabilidade

No futuro, ambientes de computação pervasiva enfrentarão uma provável demanda de crescimento de usuários. Isso afetará também o crescimento de aplicações, dispositivos e as interações entre eles. Com o aumento da capacidade dos dispositivos, a intensidade e a qualidade das interações homem-máquina tendem a aumentar [SAH03]. Na computação pervasiva fica impraticável recriar um software para cada novo dispositivo.

Além disso, as aplicações são normalmente distribuídas e instaladas separadamente para cada perfil de dispositivo. Enquanto o número de dispositivos cresce, explicitamente distribuir e instalar aplicações para cada perfil tornar-se impraticável, especialmente para computadores fisicamente distantes. Para o desempenho ser aceitável, interações com computadores fisicamente distantes devem ser reduzidas ao mínimo [COS06].

Para resolver o problema da escalabilidade, temos de desenvolver software que considera a abundância de usuários, interações, componentes e dispositivos, evitando soluções centralizadas e gargalos [COS08].

2.4.2 Heterogeneidade

A heterogeneidade é uma questão relacionada com escalabilidade e herdada dos sistemas distribuídos. Sendo assim, ela consiste na integração de componentes programados em diferentes linguagens, executando sob diferentes sistemas operacionais e em plataformas de hardware diversificadas. Os ambientes de computação pervasiva são caracterizados pela grande diversidade de dispositivos, heterogeneidade de redes e conexão intermitente.

Os aplicativos devem poder executar em diferentes tipos de dispositivos, onde cada dispositivo possui sistemas operacionais e interfaces com o usuário diferentes. Idealmente, os programas devem mascarar para o usuário as diferenças na infra-estrutura e gerenciar as conversões necessárias de um ambiente para outro [COS06].

Para [RIE07] a heterogeneidade deve ser tratada a fim de disponibilizar os serviços e recursos da rede, tratando até mesmo problemas de falhas e desconexões.

2.4.3 Integração

Embora componentes de computação pervasiva já estejam implantados em muitos ambientes, integrá-los em uma única plataforma ainda é um problema de pesquisa. O problema é similar ao encontrado em sistemas distribuídos, só que em escala maior. A integração de diversos dispositivos em um único sistema/ambiente envolve aspectos de confiabilidade, qualidade de serviço e segurança.

Segundo Costa [COS06] para facilitar a integração, deve-se privilegiar o desenvolvimento de sistemas baseados em padrões abertos. Devido à característica móvel e dinâmica das aplicações pervasivas, existe uma constante variação no conjunto de componentes que estas se comunicam. Por isso, a integração trata em especial da associação e da composição de componentes. A primeira diz respeito às formas de estabelecer uma relação lógica entre componentes para a interoperação, enquanto que a segunda lida com a utilização de componentes em vários níveis, disponibilizando e utilizando serviços uns dos outros.

2.4.4 Invisibilidade

A invisibilidade é a principal característica da computação pervasiva, seguindo a ideia de Weiser que diz que as “tecnologias desaparecem”, ou seja, tornam-se invisíveis. O sistema pervasivo requer a mínima intervenção humana. A intervenção deve ocorrer somente quando o sistema não consegue satisfazer o usuário automaticamente [SAH03]. Para Satyanarayanan [SAT01], a invisibilidade é permitir que o usuário não precise conhecer ou entender a tecnologia, apenas desfrutar dos benefícios que ela possa oferecer.

De acordo com Costa [COS08] o primeiro passo em direção a um sistema invisível é projetar aplicações adaptáveis. Para isso, precisamos do apoio de um *framework* que facilita este desenvolvimento, seguindo os objetivos do desaparecimento da computação e mantendo o usuário centrado na tarefa. No momento da execução exige-se uso ininterrupto com a mínima intervenção do usuário. Por exemplo, podem ocorrer períodos de desconexão nos dispositivos móveis. Atualmente, o sistema deve mascarar esta desconexão, mantendo os serviços ininterruptos e ainda satisfazer as necessidades do usuário, talvez com alguma degradação.

2.4.5 Percepção

A percepção também encontrada na literatura como consciência de contexto ou sensiblidade ao contexto é uma característica intrínseca de ambientes inteligentes. A maioria dos sistemas de computação e dispositivos de hoje não podem perceber seus ambientes e, portanto, não podem tornar oportuna, decisões sensíveis ao contexto. Para [SAH03] a computação pervasiva requer sistemas e dispositivos que percebam o contexto.

Dey [DEY00] define contexto como, qualquer informação que possa ser utilizada para caracterizar situação de uma entidade. A entidade pode ser uma pessoa, lugar ou objeto considerado relevante para interação entre um usuário e uma aplicação. As informações de contexto mais usuais são localização, tempo, temperatura, estados de objetos computacionais e físicos.

Implementar percepção introduz alguns problemas como: monitoramento do local, informações em tempo real a serem processadas, informações de diferentes sensores e, possivelmente, com valores que divergem. As informações que definem consciência de contexto devem ser precisas para não depender da experiência do usuário.

2.4.6 Gerenciamento do contexto

Uma vez que é possível perceber o contexto, é necessário usar essa informação e agir de forma pró-ativa. Gerência de contexto é a ação de responder a uma mudança detectada. A ideia é tomar decisões com base em informações sentidas. Decisões como a configuração de serviços de acordo com mudanças ambientais ou manutenção da memória de ambientes do passado para reativar serviços quando o usuário retorna são exemplos de atividades relacionadas ao gerenciamento de contexto.

Um outro cenário seria o de um hospital onde podemos ter uma rede de serviços de saúde, os funcionários (médicos, enfermeiros, etc) podem ter acesso aos serviços desta rede de qualquer lugar, usando um conjunto de diferentes dispositivos e redes de acesso. Em um caso de emergência, a partir das informações de contexto, poderíamos localizar o médico especialista mais próximo para atender o paciente.

Para minimizar os desafios que surgiram com a computação pervasiva, vários projetos estão sendo desenvolvidos para atacar as questões levantadas. Os projetos utilizam uma plataforma de *middleware* composto por um conjunto de serviços com o objetivo de oferecer suporte às aplicações pervasivas. Na próxima seção, veremos alguns trabalhos relacionados aos *middlewares* desenvolvidos.

3 TRABALHOS RELACIONADOS

Este capítulo integra a base teórica da pesquisa desenvolvida e apresenta alguns trabalhos relacionados a aplicações que utilizam os conceitos de agentes de software e computação pervasiva, com destaque para aplicações sensíveis ao contexto utilizando sistemas multiagentes. Será apresentada uma breve descrição sobre as aplicações, suas características, arquitetura e tecnologias utilizadas para o desenvolvimento das aplicações.

3.1 MoCA/MAX

A MoCA/MAX (*Mobile Collaboration Architecture/Multi-Agent eXtension)* [VIT06a] é um *middleware* que suporta o desenvolvimento de aplicativos sensíveis ao contexto baseados no paradigma multiagentes. O objetivo é integrar o MoCA com o *framework* JADE (*Java Agent DEvelopment Framework*) para gerenciar informações de contexto sobre dispositivos onde agentes estão executando.

De acordo com [VIT06a], para o desenvolvimento e implantação de sistemas multiagentes, é necessário ambientes distribuídos de controle integrado para fornecer os serviços básicos e funcionalidades definidas pelo paradigma de agentes de software. Segundo [JAD08], o JADE, *Java Agent DEvelopment Framework*, é um *framework* para o desenvolvimento de sistemas multiagentes que segue os padrões estabelecidos pela FIPA (*Foundation For Intelligent Physical Agents*) [FIP08], totalmente implementado em Java que fornece uma biblioteca de classes para a implementação de agentes. Além disso, o JADE provê uma plataforma distribuída sobre a qual os agentes serão executados. Por ser uma plataforma distribuída, pode-se ter agentes e SMAs em vários computadores de uma rede, e agentes migrando de um computador para o outro.

A arquitetura MoCA fornece o suporte necessário ao JADE para desenvolver e executar aplicações distribuídas sensíveis ao contexto, especialmente aquelas que compõem dispositivos móveis interligados através redes sem fio (802.11b/g). Segundo [SAC04] o MoCA disponibiliza serviços capazes de coletar, armazenar e processar informações de contexto obtidas dos dispositivos móveis em uma rede sem fio. Além disso, o MoCA integra um conjunto de API’s para o desenvolvimento de aplicações que interagem com esses serviços como consumidores de informações de contexto.

Uma aplicação desenvolvida com base na MoCA é composta por um servidor da aplicação, normalmente executado na rede fixa, e os clientes da aplicação, que são executados em dispositivos móveis. O servidor da aplicação é também um cliente dos serviços MoCA, ou seja, um consumidor de informações de contexto.

Segundo [VIT06a] na arquitetura MoCA/MAX (Figura 1 extraída de [VIT06a]) é proposto um Serviço de Gerenciamento de Contexto (CMS), implementados por agentes que atuam como uma interface entre os serviços MoCA e outros agentes na plataforma. Os agentes que formam esse serviço agem como clientes do MoCA e se comunicam com os serviços CIS (*Context Information Service*) e LIS (*Location Inference Service*) para obter informações de contexto (por exemplo, nível de energia, memória livre, localização, etc) associados com os dispositivos onde os agentes estão em execução.

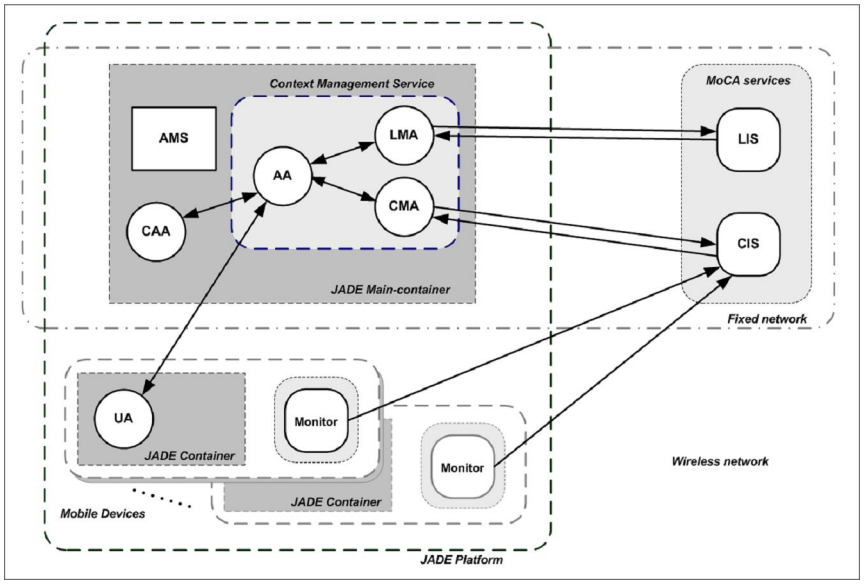

Figura 1: Arquitetura MoCA/MAX

De acordo com [VIT06a] a arquitetura possui três agentes, *Advertisement Agent* (AA), o *Location Management Agent* (LMA) e o *Context Management Agent* (CMA). O *Advertisement Agent* (AA) funciona como uma interface entre os agentes CMS e os agentes da plataforma que desejam obter informações de contexto. O *Advertisement Agent* (AA) remete ao *Advertisement Agent* (CMA) as mensagens sobre os dados do dispositivo computacional associadas com o CIS, e encaminha para o *Location Management Agent* (LMA) mensagens sobre a localização do dispositivo associadas ao LIS. O *Context Management Agent* (CMA) e *Location Management Agent* (LMA) respondem ao *Advertisement Agent* (AA) e as mensagens são enviadas ao agente de

destino. O *Context Management Agent* (CMA) é o agente responsável por obter do MoCA todas as informações cruas associadas aos dispositivos móveis que participam na plataforma. O *Location Management Agent* (LMA) é responsável por controlar questões sobre a localização dos dispositivos.

O serviços do MoCA estão representados pelos serviços CIS e LIS, que são executados em uma rede fixa, e também pelo Monitor sendo executado em cada dispositivo móvel onde, juntamente com os *containers* pertencentes a plataforma, coletam informações de contexto e enviam para o serviço CIS. Um *User Agent* (UA), executando no dispositivo móvel, e o *Context-Aware Agent* (CAA) executando na rede fixa, representam os consumidores de contexto que interagem com o *User Agent* (AA) para obter a informação do contexto. As requisições de *User Agent* (AA) e *Context-Aware Agent* (CAA) são enviadas pelo *Advertisement Agent* (AA) para o *Location Management Agent* (LMA) e o *Context Management Agent* (CMA) que se comunicam com os serviços CIS e LIS respectivamente, do MoCA, para obter as informações necessárias.

3.2 ACAI

O ACAI (*Agent-based Context-aware Infrastructure*) [KHE05] foi desenvolvido pelo Laboratório de Pesquisas em Multimídia e Agentes Móveis (MMARL) da Universidade de Ottawa. Ele oferece uma infraestrutura para o desenvolvimento de aplicações sensíveis ao contexto utilizando sistemas multiagentes. Segundo [KHE05], o ACAI procura integrar, de modo transparente, os diferentes espaços físicos utilizados por um mesmo usuário ativo oferecendo alguns serviços básicos como:

- Composição, inferência e disseminação de contexto;

- Protocolo de comunicação sensível a contexto;

- Serviços para o gerenciamento de contexto;

- Representação de contexto de maneira uniforme;

- API’s para o desenvolvimento de aplicações.

A infraestrutura do ACAI permite a coleta, processamento, inferência e disseminação de informações de contexto de forma contínua e sem revelar às aplicações a complexidade envolvida em gerenciar os fornecedores de contexto heterogêneos [KHE05].

A arquitetura do ACAI está implementada em três camadas (Figura 2 extraída de [KHE05]): a primeira é a camada de percepção, onde ocorre a captura e coleta das informações de contexto; a segunda é camada de serviços de contexto responsável pela descoberta dos serviços disponíveis no ambiente, pelo armazenamento e consumo das informações por outros serviços e pela dedução de informações que não estão disponíveis diretamente a partir da camada de percepção; e por último a camada de aplicação que funciona como interface para camada de serviços, permitindo que as informações de contexto sejam negociadas entre os provedores e consumidores.

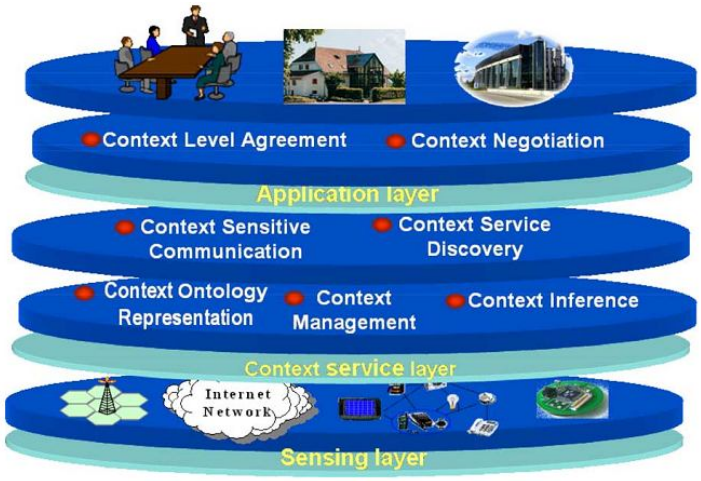

Figura 2: Arquitetura ACAI

O ACAI faz integração de ambientes sensíveis de contexto utilizando o paradigma de agentes. O *Context Management Agent* (CMA), o *Coordinator Agent* (CA) e o *Ontology Agent* (OA) são os principais agentes utilizados pelo sistema multiagentes do ACAI. Dependendo da situação, são utilizados os agentes opcionais como, *Reasoner Agent* (RA) e o *System Knowledge Base Agent* (SKBA). Para que toda essa estrutura de agentes funcione corretamente, o Agente de Fornecimento de Contexto (CPA) deve se registrar junto ao Agente de Gerenciamento de Contexto (CMA). Quando um Agente de Usuário (UA) se registra junto ao CMA informando os serviços que está procurando, este irá:

- consultar os CPA que fornecem esse tipo de contexto/serviço e obter ofertas de fornecimento;

- decidir, consultando o agente de inferência (IA), qual é o CPA que melhor se adéqua a fornecer esse contexto/serviço e;

- encaminhar a proposta ao UA.

Antes de encaminhar a proposta final para o UA, o CMA pode solicitar a um dado CPA que reformule sua proposta, para se aproximar da solicitação inicial. Se o CMA estiver de acordo com o formato do contexto a ser fornecido, ele finalmente encaminha a proposta ao UA, que passa a receber o contexto diretamente do CPA.

3.3 CHIL

O *Computers in the Human Interaction Loop* (CHIL) é um projeto que tem como objetivo criar ambientes em que computadores servem humanos focando a interação com outros humanos ao invés de se preocupar com as máquinas [SOL07]. Com base no entendimento da percepção do contexto humano, requerendo o mínimo de atenção ou interrupções humanas. Para que isso aconteça, a infraestrutura de *middleware* do projeto CHIL desenvolveu serviços pervasivos, sensíveis ao contexto para espaços inteligentes.

A infraestrutura possui mecanismos para provimento de serviço de acesso, controle de sensores e atuadores, modelagem de contexto, serviços de diretório para elementos da infraestrutura e serviços, bem como tolerância a falhas. Isso faz com que o desenvolvedor se concentre no desenvolvimento da lógica do serviço da aplicação e não na execução do *middleware*. A arquitetura do *middleware* foi implementada com um sistema multiagentes distribuído utilizando a plataforma de desenvolvimento de agentes JADE (*Java Agent DEvelopment Framework*).

Segundo [SOL07], vários tipos de agentes são utilizados pelo *framework*:

- Agentes Centrais (*Core Agents*): são independentes do serviço e da instalação em salas inteligentes. Eles fornecem um mecanismo para a comunicação distribuída entre as entidades do sistema, encarregando-se do controle sensorial da infraestrutura. O agentes que compõem os agentes centrais são:

- Agente do Dispositivo Desktop (*Device Desktop Agent*), implementa a interface com o usuário necessária para acessar os serviços pervasivos. Um mecanismo de conexão “automática” permite que a interface do usuário possa ser personalizada para determinado serviço da computação pervasiva.

- Agente do dispositivo (*Device Agent*), permite que diferentes dispositivos possam se comunicar com o *framework.*

- Agente pessoal (*Personal Agent*), transmite pedidos de usuário ao agente gerenciador, que são tratados por agentes de destino. Mantém o perfil do usuário, a fim de personalizar os serviços prestados ao usuário final.

- Agente gerenciador (*Agent Manager*), permite ao sistema aumentar dinamicamente os serviços adicionais dos agentes, bem como os serviços de computação pervasiva incorporados ao sistema.

A Figura 3 (extraída de [SOL07]) mostra o *framework* multiagentes que dá apoio a implementação de serviços pervasivos.

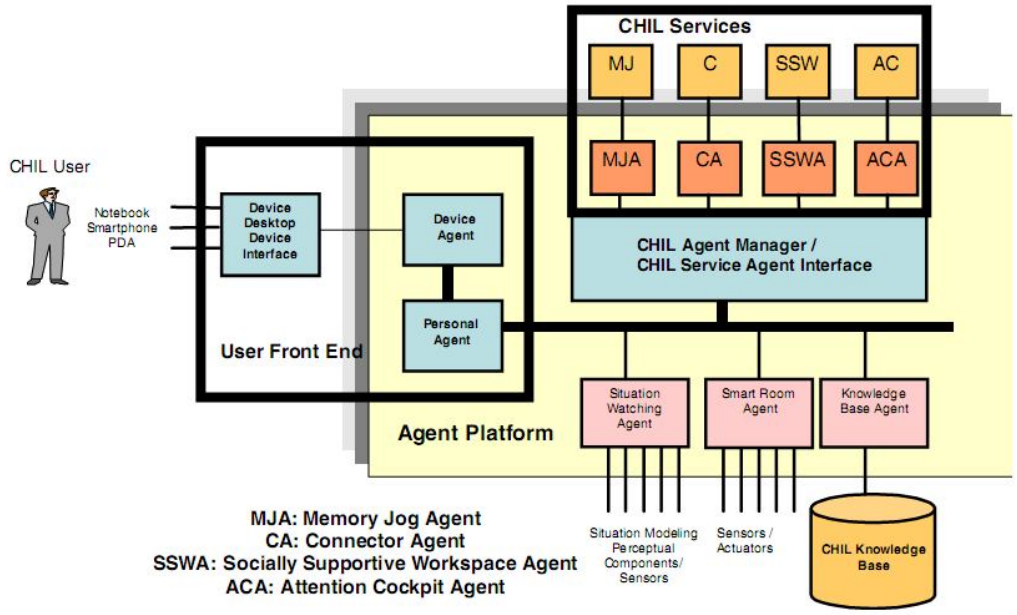

Figura 3: Infraestrutura do projeto CHIL

- Agentes de serviços básicos (*Basic Services Agents*): estes agentes incorporam os serviços lógicos dos serviços básicos, que estão associados com a infraestrutura de cada sala inteligente. Os serviços básicos incluem a capacidade de monitorar situações complexas, bem como o controle de atuadores e sensores. O agente SWA (*Situation Watching Agent*) é o responsável pelo monitoramento com base na modelagem do contexto, enquanto que o controle dos sensores e atuadores é feito através do *Smart Room Agent*. Além disso, o *Knowledge Base Agent*, permite que os agentes do *framework* acessem dinamicamente as informações sobre o estado dos componentes da computação pervasiva no ambiente (por exemplo, sensores, atuadores, componentes de

percepção), através de um *Knowledge Base Server* que é suportado como uma ontologia do sistema de gerenciamento.

- Agentes de serviços ubíquos (*Ubiquitous Service Agents*): implementa a lógica de vários serviços sensíveis ao contexto. Cada serviço pervasivo é implementado como um agente de serviços pervasivo ligados ao *framework*. No projeto CHIL, vários agentes pervasivos correspondem aos diversos serviços implementados e integrados ao *framework* como: serviço MJ (*Memory Jog*), executado através do *Memory Jog Agent* (MJA), o *Connector Agent* (CA), o *Socially Supportive Workspaces Agent* (SSWA) e o *Attention CockpitAgent* (ACA).

3.4 COBRA

O CoBrA (*Context Broker Architecture*), foi desenvolvido pela Universidade de Mariland, é uma arquitetura baseada em agentes para dar apoio a sistemas pervasivos sensíveis ao contexto em espaços inteligentes [CHE04]. A arquitetura CoBrA foi projetada para abordar quatro questões chaves na construção de sistemas sensíveis ao contexto: a forma de representar as informações de contexto, como possibilitar o compartilhamento dessas informações, como permitir a inferência de novos dados e como proteger a privacidade do usuário. Os agentes da arquitetura CoBrA foram implementados utilizando o *framework* JADE.

O componente principal da arquitetura CoBrA, possui um agente inteligente central chamado c*ontext broker* (negociador de contexto), que possui algumas responsabilidades [CHE04] em um ambiente inteligente:

- Fornecer um modelo centralizado de contexto, que todos os dispositivos, serviços e agentes no espaço podem compartilhar;

- Receber informações de contexto de fontes que não são acessíveis a partir de dispositivos que têm recursos limitados;

- Inferir, a partir das informações de contexto, dados que não podem ser recebidos diretamente de sensores;

- Detectar e corrigir inconsistências nos dados de contexto recebidos, e

- Proteger a privacidade permitindo que usuários definam suas preferências políticas de privacidade para o compartilhamento e uso de suas informações de contexto.

Segundo Chen, o *context broker* está dividido em quatro componentes principais, conforme Figura 4, extraído de [CHE04]:

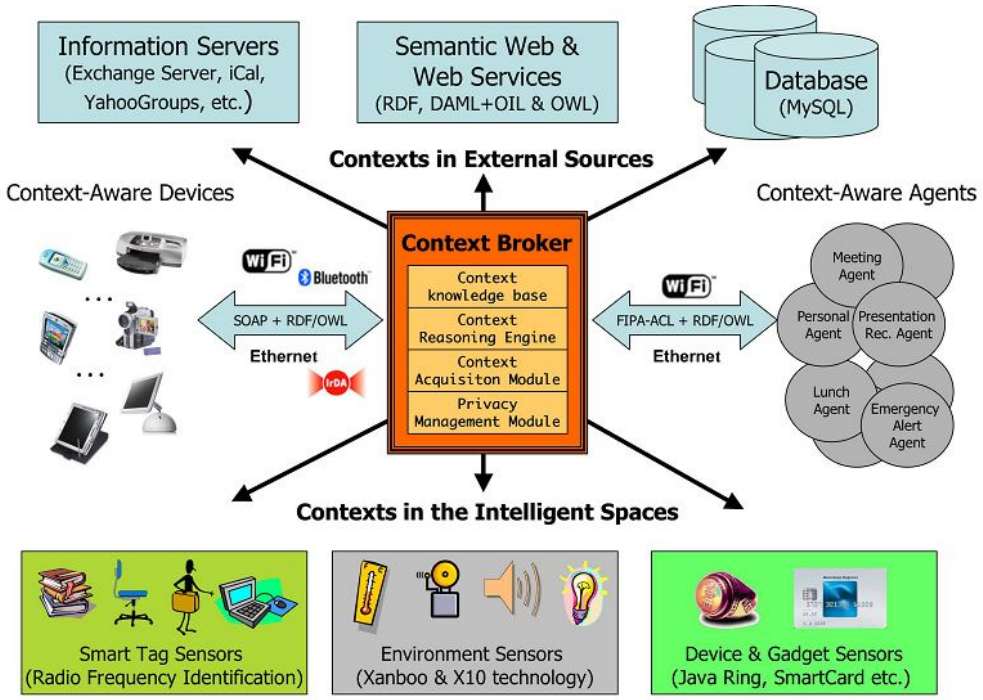

Figura 4: Arquitetura CoBrA

- *Context knowledge base*: armazena as informações de contexto, utilizando ontologias para descrever vários tipos de contexto, por exemplo, os dados da ontologia são adquiridos de informações de contexto, e os metadados descrevem a estrutura de armazenamento da representação do conhecimento;

- *Context-reasoning engine*: é uma máquina de inferência de contexto, que permite determinar novos dados de contexto a partir de dedução lógica baseada na semântica do OWL (*Web Ontology Language*) e regras de dedução;

- *Context-acquisition module*: o módulo de captação de contexto consiste em uma biblioteca de procedimentos que adquire informações de contexto de sensores, agentes e da Web. Ela esconde a complexidade envolvida na coleta de contexto;

- *Privacy-management module*: o módulo responsável pelo gerenciamento da política de privacidade determina quando as informações de contexto, de um usuário ou de um agente, podem ser compartilhadas.

3.5 Comparativo entre abordagens descritas

Como apresentado nas seções anteriores, descrevemos sobre os trabalhos relacionados que oferecem uma infraestrutura para o desenvolvimento de aplicações sensíveis ao contexto utilizando sistemas multiagentes, cada qual com suas contribuições e limitações. A Tabela 1 apresenta um comparativo entre os trabalhos relacionados com as principais características das abordagens descritas elencadas. Na tabela são destacados os seguintes aspectos:

- Gerenciamento de Contexto: fornecer facilidades para o desenvolvimento de aplicações sensíveis ao contexto;

- Descrição do Contexto: indica que tipo de descrição de contexto é usado;

- Ferramentas de desenvolvimento: que plataforma de desenvolvimento multiagentes foi utilizada;

- Tratamento da localização: como a abordagem faz para prover a localização de pessoas ou objetos;

- API's para o desenvolvimento de aplicações: verificar se as abordagens oferecem API's para o desenvolvimento de aplicações.

Como podemos observar na Tabela 1, todas as abordagens utilizaram a ferramenta de desenvolvimento JADE e também oferecem API's para o desenvolvimento de aplicações sensíveis ao contexto. As informações de contexto percebidas no ambiente são gerenciadas de forma centralizada no MoCA/MAX e CoBRA, enquanto que no ACAI e CHIL o gerenciamento de contexto é feito de forma distribuída entre os agentes da plataforma.

A maioria das abordagens se preocupa em descrever informações de contexto sobre usuários, dispositivos e espaços. Para a descrição do contexto ACAI, CoBRA e CHIL contam com o uso de ontologias OWL enquanto que o MoCA/MAX utiliza um formato proprietário para fazer a descrição do contexto.

Quanto ao tratamento da localização, o MoCA/MAX utiliza um agente chamado LMA (*Location Management Agent*) para receber mensagens sobre a localização do dispositivo no ambiente. No CoBRA o agente concentrador *Context Broker* se encarrega de repassar as informações de localização quando solicitadas por outros agentes. No CHIL as informações de contexto são armazenadas em uma base de dados e o *Knowledge Base Agent* permite que os agentes acessem dinamicamente as informações sobre o estado dos componentes da computação pervasiva no ambiente. Por fim, o ACAI utiliza um agente de fornecimento de contexto (CPA) para capturar as informações de contexto brutas e as interpreta com o auxilio do agente de ontologia. Essas informações são armazenadas em uma base de conhecimento do sistema que estarão disponíveis para os outros agentes do sistema.

Tabela 1: Comparativo entre os trabalhos relacionados

	MOCA/MAX	ACAI	COBRA	CHIL
Gerenciamento de Contexto	Centralizado (Gerenciamento de Contexto - CMS)	Distribuida (Funções distribuidas entre agentes)	Centralizada (*Context Broker*)	Distribuida (Agentes da plataforma)
Descrição do Contexto	Formato proprietário	Ontologias OWL	Ontologias OWL	Ontologias OWL
Ferramentas de desenvolvimento	JADE	JADE	JADE	JADE
Tratamento da localização	Agente (*Location Management Agent* - LMA)	Agente (Agente de Fornecimento de Contexto CPA)	Agente (*Context Broker*)	Agente (*Knowledge Base Agent*)
API's para o desenvolvimento de aplicações	sim	sim	sim	sim

4 INTRODUZINDO LOCALIZAÇÃO NO SEMANTICORE

Neste capítulo, vamos abordar a arquitetura criada para integrar serviços pervasivos a plataforma do SemantiCore [RIB04], a fim de permitir a criação de SMAs utilizando-se de agentes rodando em diversos dispositivos móveis. Escolhemos o SemantiCore pela sua facilidade na criação de SMAs e pela possibilidade de acesso a seu código fonte e por ser do interesse do grupo de pesquisa do ISEG (*Intelligent Systems Engineering Group*) e o *middleware* MoCA (*Mobile Collaboration Architecture*) [SAC04] por gerenciar as informações de contexto sobre dispositivos onde os agentes estarão executando, em especial os dispositivos móveis.

4.1 SemantiCore

Conforme [RIB04], o SemantiCore é um *framework* que provê uma camada de abstração sobre serviços de distribuição e uma definição interna de agentes capaz de oferecer aos desenvolvedores uma abstração de alto nível para a construção de SMAs. O SemantiCore surgiu em 2004, a partir de uma extensão da arquitetura do *Web Life* [RIB02]. A versão utilizada para o desenvolvimento deste trabalho foi o SemantiCore 2006.

De acordo com [ESC06] o *framework* SemantiCore 2006 é dividido em dois modelos: o modelo do agente e o modelo do domínio semântico. Os dois modelos dispõem de pontos de flexibilidade (*hotspots*) permitindo aos desenvolvedores associar diferentes padrões, protocolos e tecnologias.

O modelo de agentes define todos os elementos necessários para construir um agente no SemantiCore (*SemanticAgent*), como ilustrado na Figura 5 (extraído de [ESC06]). Este agente é composto por quatro componentes básicos: o sensorial, o decisório, o executor e o efetuador.

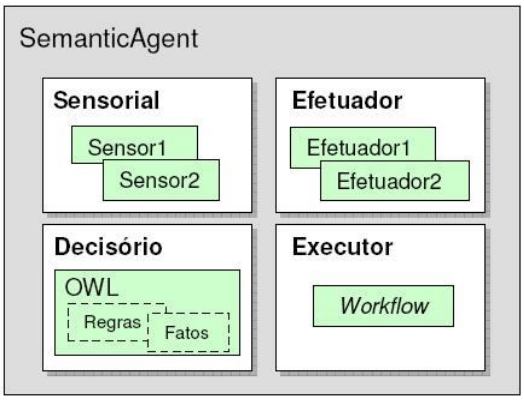

Figura 5: Arquitetura de um agente semântico

- Componente sensorial: centraliza os elementos que são capazes de receber objetos do ambiente. Este componente armazena os sensores definidos pelo desenvolvedor e verifica se um destes sensores será ativado pela recepção de um objeto do ambiente.

- Componente decisório: recebe os objetos vindos do componente sensorial e processa esses dados baseando-se em regras e fatos em uma dada linguagem de representação. O componente decisório possui um mecanismo responsável pelo processamento dos objetos recebidos. Este mecanismo é chamado DecisionEngine, e ele é um ponto de flexibilidade da plataforma, pois neste componente podem ser inseridas máquinas de inferência capazes de processar código OWL (Web Ontology Language) como é o caso da máquina de inferência do Jena [JEN09].

- Componente executor: é responsável por armazenar os planos de ação do agente e gerenciar as suas execuções. Estas ações são criadas pelo desenvolvedor estendendo a classe Action. Com isso, quando houver a necessidade de disparo de uma ação pelo resultado do processamento decisório, a ação definida pelo usuário é instanciada e gerenciada pelo executor.

- Componente efetuador: para um agente publicar um conteúdo no ambiente, é necessário que ele possua um efetuador capaz de publicar no ambiente o formato no qual este conteúdo está estruturado. Os efetuadores de um agente são controlados pelo componente efetuador. Este componente recebe dados dos outros componentes e verifica em qual efetuador este objeto deve ser processado. Assim como os sensores, os efetuadores são pontos de flexibilidade, permitindo aos desenvolvedores a utilização de diferentes tecnologias para a transmissão de dados.

Cada componente e responsável por uma tarefa especializada. A estrutura orientada a componentes de um *SemanticAgent* auxilia o desenvolvedor a estruturar o agente em partes. Esta

modularização oferece benefícios em termos de manutenção, organização do código e extensão funcional dos agentes.

O agente atua em um ambiente. No SemantiCore o ambiente é denominado domínio semântico, que requer um domínio Web para operar. Na Figura 6 (extraída de [ESC06]), representamos o modelo de domínio, composto por algumas entidades administrativas como: Controlador de Domínio (*Domain Controller*) e o Gerente de Ambiente (*Environment Manager*). O Controlador de Domínio é responsável por fazer o registro dos agentes no ambiente, pela recepção de agentes móveis vindos de outros domínios e também pelas características de segurança. O Gerente de Ambiente representa uma ponte entre o domínio semântico do SemantiCore e os domínios Web convencionais.

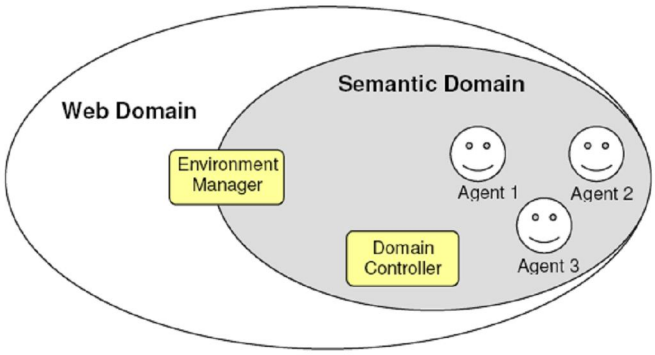

Figura 6: Representação do modelo de domínio

4.1.1 Comunicação

A comunicação entre os agentes do domínio é feita através de dois barramentos, o de controle e o de dados. O barramento de controle é responsável pela troca de mensagens entre os componentes dos agentes distribuídos, enquanto que, no barramento de dados trafegam as mensagens trocadas entre diferentes agentes, auxiliadas pelos componentes sensorial e efetuador de cada agente. Esses dois barramentos têm a finalidade de não sobrecarregar o barramento de dados com mensagens de controle.

As informações trocadas entre domínios passam pelo agente Gerenciador de Comunicação (*Communication Manager*) que tem a responsabilidade de receber e enviar informações até o

destino. A informação deve ser uma mensagem padrão do SemantiCore, chamada de *Semantic Message.*

A *Semantic Message* possui três tipos de endereçamento de mensagem: *unicast* (mensagem endereçada diretamente a um agente), *multicast* (mensagem endereçada a um grupo de agentes) e *broadcast* (mensagem é enviada para todos os agentes do domínio). Na Figura 7, podemos ver um exemplo de endereçamento de mensagem *broadcast* onde, o primeiro campo "PervasiveSemanticore" é a origem da mensagem, o segundo campo indica o destino da mensagem, no caso "*", que significa *broadcast* e por fim o campo da mensagem representado pela mensagem "*Hello*".

```
new SemanticMessage ("PervasiveSemanticore", "*", "Hello");
```

Figura 7: Exemplo de mensagem Broadcast

O SemantiCore não possui o conceito de localidade. Conseguimos trocar mensagens somente dos modos descritos anteriormente. Como a ideia deste trabalho é inserir a noção de localização no agente para que se possam criar aplicações onde o envio da mensagem, por exemplo, possa ser endereçado aos agentes localizados em uma determinada região ou enviar mensagens para os agentes que se encontram mais próximos de um dado agente.

É importante para aplicações utilizando agentes em ambientes pervasivos sensíveis ao contexto saber a sua localização atual. A localização juntamente com outros atributos como, as preferências do usuário, dispositivos próximos, atual atividade do usuário, por exemplo, podem ser utilizadas para criação de aplicações que levem a informação de forma útil ao usuário naquele instante, ou seja, a informação deve ser disponibilizada de acordo com o contexto que o usuário está inserido.

Para [HIG01] a localização simbólica está relacionada com uma ideia abstrata da localização de uma entidade. Por exemplo, um objeto poderia estar em uma sala, no corredor ou então próximo a máquina de café. A localização simbólica será útil aos agentes pois, após a definição das regiões, podemos inserir a localização aproximada dos agentes.

Uma forma de inserir localização ao agente seria estender a mensagem *Semantic Message* para incorporar a localização como forma de endereçamento da mensagem. Além de enviar mensagens da forma habitual, enviaríamos mensagens endereçadas a uma determinada região onde

se encontram os agentes, para os agentes que se encontram mais próximos da região percebida ou endereçada a um único agente.

4.2 MoCA

A MoCA (*Mobile Collaboration Architecture*) [SAC04] é uma arquitetura que oferece suporte ao desenvolvimento de aplicações distribuídas sensíveis ao contexto que envolvem dispositivos móveis interconectados através de redes wireless LAN infra-estruturadas (IEEE 802.11b/g). Foi desenvolvida pelo LAC (*Laboratory for Advanced Collaboration*) da Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

A MoCA disponibiliza serviços para coletar, armazenar e processar informações de contexto obtidas dos dispositivos móveis em uma rede sem fio. A MoCA possui ainda, um conjunto de API's (*Application Programming Interface*) para o desenvolvimento de aplicações que interagem com esses serviços como consumidores de informações de contexto.

Segundo [VIT06b] uma aplicação desenvolvida com base na MoCA é composta por um servidor da aplicação (normalmente executado na rede fixa) e os clientes da aplicação (executados em dispositivos móveis). O servidor da aplicação é também um cliente dos serviços MoCA, ou seja, um consumidor de informações de contexto. Ele registra serviços no DS (*Discovery Service*), informando seu endereço e características do serviço, e poderá ser localizado pelos clientes da aplicação. Na Figura 8 (extraída de [VIT06b]), é apresentada a arquitetura típica de uma aplicação na MoCA.

Além disso, a arquitetura oferece um conjunto de serviços essenciais para o desenvolvimento de aplicação sensíveis ao contexto e de colaboração como:

- CS (*Configuration Service*): disponibiliza informações sobre qual servidor CIS é responsável por coletar dados do dispositivo e qual deve ser a periodicidade do envio de dados.

- CIS (*Context Information Service*): recebe, armazena e processa as informações de contexto enviadas pelas instâncias do monitor em execução nos diversos dispositivos móveis. Estas informações podem ser consultadas pelas aplicações interessadas de forma síncrona ou assíncrona. Na forma síncrona, as aplicações podem consultar as informações atualizadas e um determinado dispositivo de interesse. Já na forma assíncrona, podemos registrar interesses em estados específicos do dispositivo, por exemplo, a aplicação solicita ao CIS que seja notificada quando a carga da

bateria de um determinado dispositivo chegar ao nível de 20% de capacidade de carga. Os estados específicos são descritos por expressões lógicas que envolvem diversas variáveis de contexto de um dado dispositivo.

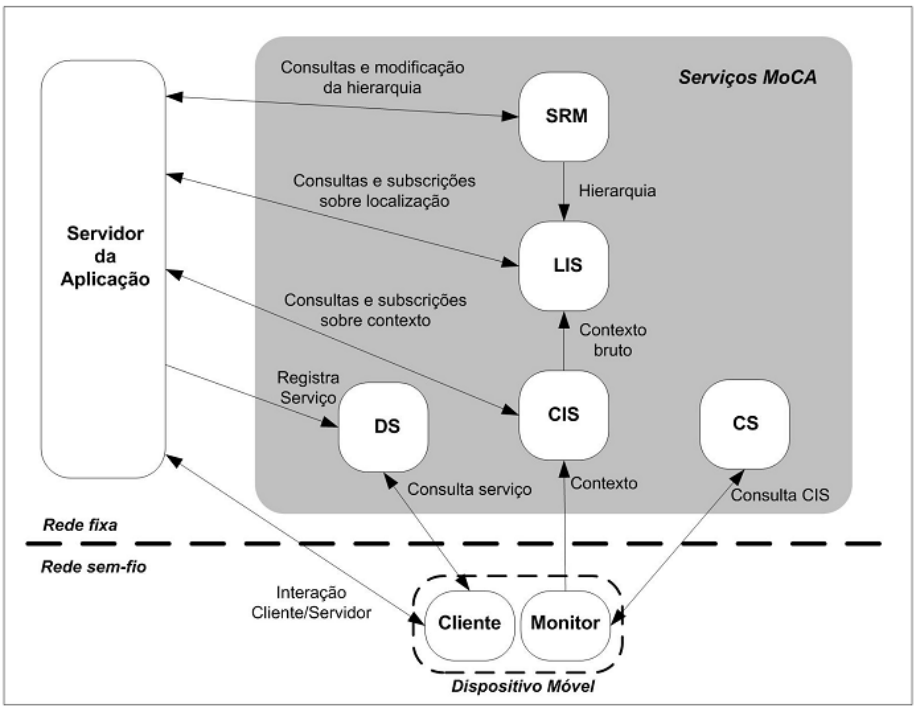

Figura 8: Arquitetura MoCA

- DS (Discovery Service): o servidor da aplicação, é um consumidor de informações de contexto, ele se registra serviço no DS, informando seu endereço e características do serviço, e poderá ser localizado pelos clientes da aplicação.

- LIS (Location Inference Service): é um serviço responsável por inferir a localização aproximada de um dispositivo móvel comparando o padrão corrente de sinais de radiofreqüência observados pelo dispositivo (obtidos de pontos de acesso 802.11 dentro do raio de cobertura) com o padrão de sinais medidos em pontos de referência pré-definidos, seja em um ambiente fechado (indoor) ou aberto (outdoor) [VIT06b]. O serviço LIS permite ao usuário definir regiões simbólicas, ou seja, associar nomes a regiões físicas bem definidas (por exemplo, salas, prédios, corredores), que são de interesse para aplicações sensíveis a localização. Assim como o CIS as informações também podem ser feitas de forma síncrona e assíncrona. De forma síncrona, podemos ter uma aplicação com a capacidade de exibir quais os dispositivos se encontram em uma determinada área

simbólica. Na forma assíncrona, podemos registrar o interesse em eventos de mudança de área de um determinado dispositivo.

- SRM (Symbolic Region Manager): permite estabelecer uma relação entre as regiões atômicas definidas pelo LIS, definindo uma hierarquia em que regiões podem estar subordinadas a outras, ou seja, contidas em outras regiões.

- Monitor: é executado no dispositivo móvel, esse serviço é responsável por coletar e divulgar as informações de contexto do dispositivo e da rede, como: a qualidade da conexão sem-fio, a carga da bateria, o uso da CPU, a memória livre, o ponto de acesso corrente (AP), e uma lista de todos os pontos de acesso dentro do alcance do dispositivo e a respectiva potência dos sinais recebidos, como podemos ver na Figura 9.

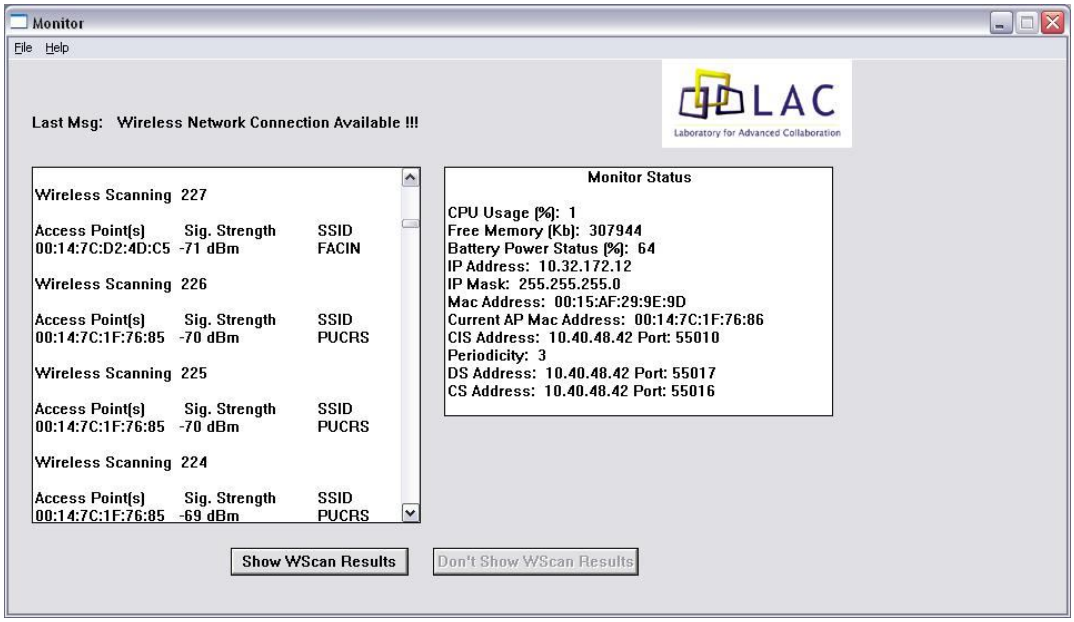

Figura 9: Monitor do MoCA em execução no dispositivo móvel

Além do envio dessas informações, o Monitor tem a capacidade de notificar o serviço CIS sobre qualquer mudança do endereço IP ou ponto de acesso (AP) corrente do dispositivo. Existe também uma versão para aplicação utilizada para simular uma rede sem fio chamado de *Monitor Simulator*. Ele simula o Monitor em execução no dispositivo móvel, enviando para o serviço CIS as informações contidas em um arquivo de configuração.

Além dos serviços, a MoCA possui API's para o desenvolvimento de aplicações. O conjunto de API's está divididos em três grupos: de comunicação, principal e opcionais. As API's de

comunicação fornecem interfaces de comunicação síncrona e assíncrona via portas UDP (*User Datagram Protocol*) e TCP (*Transmission Control Protocol*). As API's principais fornecem interfaces de comunicação com serviços básicos da arquitetura. Por fim, as API's opcionais utilizadas para o desenvolvimento de aplicações baseadas em arquitetura cliente/servidor.

De acordo com [VIT06b], abaixo, apresentamos as características das principais API's:

- *Communication Protocol*: auxilia o desenvolvimento de aplicações que implementam trocas de mensagens síıncronas ou assíncronas usando TCP ou UDP. Esta API é usada pela maioria das implementações dos serviços MoCA;

- *Event-based Communication Interface*: implementa comunicação assíncrona baseada em eventos (*publish/subscribe*);

- *CIS Client*: fornece uma interface de comunicação com o serviço CIS para permitir a realização de consultas síncronas ou assíncronas sobre informações de contexto dos dispositivos;

- *LIS Client*: fornece uma interface de comunicação com o serviço LIS para permitir a realização de consultas síncronas ou assíncronas sobre informações de localização dos dispositivos e áreas mapeadas no serviço;

- SRM: fornece uma interface de comunicação com o serviço SRM para definir ou consultar a hierarquia estabelecida entre as regiões atômicas.

4.2.1 Aplicações utilizando MoCA

Algumas aplicações sensíveis ao contexto já foram desenvolvidas utilizando a arquitetura MoCA. Entre elas podemos destacar algumas aplicações em [VIT06b] que utilizam os recursos do LIS para implementar serviços baseados em localização.

O uGuide (*Ubiquitous Guide*) é uma aplicação cliente/servidor que associa uma URL a cada região simbólica registrada no servidor de aplicação. O uGuide informa a região atual na qual o usuário se encontra e fornece uma opção de acrescentar uma *website* fornecendo informação extra sobre tal localidade. Desta forma, cada vez que um usuário entra em uma determinada região,com o UGuide ativado, ele exibirá uma mensagem informando que o usuário trocou de região.

O Nita (*Notes in the air*) [GON04] permite a publicação de mensagens em regiões simbólicas, como se fossem quadros virtuais. Esta aplicação também permite comunicação síncrona baseada em localização, isto é, o estabelecimento de salas de bate-papo definidas para regiões simbólicas.

O WMS (*Wireless Marketing Service*) é uma aplicação que possibilita que estabelecimentos comerciais enviem anúncios e cupons de desconto direcionados a um local específico para clientes móveis que passeiam em um shopping ou em uma loja de departamentos. Toda vez que um cliente (dispositivo móvel) em potencial entra em uma dessas regiões, o servidor da aplicação cria e envia para o cliente em execução no dispositivo móvel um cupom de marketing virtual que permanece válido durante todo o período pré-estabelecido. Esse cupom é específico para um determinado usuário e não pode ser replicado.

4.3 Integração SemantiCore/MoCA

Esta seção apresenta a integração do SemantiCore com a *middleware* pervasivo MoCA. A integração dessas plataformas visa fornecer suporte arquitetural adequado para o desenvolvimento de um sistema multiagentes com características pervasivas.

Como a MoCA dá suporte ao desenvolvimento de aplicações sensíveis ao contexto, optamos por inserir localização aos agentes do SemantiCore. Escolhemos a MoCA, pois além de gerenciar as informações de contexto dos dispositivos móveis, ela possui documentação disponível e suporte ao programador. A integração do *framework* SemantiCore com *middleware* MoCA gerou uma nova arquitetura, a qual denominamos de SemantiCore Pervasivo (SCPe).

4.3.1 Arquitetura do SemantiCore Pervasivo

A arquitetura do SCPe é composta de cliente, executado nos dispositivos móveis, e um servidor, com base na MoCA (*Mobile Collaboration Architecture*) [SAC04], normalmente executado em uma rede fixa. Essa arquitetura vai integrar o SemantiCore aos serviços pervasivos da MoCA, inserindo a noção de localização aproximada dos agentes que estão executando nos dispositivos móveis, como mostra a Figura 10.

O cliente (rede móvel) é composto pelo SemantiCore Pervasivo e o Monitor da MoCA, esse último, responsável por coletar e divulgar informações de contexto do dispositivo móvel e da rede onde os agentes estão em execução e o servidor (rede fixa) que é composto pelos serviços CIS, LIS e SRM do MoCA. A seguir faremos uma descrição passo a passo dos componentes que integram a arquitetura do SCPe.

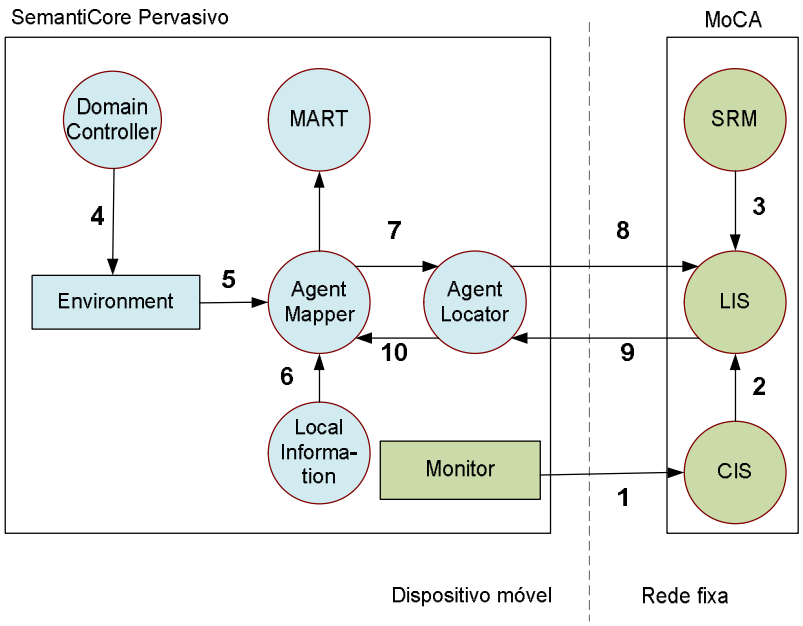

Figura 10: Arquitetura do SemantiCore Pervasivo

Passo 1. Inicialmente, as informações de contexto do dispositivo móvel coletadas pelo Monitor da MoCA são enviadas periodicamente ao servidor MoCA que se encontra na rede fixa, informando sobre qualquer mudança de endereço de rede (IP) ou ponto de acesso (AP) corrente dos dispositivos clientes. O servidor recebe, armazena e processa essas informações de contexto, dos dispositivos móveis, através do serviço CIS.

Passo 2. O serviço CIS repassa essas informações de contexto ao serviço LIS, esse se encarrega de inferir a localização aproximada do dispositivo móvel, através da definição de regiões simbólicas. Segundo [END06], a inferência é realizada através da comparação entre o vetor RSSI (*Received Signal Strength Indication*) corrente do dispositivo (onde cada elemento do vetor representa o sinal de um ponto de acesso 802.11) e os vetores RSSI previamente coletados em "pontos de referência" nas áreas de interesse, e armazenados em uma base de dados do LIS.

Passo 3. O SRM estabelece uma relação com a regiões atômicas definidas pelo LIS exibindo assim a região em que o agente se encontra naquele determinado momento.

Passo 4. O *Domain Controller* é responsável por registrar os agentes do SemantiCore no ambiente (na Figura 10, representado por *Environment*) e também pela recepção de agentes vindos de outro domínios.

Passo 5. O ambiente passa as informações como nome do agente e domínio do agente para o *AgentMapper*.

Passo 6. O *AgentMapper* tem uma função especial no SCPe - ele é encarregado de obter todas as informações necessárias, entre o dispositivo móvel e a rede fixa, para poder inferir localização ao agente. O *LocalInformation* é encarregado de enviar o endereço IP (*Internet Protocol*) juntamente com o endereço MAC (*Media Access Control*) para o componente *AgentMapper*.

Passo 7. O endereço MAC do dispositivo móvel é passado ao *AgentLocator* pelo *AgentMapper*.

Passo 8. O *AgentLocator* é encarregado de enviar o endereço MAC do dispositivo móvel e o IP do servidor MoCA para o serviço LIS.

Passo 9. O serviço LIS publica uma lista contendo o MAC (recebido do Monitor) e a região simbólica onde o dispositivo se encontra. A partir do MAC enviado pelo *AgentLocator*, o LIS consulta a lista comparando o MAC recebido do Monitor com o MAC recebido do *AgentLocator* inferindo assim a localização simbólica ao agente. O LIS envia a região simbólica ao *AgentLocator (passo 9)*.

Passo 10. O *AgentLocator* envia a informação contendo a região simbólica para o *AgentMapper*.

Passo 11. O *AgentMapper* envia as informações do ambiente do SemantiCore juntamente com o endereço MAC (dispositivo móvel), endereço IP (dispositivo móvel) e Região (rede fixa) para o *MappedAgentRoutingTable* (MART) que monta uma lista com informações referentes a localização simbólica de todos os agentes disponíveis na rede móvel. As informações reunidas, em uma única lista, facilitam a consulta e exibição da localização dos agentes e dispositivos móveis.

Para que os agentes possam acessar a sua localização, é necessária a introdução do conceito de região do ambiente onde o agente está executando, como é apresentado a seguir.

4.3.2 Inserindo o conceito de localização nos agentes

A integração entre o *framework* SemantiCore e o *middleware* MoCA surgiu para inserir localização simbólica na plataforma do SemantiCore permitindo que os agentes tenham consciência da região simbólica em que estão executando. A localização é uma das características típicas de aplicações sensíveis ao contexto.

A Figura 11 apresenta o diagrama de classes do modelo conceitual da arquitetura do SemantiCore Pervasivo. A classe *Region* é objetivo do nosso trabalho, as classes *Service* e *Resource* por enquanto existem somente conceitualmente, as classes restantes já existem e fazem parte do modelo conceitual do SemantiCore.

A classe *Fact* representa os recursos (qualquer entidade especificada em uma ontologia[1]) que são publicados no ambiente ou que fazem referência à estrutura interna do agente. Os fatos (classe *Fact*) são usados como padrão de seleção de mensagens nos sensores, como pré e pós-condição das ações e contribuem direta ou indiretamente para a tomada de decisão do agente. A classe *Fact* possui três especializações, *SimpleFact, FunctionBasedFact* e *ComposedFact*.

- *SimpleFact*: os fatos simples representam os elementos da tripla sujeito, predicado, objeto), elemento básico de uma declaração RDF (*Resource Description Framework*).

- *FunctionBasedFact* : os fatos baseados em função representam os fatos que dependem de avaliação de outros fatos para serem gerados. Esses fatos são usados, principalmente, nas regras de inferência.

- *ComposedFact* : os fatos compostos são formados por fatos simples ou compostos ou por fatos baseados em função, ligados por um operador lógico.

Um agente está associado a outras quatro classes que são: *Sensor, Effector, Action* e *Rule*. Cada instância da classe *Sensor* indica um tipo de sensor que é utilizado pelo agente. Cada sensor reconhece um tipo de padrão que tem como domínio a classe *Sensor* e como alvo a classe *Fact*. Todo evento que ocorre no ambiente deve ser verificado, caso o evento tenha o padrão associado ao sensor, ele deve ser processado. Caso contrário, o evento é descartado.

[1] No SemantiCore a ontologia é representada na linguagem OWL (*Web Ontology Language*).

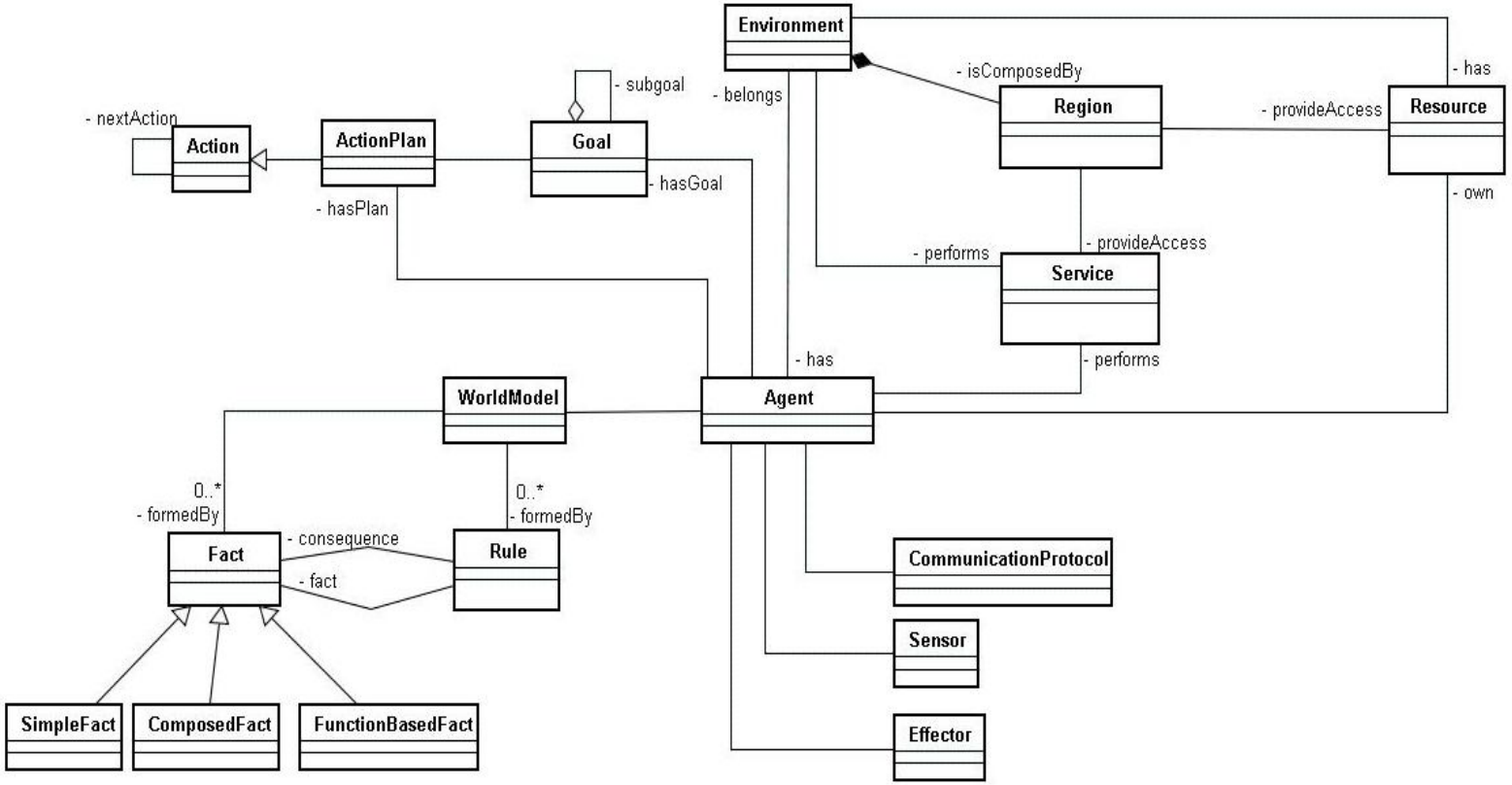

Figura 11: Diagrama de classes conceitual do SCPe

Da mesma forma que nos sensores, cada instância da classe *Effector* indica um efetuador utilizado pelo agente e que deve, portanto, ser instanciado no momento que o agente for reconstruído (os diferentes tipos de sensores e efetuadores devem estar disponíveis para instanciação).

A classe *WorldModel* indica tudo o que o agente acredita como verdade no mundo que o cerca. O *WorldModel* é formado por fatos (classe *Fact*) e regras (Classe *Rule*). A classe *Rule* representa as regras explicitadas em função do conhecimento do domínio e servem de entrada para o mecanismo decisório do agente. As regras possuem dois atributos, *Fact* (que representa fatos ocorridos) e *Consequence* (que representa as conseqüências que a ocorrência de cada fato provoca), que têm como alvo a classe *Fact*. O componente decisório deve utilizar essas regras para da reconstrução do agente.

A classe *Action* é responsável pelas ações que se pode alterar o estado do agente ou do ambiente no qual o agente está inserido. A classe *Action* é a única que possui um atributo do tipo código. Isso se dá porque ações são específicas do domínio do problema e, portanto, não estarão disponíveis para instanciação. Toda ação tem uma pré e uma pós-condição associada que tem como domínio a classe *Action* e como alvo a classe *Fact*. Essas associações indicam que é necessária a ocorrência de um fato para disparar uma ação e também que o término de uma ação implica na criação de um fato novo. Se a ação é a ação principal (ação mais externa), ela esta associada ao objetivo do agente (classe *Goal*). O objetivo é o objetivo geral do objeto de conhecimento (representado como uma ontologia) e indica para qual tipo de problema que o conhecimento encapsulado é voltado. Há também uma associação recursiva na classe *Action*, que representa o encadeamento entre as ações.

A classe *Region* é utilizada para inserir localização simbólica no agente da plataforma, ou seja, a localização física do dispositivo móvel onde o agente está em execução. A partir da consulta a lista criada no ambiente é possível saber quais os componentes estão em execução na plataforma. Para inserir a localização simbólica nos agentes, além das informações da tabela de roteamento, precisamos das informações de contexto do dispositivo móvel onde o agente está executando. A classe *Region* estende as informações contidas na tabela de roteamento com a adição de atributos especiais de contexto juntamente com a região simbólica dos agentes do ambiente. Essa região simbólica representa áreas geográficas (como, salas, corredores, *halls*, etc) que sejam relevantes para a localização do agente. Podemos criar uma aplicação para consultar quais são as regiões simbólicas existentes ou quais agentes estão em uma determinada região. Ela pode ainda receber

notificações sobre a mudança de região de um determinado agente ou a entrada/saída de um agente de uma dada região simbólica.

Cada região pode ter um ou mais serviços associados (classe *Service*) que os agentes poderão utilizar. Alguns serviços podem estar disponíveis somente em determinadas regiões ou para determinados agentes em uma região. Por exemplo, o grupo do ISEG está localizado no andar térreo do prédio 32 da PUCRS, e determinados serviços dessa região só poderiam ser acessados pelos componentes do grupo do ISEG e por mais ninguém.

A região também provê acesso a determinados recursos (classe *Resources*). A região pode delimitar a visibilidade desses recursos no ambiente. Os agentes podem ser proprietários de determinados recursos. Os recursos podem ser base de dados ou programas externos a plataforma.

A classe *Communicator* é responsável pela troca de mensagens entre os agentes distribuídos e meio por onde trafegam as mensagens trocadas entre os diferentes agentes.

No Capítulo seguinte, será apresentada a arquitetura do protótipo do SemantiCore Pervasivo com integração da noção de localização no agente.

5 IMPLEMENTAÇÃO E EXEMPLO DE USO

Para a realização dos experimentos no ambiente do SCPe, foi realizada a implementação do serviço de localização aos agentes do SemantiCore. A implementação do serviço de localização do agente foi realizada usando a linguagem Java uma vez que o SemantiCore e a MoCA também são implementadas nessa linguagem, facilitando integração das plataformas. Para testar a localização dos agentes, foi desenvolvido o protótipo, onde os dispositivos móveis instalados com o SCPe e o *daemon* Monitor da MoCA permitem inferir localização aproximada dos agentes.

Para fazer os testes, precisaríamos de vários dispositivos móveis com acesso a rede sem fio. Para resolver esse problema utilizamos o *Monitor Simulator* da MoCA que simula a utilização de dispositivos móveis em um ambiente previamente configurado. Foi desenvolvido um cenário para que pudéssemos realizar os testes sobre a localização dos agentes.

5.1 Implementação do protótipo

O protótipo do SCPe foi desenvolvido utilizando a linguagem Java. Os serviços e API's do MoCA também foram desenvolvido na linguagem Java, exceto o Monitor que foi desenvolvido na linguagem C++ para o sistema operacional Windows XP. Para permitir a integração entre o SemantiCore e MoCA foram criados componentes de software que permitissem a comunicação entre o middleware pervasivo e a plataforma do SemantiCore. Para isso, criamos algumas classes (Figura 12) para que a integração pudesse ser realizada: PervasiveSemantiCore, MappedAgentRoutingTable, LocalInformation, AgentMapper, PervasiveAgent e AgentLocator.

A classe ***PervasiveSemantiCore*** é a classe principal da arquitetura do SCPe, ela estende as funcionalidades do *framework* SemantiCore. Esta relação é necessária, pois permite que o SCPe tenha acesso aos agentes de software em execução.

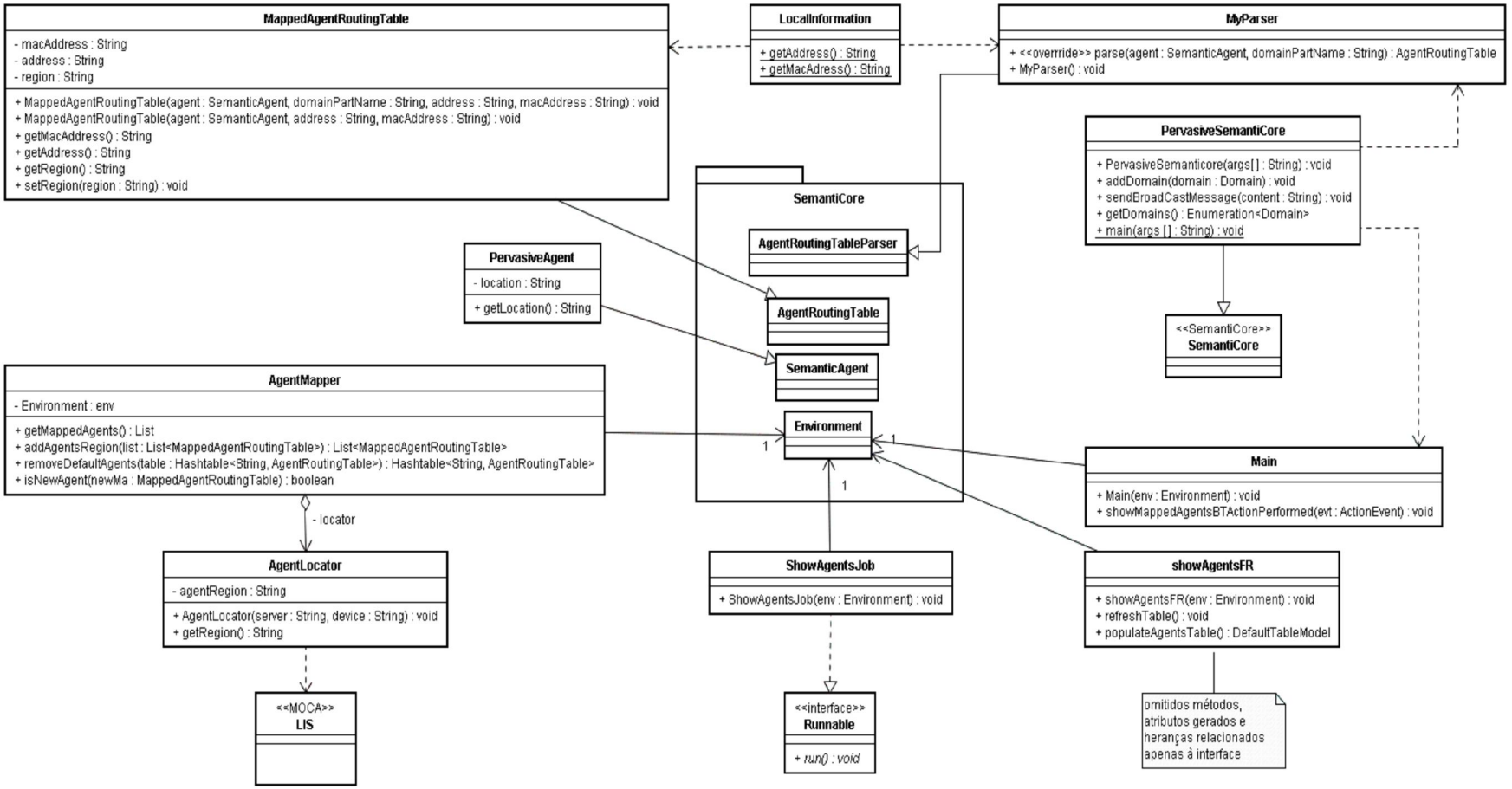

Figura 12: Diagrama de classes do SCPe.

A classe ***MappedAgentRoutingTable*** é uma extensão da classe ***AgentRoutingTable***, reunindo informações da classe ***AgentRoutingTable*** (domínio e nome do agente), adicionando novos atributos com o objetivo de prover as informações adicionais referentes à localização (região) do agente e estado do dispositivo móvel na rede (endereço IP e endereço MAC). A classe ***MyParser*** é um recurso de implementação que faz a classe ***AgentRoutingTableParser*** retornar ***MappedAgentRoutingTable*** adicionando informações adicionais a plataforma do SemantiCore.

As informações do dispositivo móvel (endereços IP e MAC) são obtidos através da classe ***LocalInformation***, que utilizando comandos disponíveis através da API do Java, obtêm estes dados do dispositivo móvel em que o agente está executando. Esta classe mantém uma associação com as classes ***MappedAgentRoutingTable*** e ***MyParser***.

A classe ***AgentLocator*** é responsável por recuperar a região (localização do agente) com uma chamada a classe LIS do MoCA, utilizando o endereço MAC do dispositivo móvel e o endereço IP do servidor do serviço LIS. O serviço LIS confere se o endereço MAC, passado pelo ***AgentLocator***, está registrado na sua base de dados, recuperando assim a região onde o dispositivo móvel que contêm o agente em execução está localizado.

A classe ***AgentMapper*** é responsável por retornar uma lista contendo todos os agentes mapeados na rede, isto é possível através do método ***getMappedAgents***. O método ***getMappedAgents*** recupera os dados referentes aos agentes presentes no ambiente, além de remover os agentes padrão do *framework* (que não interessam à lista), e adiciona a localização dos agentes. A classe ***PervasiveAgent*** é uma extensão da classe ***SemanticAgent,*** ela é utilizada para inserir a localização corrente do agente, tem como retorno uma string contendo a localização do agentes no ambiente pervasivo.

A classe ***ShowAgentsJob*** é a tarefa disparada pela *thread* que fica atualizando a lista de agentes, em segundo plano, por um período de tempo.

As classes **showAgentFR** e ***Main*** são responsáveis por gerar as interfaces gráfica do SCPe.

Na próxima seção será descrito um exemplo de cenário para ilustrar a utilização do SCPe na inserção de localização aos agentes em execução.

5.2 Cenário

Para simular a aplicação desenvolvida utilizamos como cenário o prédio 32 da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), prédio onde funciona a Faculdade de

Informática (FACIN), com seus cursos de graduação e pós-graduação. Para fins de testes fizemos o mapeamento dos andares térreo e quinto andar. Cada um dos referidos andares foram divididos em áreas, como mostra a Figura 13. Essas áreas servem de referência para a aplicação da localização dos agentes dentro de uma determinada região, no caso a região do prédio 32. Por exemplo, se o dispositivo com o SCPe estiver conectado ao ponto de acesso (AP) próximo a região do Auditório (indicado pelo número 3 na Figura 13) ele deverá indicar que o agente está na área do Auditório da região do prédio 32.

O andar térreo do prédio 32 foi dividido em oito áreas e atribuímos nomes a essas áreas como segue: 1 - Centro Acadêmico; 2- Bar 32; 3 - Auditório; 4 - Escada/Elevador; 5 - Convênio PUCRS-DELL; 6 - Saguão; 7 - Convênio PUCRS-HP; 8 - Frente Prédio 32.

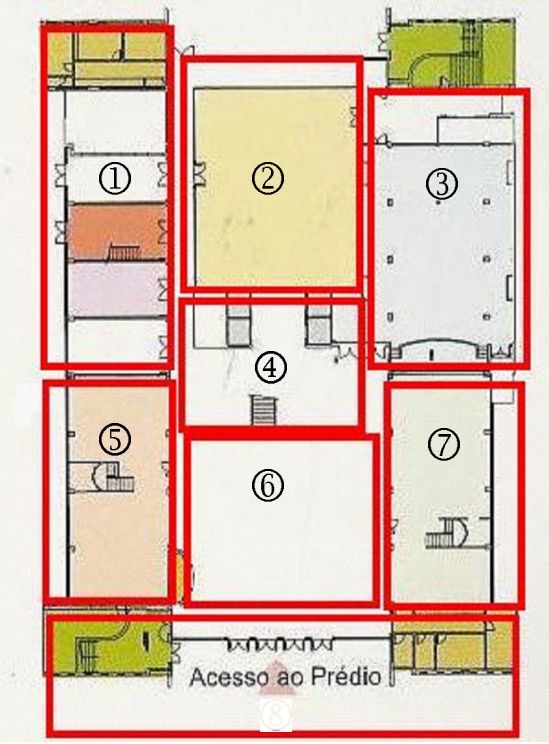

Figura 13: Áreas do andar Térreo do prédio 32(FACIN)

O quinto andar (Figura 14), do prédio 32, foi dividido em quatro áreas, como segue: 1 - Salas de aula PPGCC; 2 - PPGCC; 3 - Direção e 4 - Graduação.

A partir da definição das áreas podemos então montar os arquivos de configuração do *Monitor Simulator*.

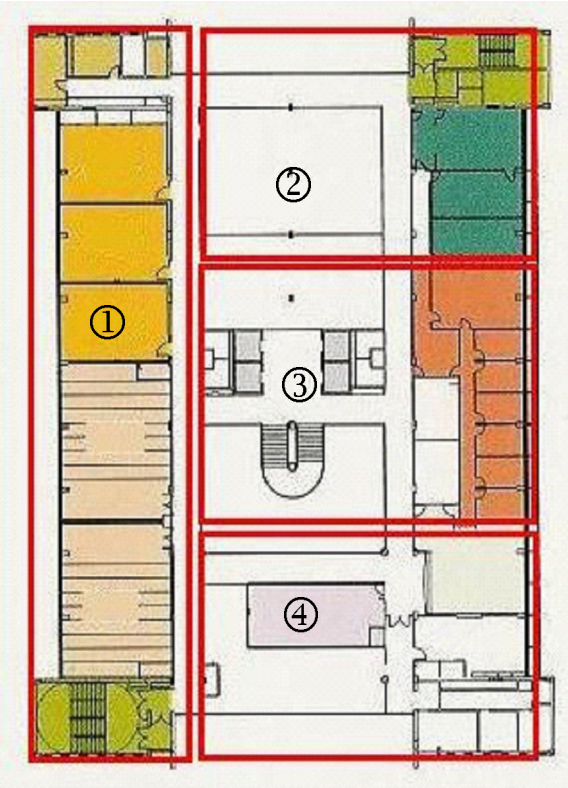

Figura 14: Áreas do quinto andar do prédio 32(FACIN)

5.3 Monitor Simulator

O *Monitor Simulator*, como vimos no capítulo anterior na seção 4.2, simula o *daemon* Monitor instalado nos dispositivos móveis. Para utilizarmos o *Monitor Simulator* podemos utilizar os arquivos de *scan* da MoCA já configurados ou configurar nossos próprio arquivos de testes. Os arquivos de *scan*, são do tipo texto (txt) e contém as seguintes informações (Figura 15): CPU, memória livre, nível de energia, periodicidade, mudança de IP, mudança de AP, endereço IP, máscara de rede, endereço MAC, MAC do AP atual, MAC do AP corrente e MAC dos APs que o dispositivo está recebendo sinal.

```
CPU#Free Memory#EnergyLevel#Periodicity#IPChange#APChange#IP#MASK# MobileHost's
MacAddr#Current AP's MacAddr&AP1&AP2&AP3&APn
```

Figura 15: Estrutura do arquivo de scan

Para enviar os arquivos de *scan* para o serviço CIS da MoCA precisamos adicionar os arquivos de *scan* ao arquivo de configuração do *Monitor Simulator*, como mostra a Figura 16. O conteúdo do arquivo foi modificado para representar os endereços IP corretos e números de porta dos serviços escolhidos pelo usuário, trazendo informações necessárias para o envio ao serviço CIS. As informações serão enviadas ao CIS a cada em um intervalo de um segundo (cada intervalo de 1000 representa 1 segundo), como mostra a linha 1 da Figura 16.

```
 1 monitor.scanInterval=1000
 2 monitor.repeating=true
 3 cis.server.host=127.0.0.1
 4 cis.monitor.port=55010
 5 file1=conf/Scan-Salas_PPGCC.txt
 5 interval1=5000
 6 file2=conf/Scan-DELL_01.txt
 7 interval2=1000
 8 file3=conf/Scan-Graduacao.txt
 9 interval3=2000
10 file4=conf/Scan-PPGCC_01.txt
11 interval4=5000
12 file5=conf/Scan-wifi_02.txt
13 interval5=1000
14 file6=conf/Scan-PPGCC_02.txt
15 interval6=2000
16 file7=conf/Scan-DELL_02.txt
17 interval7=5000
18 file8=conf/Scan-Salas_PPGCC.txt
19 interval8=2000
```

Figura 16: Arquivo de configuração do Monitor Simulator

A linha 2 representa se desejamos que o arquivo seja repetido, no caso de *true* o arquivo fica em *looping*, enviando as informações ao serviço CIS em um intervalo de tempo predefinido. Por exemplo, o arquivo “Scan-DELL_02.txt” (linha 5), será enviado por 5 segundos (ou 5 exames) a cada repetição, este arquivo indica que o dispositivo está conectado à área do Convênio PUCRS-DELL (área 5 da Figura 13) definida na seção anterior.

As linhas 3 e 4 respectivamente representam o local onde o servidor do serviço CIS está em execução através do endereço IP, no caso “127.0.0.1”, e a porta do servidor CIS para onde os dados da simulação de contexto serão enviados. O *Monitor Simulator* pode utilizar uma interface gráfica para facilitar a descrição do comportamento do dispositivo móvel em simulação, chamada de *Monitor Simulator Gui*.

5.4 Configurando o MoCA

O arquivo de configuração do serviço CIS, Figura 17, é composto pelo endereço do servidor e as portas de comunicação. O servidor do serviço CIS (linha 4) se encontra no servidor de endereço “127.0.0.1”, que significa que o servidor está *localhost*. A linha 1, apresenta a porta que as informação são recebidas do Monitor. Na linha 2 temos a porta que comunica o servidor sobre eventos como, mudança de área do dispositivo ou eventos em qualquer dispositivo que entra ou saia de uma determinada região. A linha 3 representa a porta de comunicação com o servidor do serviço CIS.

```
1 cis.monitor.port=55010
2 cis.event_server.port=55000
3 cis.server.port=55001
4 cis.server.address=127.0.0.1
```

Figura 17: Arquivo de configuração do CIS

O serviço LIS possui três arquivos de configuração: `lis.properties`, `cis.properties` e `srm.properties`. O `lis.properties`, exibido na Figura 18, as configurações das portas são previamente definidas e não é aconselhável alterá-las. Devemos ficar atentos ao endereço do servidor do serviço LIS (linha 1), que foi definido como “127.0.0.1”.

```
1 lis.server.host=127.0.0.1
2 lis.server.port=55021
3 lis.publisher.port=55020
```

Figura 18: Arquivo de configuração do LIS (lis.properties).

O serviço CIS também deve ser configurado no LIS para a troca de mensagens entre os serviço, como exibido na Figura 19.

```
1 cis.server.host=127.0.0.1
2 cis.server.port=55001
3 cis.publisher.port=55000
4 cis.monitor.port=55010
```

Figura 19: Arquivo de configuração do LIS (cis.properties).

Para que se tenha a verificação da hierarquia das regiões atômicas o arquivo `srm.properties` deve ser configurado como na Figura 20.

```
1 srm.server.host=127.0.0.1
2 srm.query.port=55030
3 srm.event.port=55031
```

Figura 20: Arquivo de configuração do LIS (srm.properties).

O serviço SRM define a hierarquia das áreas definidas a qual serão utilizadas para definir a localização dos usuários. A hierarquia é colocada em um arquivo XML contendo a hierarquia das áreas disponíveis para o usuário. A Figura 21, mostra parte da descrição de um arquivo XML com a hierarquia entre as regiões definidas. Dentro de uma região podemos ter outras regiões (sub-regiões). A região definida na Figura 21é chamada de PUCRS que possui outras três sub-regiões: Prédio 30, Prédio 40 e Prédio 32. Dentro da região Prédio 32 temos as sub-regiões definidas para o Prédio 32, neste trabalho definimos duas sub-regiões: “Terreo” e “5º Andar”. As sub-regiões “Terreo” e “5º Andar” por sua vez possuem sub-regiões como vimos na seção 5.2. Por exemplo, dada a hierarquia “PUCRS/Predio 32/Terreo/DCE”, através da consulta ao serviço LIS saberíamos que o dispositivo se encontra localizado na “PUCRS”, “Predio 32”, no andar “Terreo” mas precisamente na região definida como DCE.

```
<?xml version="1.0" standalone="yes" ?>
- <HierarchyXML>
  <id>PUCRS</id>
  - <root>
    <name>PUCRS</name>
    <parent>null</parent>
    - <subRegions>
      - <region>
        <name>Predio 30</name>
        <parent>PUCRS</parent>
        <subRegions />
      </region>
      - <region>
        <name>Predio 40</name>
        <parent>PUCRS</parent>
        <subRegions />
      </region>
      - <region>
        <name>Predio 32</name>
        <parent>PUCRS</parent>
        - <subRegions>
          - <region>
            <name>Terreo</name>
            <parent>Predio 32</parent>
            - <subRegions>
              - <region>
                <name>DCE</name>
                <parent>Terreo</parent>
                <subRegions />
              </region>
```

Figura 21: Hierarquia descrita no serviço SRM

5.5 Exemplo de uso do protótipo

A arquitetura do SCPe permite a noção de inserção de localização dos agentes de um domínio, facilitando assim a construção de aplicações para implementar serviços baseados em localização. Para ilustrar, desenvolvemos uma aplicação exemplo utilizando o cenário descrito na seção 5.2.

A partir do cenário descrito do protótipo criamos uma aplicação para a localização do professor ou dos alunos de uma determinada disciplina. Vamos supor que o professor da disciplina de Algoritmos do curso de Ciência da Computação da PUCRS pediu a seus alunos que fizessem um trabalho final da disciplina e que o mesmo fosse entregue impresso ao professor na data determinada. Um aluno chegou à universidade para entregar o trabalho ao professor fora do horário de aula. Para localizar o professor utilizou o aplicativo de localização utilizado pelo professor e pelos alunos da disciplina de Algoritmos. O aluno utilizou seu dispositivo móvel com o SCPe, Monitor e o aplicativo de localização instalados e fez uma busca pelo nome do agente do professor para ver se o mesmo se encontrava na universidade. A partir da busca conseguiu verificar que o professor estava na sala dos professores da graduação, localizada no sexto andar do prédio da universidade. Sabendo onde o professor se encontrava conseguiu entregar o trabalho final no prazo determinado.

O aplicativo consiste em localizar o professor e/ou alunos matriculados na disciplina, esse aplicativo chamamos de "*Locate Agent*". Para isso foi criado um domínio no SCPe para a disciplina de Algoritmos onde professor e alunos possuem um agente pessoal que, além da troca de mensagens entre eles, serve também para localizá-los no ambiente do prédio 32 da PUCRS.

Abaixo seguem as características do ambiente de desenvolvimento utilizado na construção do aplicativo:

- Linguagem de programação Java;

- Ambiente de desenvolvimento NetBeans;

- Editor de XML (Peters XML);

- Ferramenta Mapper (criação de mapas do serviço LIS para inferir a localização).

O exemplo ilustra os agentes em um domínio do SCPe em execução em um dispositivo móvel que tenha placa de rede wireless padrão IEEE 802.11. As informações de contexto do

dispositivo móvel onde o agente está em execução são enviadas ao servidor do MoCA, mais especificamente ao serviço CIS. A partir das informações de contexto é possível inferir a localização aproximada do agente em um ambiente.

Antes de iniciar a execução dos serviços do MoCA é necessário que os arquivos dos serviços CIS, LIS, SRM e *Monitor Simulator* estejam configurados. Com os arquivos configurados, podemos iniciar a execução desses serviços no servidor. Como vamos fazer uma simulação, é preciso configurar também os arquivos do *Monitor Simulator*. É recomendado seguir uma ordem na execução dos serviços devido a uma série de dependências funcionais entre os serviços que o usuário deverá observar antes de iniciar cada um deles [VIT06b]. O *Monitor Simulator* pode ser o primeiro a ser executado, seguido pelos serviços CIS, SRM e por último o LIS. Nas seções anteriores vimos como configurar esses serviços.

Após a inicialização dos serviços do MoCA no servidor, é hora de iniciar a execução dos agentes da plataforma do SemantiCore. Os agentes da plataforma, como vimos anteriormente, se registram na plataforma do SCPe através do Controlador de Domínio.

O aplicativo desenvolvido como exemplo do funcionamento do protótipo é composto por três agentes executando na plataforma do SCPe: o agente do professor chamado “ag_Prof” e os agentes dos alunos, “ag_Xiru” e “ag_Tche”. O agente “ag_Prof” é o agente principal da plataforma executado no domínio “ALGORITMO”, como mostra a Figura 22.

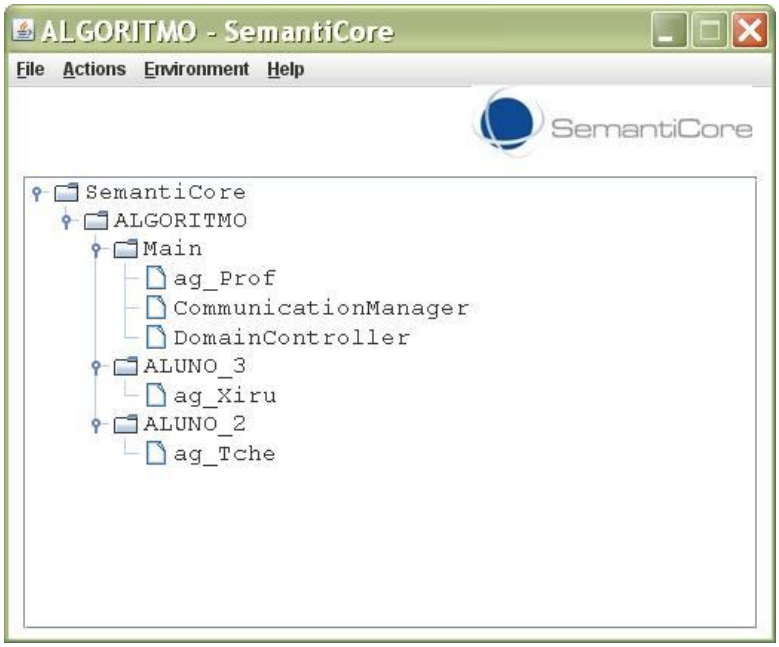

Figura 22: Domínio ALGORITMO com os agentes em execução.

Os agentes dos alunos devem se registrar na plataforma através de um subdomínio, para cada aluno da disciplina é criado um subdomínio diferente, no exemplo o “ag_Tche” pertence ao subdomínio “ALUNO_2” e o “ag_Xiru” pertence ao subdomínio “ALUNO_3”.

Para que a plataforma funcione corretamente é preciso que o domínio “ALGORITMO” esteja em execução, ou seja, o “ag_Prof” deve estar em execução. A partir da execução do agente além da janela da plataforma do SCPe será criada uma outra janela (Figura 23) contendo dois botões: “Locate *agent*” e “*exit*”.

Figura 23: Janela para exibição dos agentes mapeados

O botão “Exit” serve apenas para sair do aplicativo que utiliza a plataforma do SCPe enquanto que o botão “*Locate agent*” vai exibir a janela do aplicativo “*Locate Agent*”, que exibirá uma janela (Figura 24) onde possui um campo para a pesquisa do agente pretendido e o botão “Localizar” para confirmar a pesquisa. A pesquisa é feita pelo nome do agente registrado na plataforma.

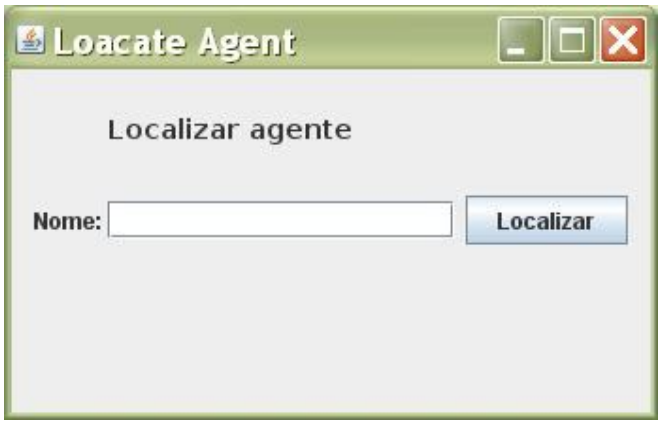

Figura 24: Janela do aplicativo Locate Agent

Os agentes “ag_Xiru” e “ag_Tche”, como vimos anteriormente, pertencem respectivamente aos subdomínios “ALUNO_2” e “ALUNO_3” que estão executando em um mesmo domínio “ALGORITMO”. Estes agentes estão executando em um mesmo domínio, mas em dispositivos

móveis diferentes. A Figura 25 exibe os agentes do subdomínio “ALUNO_2” em execução na plataforma do SCPe.

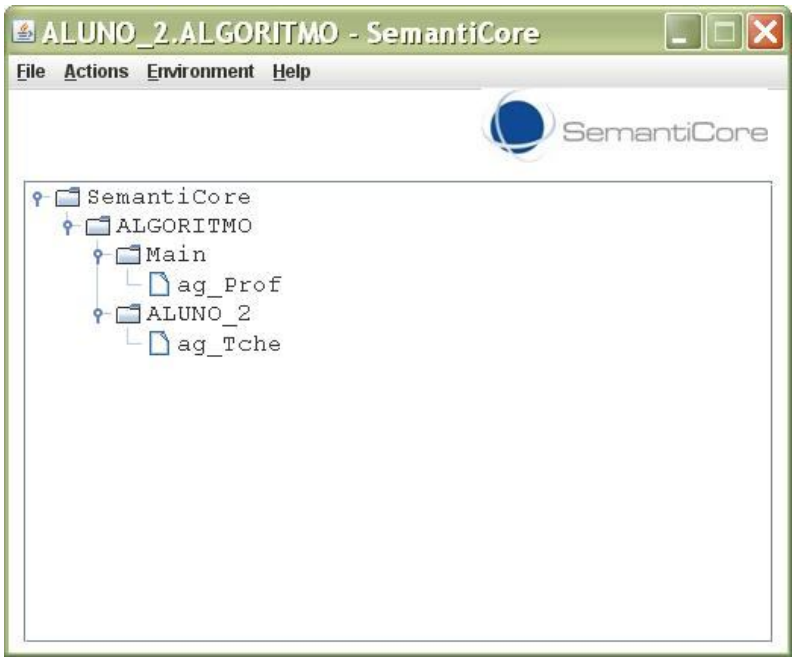

Figura 25: Subdomínio ALUNO_2 com o agente em execução

A Tabela 2 mostra as informações referentes aos agentes e características dos dispositivos utilizados para simular o funcionamento de um ambiente distribuído com vários dispositivos espalhados no ambiente.

Tabela 2: Características dos agentes e dispositivos do exemplo

Agente		Dispositivo	
Nome	Domínio	MAC	IP
ag_Prof	ALGORITMO	00:02:2D:A5:06:01	10.10.10.1
ag_Tche	ALUNO_2.ALGORITMO	00:02:2D:A5:06:02	10.10.10.2
ag_Xiru	ALUNO_3.ALGORITMO	00:02:2D:A5:06:03	10.10.10.3

As informações sobre os dispositivos são colocadas em arquivos de *scan* (Figura 26), que além dos endereços IP e MAC, contêm outras informações como: CPU, memória livre, nível de energia, periodicidade, mudança de IP, mudança de AP, endereço IP, máscara de rede, endereço

MAC, MAC do AP atual, MAC do AP corrente e MAC dos APs que o dispositivo está recebendo sinal.

Figura 26: Arquivo de configuração do Monitor Simulator Gui.

A Figura acima representa a configuração de um arquivo de *scan* do dispositivo em que estão executando os agentes “ag_Tche” e “ag_Xiru”. O *Monitor Simulator* envia as informações de contexto dos dispositivos para o serviço CIS do MoCA com uma periodicidade predefinida no arquivo de *scan*. O CIS envia as informações de contexto dos dispositivos móveis (simulados) para o LIS que consulta o mapa de regiões para inferir a localização simbólica aos dispositivos em execução. O LIS faz a publicação das informações do dispositivo no servidor do MoCA.

Para vermos o resultado da pesquisa, devemos inserir o nome do agente no campo “Nome” e pressionar o botão “Localizar”, como vimos na Figura 24. O resultado da pesquisa será exibido na mesma janela. O agente poderá ser encontrado no ambiente pervasivo ou o agente pesquisado pode não ser encontrado no ambiente exibindo uma mensagem para cada um dos casos. Na Figura 27 podemos ver o agente “ag_Prof” sendo pesquisado por um dos alunos da disciplina de Algoritmo. Se o agente do professor for encontrado, será exibida a localização do agente do professor no ambiente pervasivo, caso contrário, será exibida a mensagem “ag_Prof não encontrado”. Na Figura 27 podemos observar que a pesquisa feita pelo agente do professor (ag_Prof) foi encontrado na

região “Graduacao” definidas na seção 5.2, mais precisamente no sexto andar do prédio 32 da PUCRS, salas definidas como “Graduacao”.

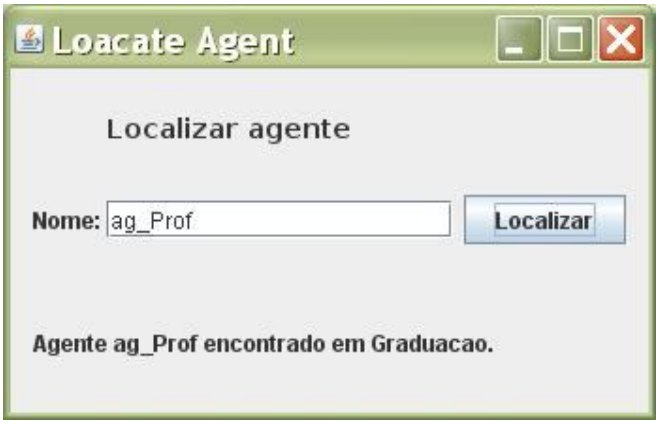

Figura 27: Pesquisa do agente do professor realizada no *Locate Agent*

O agente a ser pesquisado pode não ser encontrado na plataforma por motivo de erro de digitação por parte do usuário ou pelo fato do agente não estar em execução na plataforma, nestes casos, é emitido uma mensagem que o agente não foi encontrado, como mostra a Figura 28. Neste caso o agente “ag_Tche” não foi encontrado por não estar em execução na plataforma.

Figura 28: Agente não encontrado no ambiente

A consulta de um determinado agente é feita através da pesquisa ao agente principal da plataforma, o qual monta uma lista com todos os agentes em execução na plataforma juntamente com sua respectiva localização do agente no ambiente pervasivo. O nome do agente digitado é comparado com os nomes dos agentes da lista "*Mapped Agents*", se o agente se encontra nesta lista é capturado o campo *Region* que contém a localização simbólica do agente no ambiente pervasivo. O conteúdo do campo "*Region*" é encaminhado ao agente que fez a consulta.

Na Figura 29, podemos ver o mapeamento de todos os agentes pertencentes ao domínio "ALGORITMO" com seus atributos: nome do agente, o domínio que o agente pertence, o endereço IP e MAC do dispositivo onde o agente está em execução e à região simbólica do agente, ou seja, a localização do agente em uma determinada região física já predefinida. O botão "*Close*" é utilizado somente para fechar a janela.

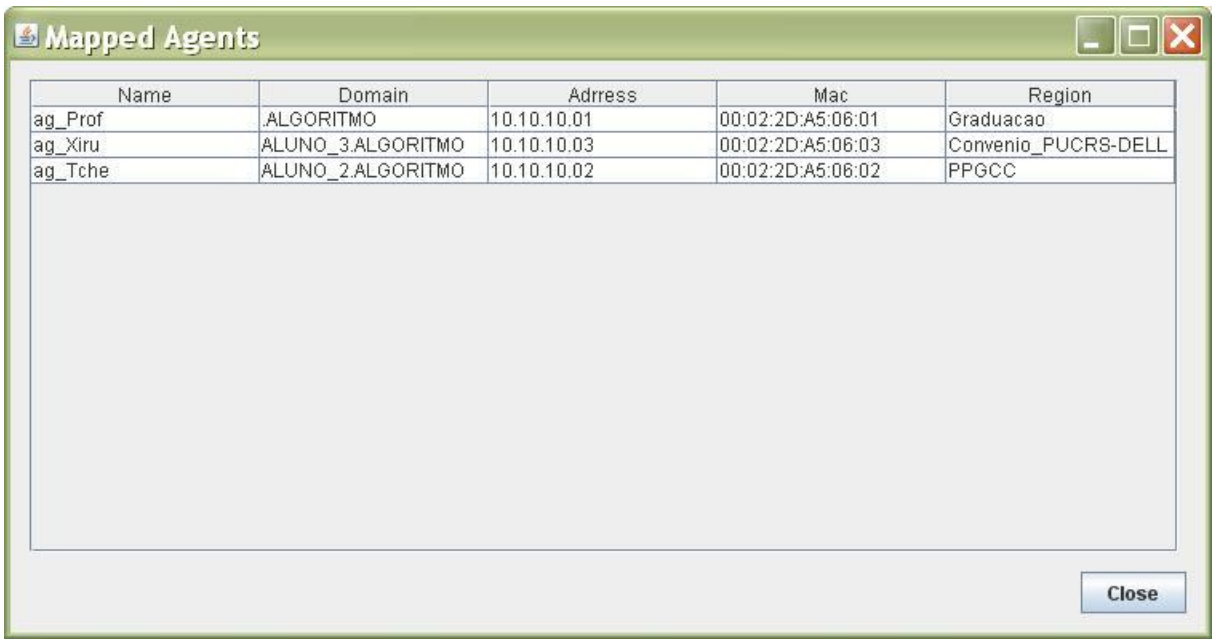

Figura 29: Agentes em execução na plataforma do SCPe com localização.

A Figura acima, mostra o agente "ag_Prof" pertencente ao domínio "ALGORITMO" em execução no dispositivo móvel com endereço IP número "10.10.10.1" e endereço MAC "00:02:2D:A5:06:01" e está na região pertencente ao sexto andar do prédio 32 da PUCRS, mais precisamente na sala da Graduação. O agente "ag_Tche" pertence ao subdomínio "ALUNO_2" está em execução no dispositivo móvel com o endereço IP número "10.10.10.2" e endereço MAC "00:02:2D:A5:06:02" e está na região pertencente a sala definida como PPGCC (Programa de Pós-Graduação em Ciência da Computação). Já o agente "ag_Xiru" pertence ao subdomínio

“ALUNO_3” e está em execução no dispositivo móvel com endereço IP número “10.10.10.3” e endereço MAC “00:02:2D:A5:06:03”, e está na região pertencente ao andar Térreo do prédio 32 da PUCRS, mais precisamente, localizado na sala definida como Convênio PUCRS-DELL.

Durante a descrição do exemplo de uso, não mencionamos sobre a obtenção do mapa com os pontos de inferência do serviço LIS, eles foram obtidos através do mapa baixado do próprio LIS. Para simulação foram feitos alguns ajustes no arquivo do mapa para ficar condizente com o cenário descrito. Cabe ressaltar que para a obtenção do mapa contendo os padrões de radiofreqüência observados em pontos de referência definidos, deve-se usar a ferramenta *Mapper*. O maior esforço para o desenvolvedor é obter das áreas mapeadas os sinais de radiofreqüência, para isso é utilizado *daemon* do MonitorXP.

Com o exemplo de uso descrito neta seção, procuramos mostrar as primitivas do modelo proposto e a forma de utilizá-las na obtenção da noção de localização do agente dentro de um ambiente pervasivo. A ideia principal de integração do *middleware* pervasivo, no caso o MoCA sobre a plataforma SemantiCore através da localização do agente foi conseguida e demonstrado no exemplo de uso.

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho apresentamos uma proposta de integração entre duas tecnologias emergentes na área computacional, os sistemas multiagentes e a computação pervasiva. O entendimento e a utilização desses dois paradigmas juntos vêm ao encontro ao desenvolvimento de novas tecnologias na área computacional. A meta do trabalho foi inserir localização no agente do SemantiCore, integrando assim as informações de contexto extraídas do ambiente com os agentes atuantes em um domínio.

Além da proposta de integração, este trabalho também agrega como contribuição uma revisão na literatura sobre agentes de software, sistemas multiagentes e computação pervasiva que nos deu embasamento teórico para a realização do trabalho. Com isso, podemos perceber que novas tecnologias, métodos, linguagens de modelagem, plataformas de desenvolvimento, ferramentas, linguagens de programação, plataforma de *middlewares* e aplicações pervasivas formam um ambiente complexo e cheio de desafios.

Apresentamos também alguns trabalhos relacionados que utilizaram no seu desenvolvimento os conceitos de agentes de software e computação pervasiva. As arquiteturas estudadas estão baseadas em agentes para dar apoio a sistemas pervasivos sensíveis ao contexto. A fusão dessas duas tecnologias surge como um modelo computacional para integrar de forma transparente os componentes de hardware e software e auxiliar a interação entre pessoas em um ambiente.

Como foi visto os agentes podem se comunicar entre si para adquirir conhecimento para atender seus objetivos. Com isso, em um ambiente pervasivo, podemos incorporar aos agentes atributos relacionados à percepção do contexto onde está inserido. As informações de contexto, em especial a localização de um dispositivo móvel, pode ser útil ao agente para descobrir os recursos que ele tem disponíveis para uso ou então descobrir serviços de seu interesse.

O modelo conceitual proposto juntamente com a implementação do SCPe e o exemplo de uso constitui a principal contribuição deste trabalho. O SCPe permite que o agente tenha a noção de localização simbólica em um ambiente físico. A inserção da localização só foi possível pois utilizamos o *middleware* MoCA para obter as informações de contexto do dispositivo móvel onde o agente se encontra em execução. Entre as várias informações de contexto do dispositivo móvel delimitamos o nosso tema em usar apenas a localização do dispositivo móvel. Com isso conseguimos inserir a localização no agente da plataforma do SCPe, a partir da localização do

agente podemos desenvolver aplicações para ambientes pervasivos utilizando a plataforma do SCPe.

Dessa forma, como trabalhos futuros fica o desafio ao grupo do ISEG dar continuidade ao trabalho procurando integrar as informações de contexto, que não foram exploradas neste trabalho ao SCPe. Com isso poderíamos criar um agente ou serviço que faça o gerenciamento das informações de contexto fazendo com que: responda a eventos quando for detectada uma mudança de região; armazene o caminho percorrido pelo agente; consulte as regiões próximas ou ao redor do agente; consulte quais agentes estão em uma mesma região. Por fim, seria possível ainda criar um mecanismo para garantir a política e privacidade das informações de contexto de um usuário ou um agente e a possibilidade dessas informações serem compartilhadas.

REFERÊNCIAS BIBLIOGRÁFICAS

[BEL07] BELLIFEMINE, F; Caire, G.; GREENWOOD, D. “Developing Multi-Agent Systems with JADE”. England: John Wiley & Sons, 2007, 286p.

[BON05] BONATTO, D.; BARBOSA, J.; CAVALHEIRO, G. G. H.; RAMOS, J. D. G. PHolo: “Uma Arquitetura para a Computação Pervasiva utilizando o Holoparadigma”. In: 6º Workshop de Sistemas de Computação de Alto Desempenho, WSCAD, 2005, 8p.

[BOR03] BORDINI, R.H., VIEIRA, R. “Linguagens de Programação Orientadas a Agentes: uma introdução baseada em AgentSpeak(L)”. Capturado em: http://www.inf.ufrgs.br/~revista/docs/rita10/rita_v10_n1_p7a38.pdf, Fevereiro 2008.

[BOR07] BORDINI, R. H.;HÜBNER, J. F. A.; WOOLDRIDGE, M. “Programming multi-agent systems in AgentSpeak using Jason”. England: John Wiley & Sons, 2007, 273p.

[BRE98] BRENNER, W.; ZARNEKOW, R.; WITTIG, H. “Intelligent Software Agents: Foundations and Applications”. New York: Springer-Verlag, 1998, 333p.

[CAI02] CAIRE, G; CHAINHO, F.; EVANS, R. “Agent-oriented analysis using Message/UML”. In: Agent-Oriented Software Engineering, WOOLDRIDGE, M.; WEISS, G.; CIANCARINI, P.(Eds). Second International Workshop, AOSE 2001, LNCS 2222 Stringer, Canada, 2002, pp. 119-135.

[CHE04] CHEN, H. “An Intelligent Broker Architecture for Pervasive Context-Aware Systems”, PhD thesis, Department of Computer Science, University of Maryland, Baltimore County, 2004, 121p.

[CHO04] CHOREN, R.; LUCENA, C. J. P. “Modeling Multi-agent systems with ANote”. Software and Systems Modeling, Springer Berlin/Heidelberg, vol. 4-2, May 2005, pp. 199-208.

[COS08] COSTA, C.A., Yamin, A.C., Geyer, C.F. "Toward a General Software Infrastructure for Ubiquitous Computing," IEEE Pervasive Computing, vol. 7-1, Jan-Mar 2008, pp. 64-73.

[DEL99] DELOACH, S. A. “Multiagent Systems Engineering: a Methodology and Language for Designing Agent Systems”. In: Agent-Oriented Information System (AOIS99), 1999, 9p.

[DEY00] DEY, A. K. “Providing Architectural Support for Building Context-Aware Applications”, Ph.D. Thesis, Georgia Institute of Technology, 2000, 170p.

[END06] ENDLER, M.; SACRAMENTO, V.; RUBINSZTEJN, H.; NASCIMENTO, F. N.; ROCHA, R.; VITERBO, J. F.; BAPTISTA, G. L. “Experiências no Desenvolvimento de uma Arquitetura de Middleware para Ciência de Contexto”, Monografias em Ciência da Computação n. 37/06, Editor: Prof. Carlos José Pereira de Lucena, PUC-Rio, 2006, 16p.

[ESC06] ESCOBAR, M.; LEMKE, A. P.; RIBEIRO, M.B. “SemantiCore 2006 – Permitindo o Desenvolvimento de Aplicações baseadas em Agentes na Web Semântica”, Estudo desenvolvido pelo Intelligent Systems Engineering Group da PUCRS. Programa de

Pós-Graduação em Ciência da Computação – Faculdade de Informática – PUCRS, 2006, 11p.

[FIP08] FIPA. “The Foundation for Intelligent Physical Agents”. Capturado em: http://www.fipa.org/, Novembro, 2008.

[GON04] GONÇALVES, K,; RUBINSZTEJN, H.K.; ENDLER, M.; Santana, B.; Barbosa, S.D.J. “Um aplicativo para comunicação baseada em localização”, In: 6°. Workshop de Comunicação sem Fio e Computação Móvel (WCSF 2004), 2004, pp. 225-231.

[GRI04] GRIMM, R. “One.world: Experiences with a pervasive computing architecture”. IEEE Pervasive Computing, vol. 3-3, July-Sept 2004, pp. 22–30.

[HIG01] HIGHTOWER, J.; BORRIELLO, G. "Location Systems for Ubiquitous Computing," Computer, vol. 34-8, Aug. 2001, pp. 57-66.

[JAD08] JADE “Java Agent DEvelopment Framework”. Capturado em: http://jade.tilab.com/, Novembro 2008.

[JEN09] JENA website. Capturado em: http://jena.sourceforge.net/, Agosto 2009.

[KHE05] KHEDR, M.; KARMOUCH, A. “ACAI: Agent-Based Context-Aware Infrastructure for Spontaneous Applications”. Journal of Network and Computer Applications, 28, 2005, pp. 19-44.

[LIN01] LIND, J. “Issues in agent-oriented software engineering”. In: Agent-Oriented Software Engineering, vol. 1957, 2001, pp.45-48.

[MAE95] MAES, P. “Intelligent Software: Easing the Burdens that Computers put on people”. In: Scientific American, vol. 273-3, Sept 1995, pp. 84-86.

[MAE97] MAES, P. “On software agents: humanizing the global computer”. IEEE Internet Computing, July-August 1997, 10p.

[MYL01] MYLOPOULOS, J., KOLP, M., AND CASTRO, J. “UML for Agent-Oriented Software Development: The Tropos Proposal”. In: Proceedings of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools, 2001, pp.422-441.

[ODE00] ODELL, J.; PARUNAK, H.; BAUER, B. “Extending UML for Agents”. In: Anais of the Agent-Oriented Information Systems Workshop, 2000, pp. 3-17.

[PAD02] PADGHAM, L; WINIKOFF, M. “Prometheus: A Methodology for Developing Intelligent Agents”, In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems, 2002, 12p.

[PES06] PESSOA, R.M. “Infraware: Um Middleware de Suporte a Aplicações Sensíveis ao Contexto”. Dissertação Mestrado, Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, 2006, 139p.

[RIB02] RIBEIRO, M. B. “Web Life: Uma arquitetura para a implementação de sistemas multi-agentes para a Web”, Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2002, 204 p.

[RIB04] RIBEIRO, M. B.; LUCENA, C. “Multi-Agent Systems and the Semantic Web – The SemanticCore Agent-based Abstraction Layer”. In: Proceedings of Sixth

International Conference on Enterprise Information Systems ICEIS, 2004, pp.263-270.

[RIE07] RIES, L. H.; HESSEL, F.P. “Uma plataforma para integrar dispositivos eletrônicos em ambientes pervasivos”, Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2005, 80p.

[RUS04] RUSSEL, S; NORVIG, P. “Inteligência Artificial”. Rio de Janeiro: Elseiver, 2004, 1021p.

[SAC04] SACRAMENTO, V., ENDLER, M., RUBINSZTEJN, H.K. , LIMA, L.S. , GONÇALVES, K., NASCIMENTO, F.N.do , BUENO, G. , “MoCA: A Middleware for Developing Collaborative Applications for Mobile Users”. IEEE Computer Society, vol. 5-10, October 2004, 14p.

[SAH03] SAHA, D; MUKHERJEE, A. “Pervasive computing: a paradigm for the 21st century,”. IEEE Computer Society, vol. 36-3, March 2003, pp. 25–31.

[SAT01] SATYANARAYANAN, M. “Pervasive computing: Vision and challenges,” IEEE Personal Communication, vol. 8, no. 4, pp. 10–17, August 2001.

[SIL03] SILVA, V.; GARCIA, A.; BRANDAO, A.; CHAVEZ, C.; LUCENA, C.; ALENCAR, P. "Taming Agents and Objects in Software Engineering" In: Software Engineering for Large-Scale Multi-Agent Systems, vol. 2603, 2003, pp. 1-26.

[SIL04] SILVA, V., T.; LUCENA, C., J. “Uma linguagem de modelagem para sistemas multi-agentes baseada em um framework conceitual para agentes e objetos”. Tese de Doutorado, Programa de Pós-Graduação em Informática, PUC-Rio, 2004, 252p.

[SOL07] SOLDATOS, J., PANDIS, I., STAMATIS, K., POLYMENAKOS, L., CROWLEY, J.L. “Agent Based Middleware Infrastructure for Autonomous Context-Aware Ubiquitous Computing Services”. Computer Communications 30, 2007, pp. 577–591.

[SYC98] SYCARA, K. “Multiagents Systems”. In: American Association for Artificial Intelligence, Summer 1998, pp. 79-92.

[VIT06a] VITERBO, J.F.; MALCHER, M.G.; ENDLER, M. “Supporting the development of context-aware agent-based systems for mobile networks”. In: MobiDE ’06 Chicago, Illinois USA, 2006, 8p.

[VIT06b] VITERBO, J. F., SACRAMENTO, V., ROCHA, R.C.A., ENDLER, M. “MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis”. In: Proceedings of the XXIV Brazilian Symposium on Computer Networks (SBRC 2006), Tool Session, 2006, 8p.

[WEI91] WEISER, M. “The Computer for the 21st Century”. Scientific American, vol. 265-3, September 1991, pp. 94-104.

[WEI99] WEISS, G. “Multiagent systems: a modern approach to distributed artificial intelligence”. Cambridge: The MIT Press, 1999, 619p.

[WOO95] WOOLDRIDGE, M., JENNINGS, N. “Intelligent Agents: Theory and Practice”. Cambridge University Press : Knowledge Engineering Review, vol. 10-2, 1995, pp. 115-152.

[ZAM03] ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. “Developing multiagent systems: the Gaia Methodology”. ACM Transactions on Software Engineering and Methodology, vol. 12-3, 2003. pp. 317-370