Richard Pereira e Silva

Um Estudo de Orientação a Contexto em Ambiente de Redes sem Fio

Florianópolis

2013

Richard Pereira e Silva

Um Estudo de Orientação a Contexto em Ambiente de Redes sem Fio

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação, da Universidade Federal de Santa Catarina, como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Dr. Mário Antônio Ribeiro Dantas

Coorientador: Dr. Elder Rizzon Santos

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis

2013

Monografia de graduação sob o título "Um Estudo Sobre Orientação a Contexto em Ambiente de Redes Sem Fio", defendida por Richard Pereira e Silva e aprovada em (dia) de (mês) de (ano), em Florianópolis, pela banca examinadora constituída pelos professores:

Prof. Dr. Mário Antônio Ribeiro Dantas
Universidade Federal de Santa Catarina
Orientador

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina
Coorientador

Prof. Dr. Ricardo Azambuja Silveira
Universidade Federal de Santa Catarina
Membro da Banca

Prof. Dr. Carlos Barros Montez
Universidade Federal de Santa Catarina
Membro da Banca

Resumo

Atualmente, o estudo de novas técnicas para construção de sistemas ubíquos vem sendo impulssionado pelo sucesso dos dispositivos móveis no mercado de eletrônicos. Um dos temas em evidência nos últimos anos é o desenvolvimento de software orientado a contexto. Contudo, ainda existem muitos problemas a serem resolvidos nessa área, como a forma ideal de representação e interpretação do contexto.

Um estudo das abordagens utilizadas na construção de software orientado a contexto torna-se interessante uma vez que não existe consenso sobre a melhor forma de representar e interpretar o contexto obtido. Neste estudo, são apresentadas abordagens para representação e interpretação do contexto, assim como um estudo de caso utilizando ontologias e agentes inteligentes para desempenhar tais funções.

Ao final do trabalho, é apresentado um caso de uso no qual pode ser visto o funcionamento do sistema desenvolvido e por fim são apresentadas as conclusões sobre as ferramentas e abordagens utilizadas durante o desenvolvimento do estudo de caso.

Abstract

Currently, the study of new ubiquitous systems building techniques is beeing stimulated by the success of the mobile devices in the electronic's market. One of the most noticeable themes in the last years is the development of context-aware software. However, there are many problems to be solved in this area, for example, the best approach of representation and interpretation of context.

A study of the approaches used in the context-aware software making becomes interesting once there is no consensus about the best way to represent and interpret the obtained context. In this study, approaches to represent and interpret the context are presented, as well as a study case using ontologies and intelligent agents to perform such functions.

By the end of the work, a use case is shown in which the behaviour of the developed system can be seen, and finally, conclusions about the tools and approaches used during the development of the study case are shown.

Agradecimentos

Aos meus pais, Alexandre e Lourdes, aos quais eu dedico este trabalho por todo seu carinho e dedicação desde os primeiros estágios de minha vida.

Agradeço aos meus orientadores, Mário e Elder, por sua paciência e respeito dedicados durante o período de realização deste projeto.

Agradeço também aos meus amigos e familiares que me ajudaram de qualquer forma, seja com palavras de incentivo ou apenas desejando meu bem.

Por último, mas não menos importante, agradeço a minha esposa Marina que esteve comigo em todos os momentos, me motivando a alcançar objetivos cada vez maiores.

Lista de Figuras

Lista de Tabelas

Glossário

BDI	Belief,Desire,Intention, 31
CONON	Context Ontology, 27
FIPA	Foundation for Intelligent Physical Agents, 30
JADE	Java Agent DEvelopment Framework, 44
OWL	Web Ontology Language, 27
PDA	Personal Digital Assistant, 34
RDF	Resource Description Framework, 20
RFID	Radio-Frequency IDentification, 34
SPARQL	SPARQL Protocol and RDF Query Language, 51
SPARUL	SPARQL Update Language, 51
UML	Unified Modeling Language, 20
W3C	World Wide Web Consortium, 27
XML	eXtensible Markup Language, 20

Sumário

1 Introdução

A principal motivação para a realização deste trabalho foi o desejo de trabalhar com redes de sensores sem fio e orientação a contexto utilizando conhecimentos obtidos durante o curso.

A orientação a contexto é um tema atualmente em evidência devido ao grande sucesso dos dispositivos móveis no mercado de eletrônicos. O desenvolvimento de sistemas móveis modernos está relacionado com o avanço nas técnicas de utilização de dados de contexto, como a localização física do dispositivo, preferências do usuário ou qualquer informação relacionada ao contexto em que o sistema esteja inserido.

Muitas pesquisas têm sido realizadas sobre o desenvolvimento de software orientado a contexto, mas existe pouco consenso sobre as melhores abordagens para representação e interpretação do contexto. Este trabalho começa com um estudo sobre os principais conceitos sobre orientação a contexto, os principais desafios de pesquisa nessa área, e termina com a apresentação de um estudo de caso sobre o tema.

O estudo de caso utiliza o domínio de redes de sensores sem fio para avaliar algumas abordagens de implementação de sistemas orientados a contexto, como por exemplo a utilização de ontologias para representação de dados e agentes inteligentes para realização de interpretação do contexto.

1.1 Objetivos

1.1.1 Objetivos Gerais

O objetivo deste trabalho é realizar um estudo sobre as possíveis aplicações de conceitos de orientação a contexto e inteligência artificial em ambientes de redes de sensores sem fio.

1.1.2 Objetivos Específicos

- Fazer o levantamento do estado da arte de abordagens de desenvolvimento de aplicações orientadas a contexto;
- Avaliar qualitativamente as tecnologias estudadas visando a escolha das mais adequadas para o estudo de caso;
- Elaborar um estudo de caso utilizando as tecnologias estudas em um ambiente real de redes de sensores sem fio.

1.1.3 Estrutura do Trabalho

Os primeiros capítulos deste trabalho serão dedicados ao levantamento de conceitos e tecnologias envolvidas no desenvolvimento do estudo de caso.

O capítulo 2 apresentará uma revisão teórica sobre orientação a contexto, abordando temas como modelos de representação de contexto, componentes de aplicações orientadas a contexto e desafios de pesquisa.

No capítulo 3 será apresentada uma revisão teórica sobre agentes inteligentes e sistemas multiagente além de trabalhos relacionados aos temas de orientação a contexto e agente inteligentes.

O capítulo 4 trará uma proposta de estudo de caso baseado nos temas dos capítulos 2 e 3 e buscará especificar os parâmetros de implementação do estudo de caso.

O capítulo 5 descreverá a implementação e as ferramentas utilizadas durante o desenvolvimento do estudo de caso. Já no capítulo 6 serão discutidos os resultados e conclusões, assim como os trabalhos futuros.

2 *Orientação a Contexto*

O desenvolvimento de aplicações orientadas a contexto surgiu como uma área da computação ubíqua. Portanto, este capítulo começará com uma breve introdução à computação ubíqua que será seguida por um estudo sobre a orientação a contexto.

2.1 Computação Ubíqua

A computação ubíqua é também conhecida como computação pervasiva ou, mais recentemente, everyware [Greenfield 2006]. Cada um desses termos foi criado dando ênfase em uma característica diferente da computação ubíqua.

O conceito de computação ubíqua foi introduzido por [Weiser 1991] como sendo um novo modelo de computação pós-desktop, no qual as pessoas utilizariam computadores inconscientemente para realizar tarefas do dia-a-dia. Estes computadores seriam usados inconscientemente devido ao seu tamanho reduzido, eles estariam presentes em muitos objetos do nosso ambiente e seriam tão simples de usar que se tornariam invisíveis na maior parte do tempo.

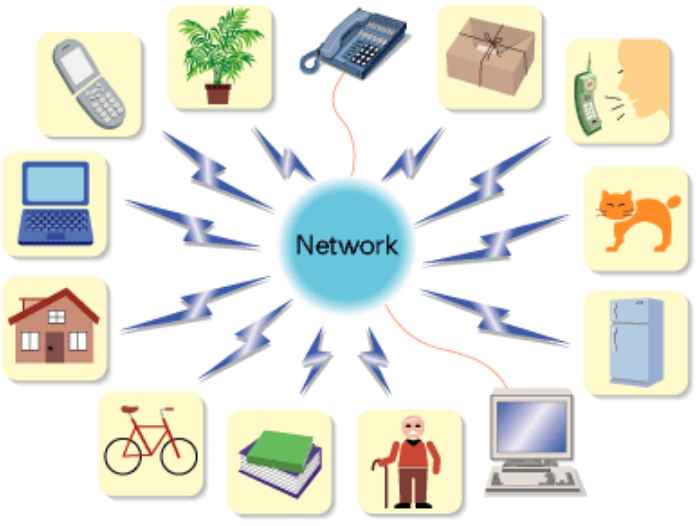

Figura 2.1: Modelo de computação ubíqua

Atualmente, a computação ubíqua vem se tornando uma realidade devido a redução dos dispositivos computacionais, porém ainda existem muitas pesquisas na área de desenvolvimento de software para sistemas ubíquos.

2.2 Definição de Contexto

Um dos primeiros trabalhos sobre aplicações computacionais orientadas a contexto foi escrito por [Schilit, Adams e Want 1994]. Este trabalho definiu contexto como sendo informações sobre localização, identidades de pessoas e objetos próximos entre si, assim como as mudanças nesses objetos.

Em [Dey 2001], foi proposta uma nova definição para contexto, esta definição mostrou-se mais geral e portanto tornou mais fácil para o desenvolvedor de aplicações enumerar os dados de contexto para uma aplicação específica. Segundo [Dey 2001], a definição antes proposta por [Schilit, Adams e Want 1994] causava dúvida em desenvolvedores, pois eles não sabiam exatamente se os seus dados se encaixavam na definição de informações de contexto.

A definição de [Dey 2001] diz que: "Contexto é qualquer informação que possa ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre usuário e uma aplicação, incluindo o usuário e a aplicação."

A definição de contexto proposta por [Dey 2001] será utilizada para a realização deste trabalho.

2.3 Tipos de Dados de Contexto

Uma das classificações mais interessantes encontradas na literatura para os diversos tipos de dados de contexto é a de [Chen e Kotz 2000]. Eles afirmam que todos os dados de contexto estão contidos em um espaço de quatro dimensões definido pelos aspectos computacionais, físicos, temporais e de usuário.

Se encaixariam em contexto computacional todos os dados relacionados ao sistema computacional em que a entidade está situada. Nesta classificação estariam, por exemplo, dados de posição de recursos como impressoras, informações de conexão e outros.

Informações de contexto físico são dados como luminosidade do ambiente, níveis de ruído, temperatura e outros dados que descrevam a situação física do usuário ou do dispositivo em

questão. O contexto de usuário diz respeito a localização de pessoas próximas, perfil de usuário e condição social do usuário da aplicação. Dados temporais como estação atual, ano, semana e mês se encontram na definição de contexto temporal. Alguns tipos de dados de contexto podem se encaixar em mais de uma classificação.

2.4 Aplicações Orientadas a Contexto

O conceito de aplicações orientadas a contexto foi proposto por [Schilit, Adams e Want 1994] como sendo qualquer software que se adapta de acordo com a sua localização de uso, coleções de pessoas e objetos próximos, assim como mudanças desses objetos ao longo do tempo. Contudo, esta definição mostrou-se limitada à medida que a computação móvel evoluiu e a demanda por aplicações orientadas a contexto aumentou.

Segundo [Dey 2001]: "Um sistema é orientado a contexto se ele usa dados de contexto para prover informação relevante e/ou serviços para o usuário, uma vez que relevância depende da tarefa do usuário." Esta definição foi criada por [Dey 2001] com o intuito de englobar todas as aplicações orientadas a contexto que existiam e que poderiam ser criadas no futuro, mas que não se encaixavam na antiga definição de [Schilit, Adams e Want 1994].

A definição proposta por [Dey 2001] será utilizada durante a realização deste trabalho devido a sua abrangência e a sua grande aceitação na maioria dos artigos recentes sobre orientação a contexto. (citar referências)

2.5 Tipos de Aplicações Orientadas a Contexto

Existem muitos requisitos e finalidades diferentes entre as aplicações orientadas a contexto. Essas diferenças fizeram com que [Schilit, Adams e Want 1994] produzissem uma classificação clássica para os sistemas orientados a contexto. A classificação de [Schilit, Adams e Want 1994] divide as aplicações em quatro classes com características diferentes: seleção do mais próximo, reconfiguração automática baseada no contexto, ações disparadas por contexto e comandos dependentes de contexto.

A seleção do mais próximo é uma técnica de interface que faz com que os objetos fisicamente mais próximos do dispositivo sejam enfatizados ou tornem-se mais fáceis de serem escolhidos pelo usuário.

A reconfiguração baseada no contexto consiste em adicionar, remover ou alterar as conexões

entre componentes de software devido a informações contextuais.

Dentro da classe de comandos dependentes de contexto estão as aplicações que executam ações customizadas pelo contexto em que estão inseridas.

Aplicações que possuem ações disparadas por contexto são simplesmente reativas. Elas reagem aos dados de contexto recebidos tomando uma ação específica. Essa reatividade pode ser implementada com estruturas de seleção Se-Então.

2.5.1 Orientação a Contexto Ativa e Passiva

Em [Chen e Kotz 2000] foi proposta uma classificação diferente para as aplicações orientadas a contexto. Eles a definem como orientação a contexto ativa e passiva. A ativa compreende as aplicações que automaticamente se adaptam ao contexto mudando o comportamento da aplicação. De outra forma, a passiva apenas apresentaria o contexto atual ou anterior ao usuário interessado. De acordo com [Chen e Kotz 2000], a orientação a contexto ativa é mais interessante pois produz mais aplicações em dispositivos móveis e demanda uma maior infra-estrutura de suporte.

2.6 Componentes de aplicações orientadas a contexto

Em [Malik, Mahmed e Javed 2007] é fornecida uma abstração para os principais componentes de um sistema orientado a contexto. Esses componentes realizam tarefas específicas e serão descritos em mais detalhes.

Na figura 2.2, os retângulos representam as abstrações de cada componente do sistema e as setas representam a comunicação ou o fluxo de dados entre os componentes.

O componente de aquisição de contexto é o responsável pela percepção ou obtenção dos dados de contexto e o posterior agrupamento desses dados. Depois de agrupadas, as informações são enviadas para o próximo componente, responsável por estruturar os dados de entrada utilizando um modelo de representação que facilite o acesso, armazenamento e a interpretação do contexto.

Quando as informações já se encontram estruturadas, elas podem ser inseridas em um dispositivo de armazenamento, para que possam ser interpretadas futuramente. A interpretação do contexto ocorre necessariamente antes da adaptação do sistema, pois ela agrega significado às informações obtidas na fase de aquisição. Na fase final, o sistema adapta o seu funcionamento de acordo com as informações obtidas da interpretação do contexto.

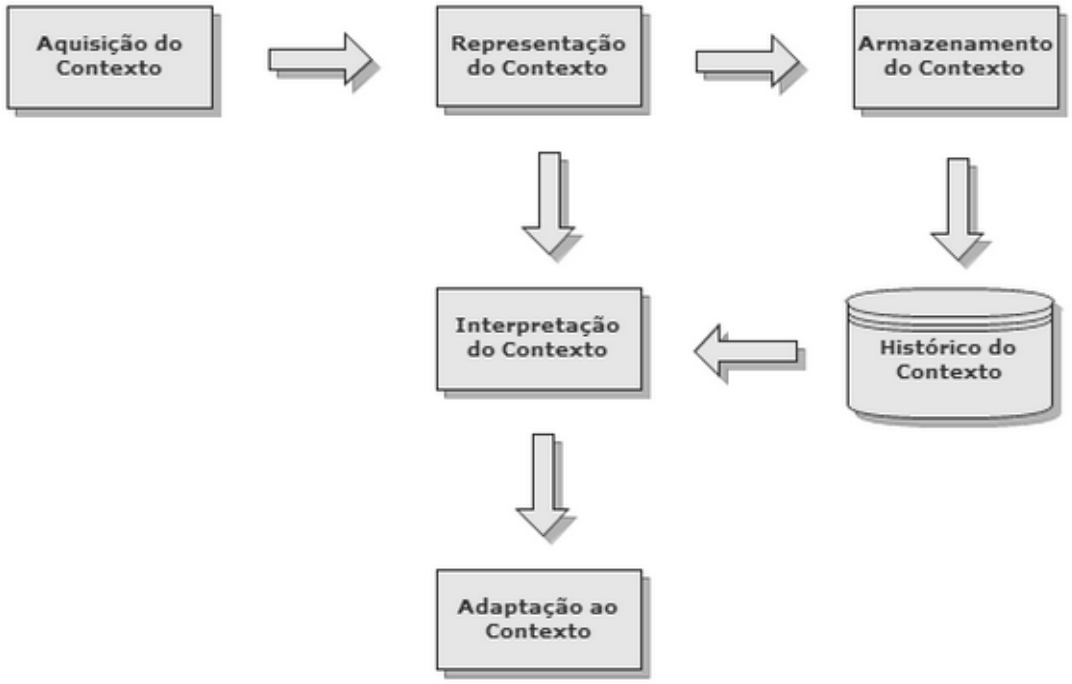

Figura 2.2: Componentes de um sistema orientado a contexto [Malik, Mahmed e Javed 2007]

2.7 Desafios gerais na pesquisa de orientação a contexto

A orientação a contexto ainda é uma área relativamente nova na computação, por isso ainda existem muitas questões a serem respondidas e muitas pesquisas vêm tentando solver os problemas encontrados até então.

Alguns dos desafios na pesquisa de orientação a contexto foram descritos em [Winograd 2001] e resumidos em uma tabela por [Malik, Mahmed e Javed 2007]:

Desafio	Descrição
Definição de contexto	Contexto é um termo que pode ser interpretado de várias formas. Os desenvolvedores de aplicações e frameworks devem definir contexto e identificar parâmetros relevantes ao seu escopo.
Modelos de Orientação a Contexto	As arquiteturas de orientação a contexto ainda estão na sua infância. A maioria dos modelos de arquitetura são específicos para cada aplicação. Padrões e ferramentas ainda estão para ser desenvolvidas.

Sensoriamento de Dados de Contexto	Dispositivos de sensoriamento de contexto ainda estão sendo desenvolvidos. Weareable computing é uma subárea da computação orientada a contexto que realiza projetos e desenvolvimento de sensores de contexto
Previsão de Dados de Contexto	A indisponibilidade de sensores de contexto requer a previsão de dados de contexto baseada no histórico. Técnicas probabilísticas como modelos Bayesianos podem ser utilizados.
Representação e Armazenamento de Dados de Contexto	O esquema de representação de contexto deve facilitar a interpretação do contexto e o processo de compartilhamento além de seguir uma estrutura padronizada.
Interpretar o Contexto e Adaptar o Sistema	A interpretação do contexto e adaptação do comportamento do serviço é um dos principais desafios da orientação a contexto. O processo de interpretação é orientado a adaptação.
Avaliação de Sistemas Orientados a Contexto	Os critérios de avaliação devem ser definidos para verificação de sistemas orientados a contexto. Métricas para controle de qualidade e satisfação do usuário final devem ser produzidas.
Controle de Privacidade	Os dados das entidades participantes são privados e devem ser protegidos de exposição a entidades maliciosas enquanto estiverem transitando ou em dispositivos de armazenamento.

Tabela 2.1: Desafios na pesquisa de orientação a contexto

2.8 Formas de representação do contexto

Existem muitas formas de se representar computacionalmente o contexto. Algumas destas formas se destacam nos dias de hoje por possuirem características especificas. Ainda existem pesquisas sobre como se representar o contexto ou como melhorar os métodos existentes,

portanto nenhum método é unanimidade e o desenvolvedor deve procurar o modelo que seja o mais adequado para resolver o seu problema.

A seguir, uma breve descrição de alguns dos modelos de representação computacional de informações de contexto.

2.8.1 Modelos Chave-Valor

O Modelo Chave-Valor utiliza mapeamentos muito simples para organizar as informações de contexto. Em [Schilit, Adams e Want 1994] foi dado um exemplo de como se utilizar esse modelo utilizando variáveis de ambiente do sistemas operacional.

2.8.2 Modelos de Marcação

Modelos de marcação são estruturas de dados hierárquicas que consistem de texto de marcações com atributos e conteúdo. A aplicação típica deste tipo de representação em aplicações orientadas a contexto é a descrição de perfis. Exemplos de tecnologias que podem ser usadas na construção desses modelos são XML e RDF.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
    xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
    xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
    xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
    <SessionProfile rdf:ID="Session">
        <cscp:defaults rdf:resource=
            "http://localContext/CSCPProfile/previous#Session"/>
        <device><dev:DeviceProfile>
            <dev:hardware><dev:Hardware>
                <dev:memory>9216</dev:memory>
                </dev:Hardware></dev:hardware></dev:DeviceProfile>
        </device>
    </SessionPofile>
</rdf:RDF>
```

Figura 2.3: Exemplo do uso de RDF na representação do perfil de um dispositivo

2.8.3 Modelos Gráficos

Modelos gráficos como a UML podem ser utilizados na representação de contexto devido a sua generalidade. Um exemplo de como utilizar a UML para representar informações contexto foi mostrado por [Bauer 2003].

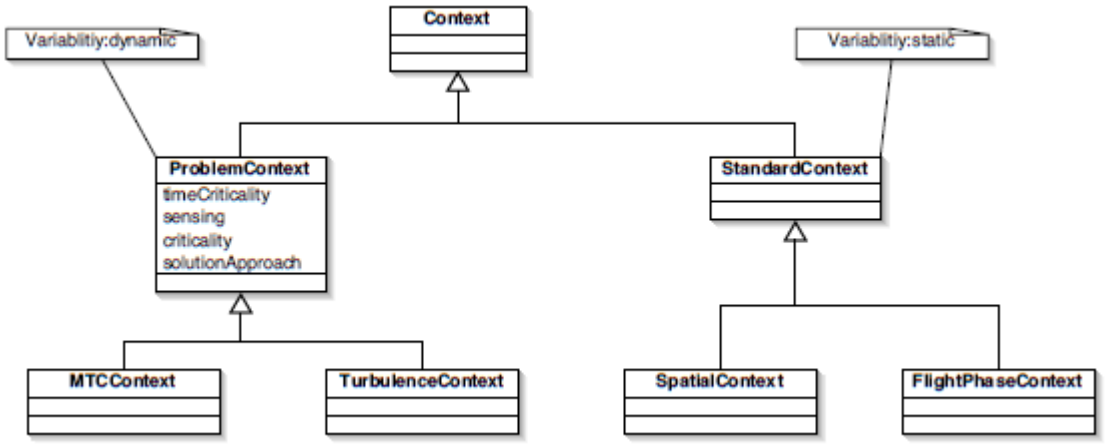

Figura 2.4: Modelagem em UML para um sistema de controle de tráfego áereo. [Bauer 2003]

2.8.4 Modelos Orientados a Objeto

Os modelos orientados a objeto buscam tirar proveito das principais características da orientação a objeto: encapsulamento e reusabilidade. A dinâmica do contexto pode ser um problema em muitos sistemas, contudo modelos orientados a objeto encapsulam os elementos de contexto de forma que os outros componentes possam acessar esses dados através de uma interface padrão.

2.8.5 Modelos Baseados em Lógica

Em um sistema utilizando modelo de contexto baseado em lógica o contexto é definido por fatos, expressões e regras. Nesses modelos, a informação de contexto é adicionada, atualizada ou excluída como sendo fatos ou inferida através das regras presentes no modelo.

2.8.6 Modelos Baseados em Ontologias

Ontologias são utilizadas para descrever conceitos e suas relações de maneira que o computador possa interpretar. Portanto, as ontologias são ferramentas muito úteis, pois podemos descrever informações relevantes sobre o contexto para uma aplicação específica.

A figura 2.5 mostra uma ontologia descrevendo informações que poderiam ser usadas como dados de um sistema de aprendizado orientado a contexto.

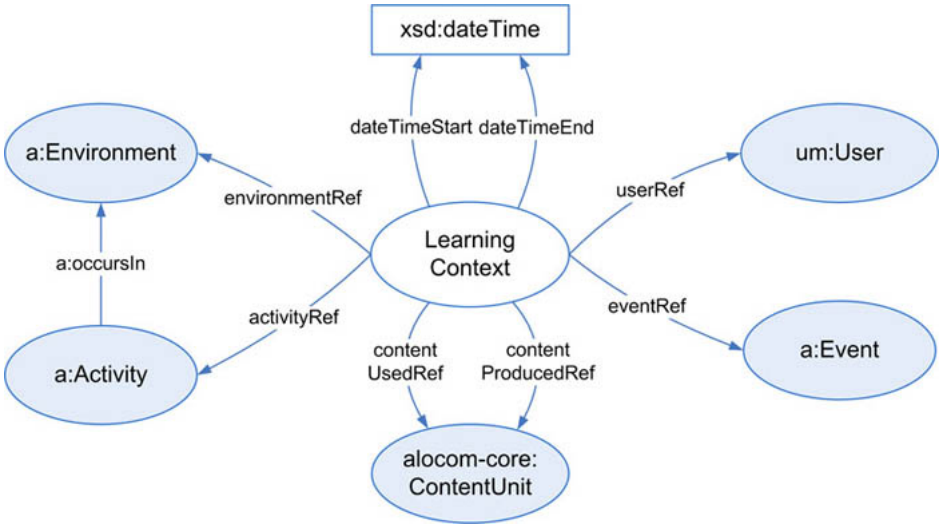

Figura 2.5: Exemplo de ontologia

2.8.7 Estudo Comparativo dos Modelos

Em [Strang e Linnhoff-Popien 2004] é apresentado um estudo comparativo de técnicas de representação de contexto. Os critérios avaliados foram: composição distribuída (dc), validação parcial (pv), riqueza e qualidade da informação (qua), incompleteza e ambiguidade (inc), nível de formalização (for) e aplicabilidade nos ambientes existentes (app).

- **Composição distribuída (dc):** critério baseado no desempenho e capacidade de administrar o modelo e os seus dados em ambientes distribuídos.
- **Validação Parcial (pv):** é altamente desejável que os dados de contexto sejam validados com um modelo de contexto em uso. Isso é importante devido à complexidade das relações entre os dados de contexto, que podem fazer a modelagem ser propensa a erros.
- **Riqueza e Qualidade da Informação (qua):** avalia a qualidade dos dados fornecidos pela representação escolhida, assim como a existência de indicações de riqueza e qualidade desses dados.
- **Incompleteza e ambiguidade (inc):** o conjunto de informações de contexto disponível a qualquer momento caracterizando as entidades no ambiente de computação ubíqua é usualmente incompleta e/ou ambígua, em particular se a informação é recebida de redes de sensores. Isso deve ser coberto pelo modelo, seja por interpolação dos dados recebidos ou por qualquer outro método.
- **Nível de Formalização (for):** fatos de contexto e relações devem ser descritos de maneira precisa. Por exemplo, para realizar a terafa “Imprimir documento em impressora perto de

mim", é necessáriio ter uma definição precisa dos termos usados na tarefa, o que "perto" significa para "mim". É importante que todas as entidades do sistema compartilhem a mesma interpretação dos dados trocados .

- **Aplicabilidade nos Ambientes Existentes (app):** avalia a facilidade para utilização deste modelo com as tecnologias existentes, por exemplo, frameworks de Web Service.

Abordagem - Requisitos	dc	pv	qua	inc	for	app
Modelos Chave-Valor	-	-	- -	- -	- -	+
Modelos Marcação	+	+ +	-	-	+	+ +
Modelos Gráficos	- -	-	+	-	+	+
Modelos Orientados a Objeto	+ +	+	+	+	+	+
Modelos Baseados em Lógica	+ +	-	-	-	+ +	-
Modelos Baseados em Ontologias	+ +	+ +	+	+	+ +	+

Tabela 2.2: Avaliação dos Modelos de Representação de Dados de Contexto

Na tabela 3.1 foram atribuídas quatro notas, sendo - - a mais baixa e + + a mais alta.

2.9 Ontologias

Como pode ser visto na tabela 3.1, as ontologias são vistas como uma boa forma de se representar o contexto, levando em consideração os requisitos utilizados no estudo de [Strang e Linnhoff-Popien 2004]. A palavra ontologia (do grego "ontos" e "logoi", "conhecimento do ser") tem sua origem na filosofia, e diz respeito a uma área da filosofia que trata de questões metafísicas.

Em ciência da computação, uma ontologia tem um significado diferente: é um modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre eles. Uma ontologia pode ser utilizada para realizar inferência sobre os objetos do domínio. Ontologias são utilizadas em inteligência artificial, web semântica, engenharia de software e arquitetura da informação, como uma forma de representação de conhecimento sobre o mundo ou alguma parte dele.

Ontologias são usadas na representação de contexto e algumas das razões para isso são enumeradas por [Ay 2007] e [Wang et al. 2004].

- Uma ontologia serve como um modelo de conceitos (entidades) e relacionamentos, o que torna-a um meio de se representar o contexto.

- As ontologias possuem uma definição formal, o que é uma pré-condição para fazer a interpretação do contexto por um computador.
- É possível aplicar regras sobre ontologias para implementar a interpretação do contexto.
- Reusando ontologias bem definidas de diferentes domínios podemos compor novas ontologias sem ter que criá-las do zero.

2.10 Considerações

Neste capítulo, foram mostrados conceitos e abordagens de implementação de sistemas orientados a contexto. Das abordagens de representação de contexto mostradas no capítulo, o modelo baseado em ontologias se mostrou mais adequado para aplicações com requisitos complexos.

Outras abordagens como modelos gráficos, modelos de marcação e chave-valor podem ser utilizados em sistemas com baixa complexidade, pois estes impõem poucas restrições como não-ambiguidade e qualidade de dados.

No próximo capítulo será feito um levantamento sobre os conceitos e abordagens de implementação de agentes inteligentes, pois estes são parte integrante da etapa de interpretação do contexto. Como pode ser visto na figura 2.2, a etapa de interpretação do contexto é parte de qualquer sistema orientado a contexto e esta etapa permanece como um desafio na pesquisa de orientação a contexto, de acordo com a tabela 2.1.

3 *Interpretação do Contexto e Agentes Inteligentes*

No que diz respeito a orientação a contexto, alguns trabalhos como [Wang et al. 2004] e [Padovitz, Loke e Zaslavsky 2008] têm realizado estudos sobre possíveis métodos de utilização de agentes inteligentes na interpretação de contexto. Neste capítulo serão abordados os conceitos básicos de agentes inteligentes e algumas abordagens para utilização destes na orientação a contexto.

3.1 Agentes Inteligentes

Segundo [Russel e Norvig 2004]: ”Um agente é tudo aquilo que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores.”

De acordo com a definição de [Russel e Norvig 2004], pode ser visto como agente, por exemplo, um robô que possua câmeras como sensores e um motor como atuador. Da mesma forma, um software que receba dados de entrada através de uma interface qualquer e produza saídas (exibindo algo na tela ou gravando um arquivo), pode ser visto como um agente.

Algumas características importantes de agentes são descritas por [Jennings e Wooldridge 1996]:

- Autonomia: agentes devem ter a capacidade de realizar a maioria de suas tarefas sem a intervenção direta do usuário ou de outros agentes. Para ser considerado autônomo, um agente deve ter um nível de controle sobre suas ações.
- Habilidade Social: agentes podem ser capazes de interagir com outros agentes de software e humanos para completar suas tarefas e para ajudar outros agentes a completarem suas tarefas.
- Reatividade: agentes devem perceber o seu ambiente, seja ele físico ou não, e responder a

tempo as mudanças que nele ocorrem.

- Pró-atividade: agentes não devem simplesmente responder ao seu ambiente, mas também mostrar iniciativa própria para alcançar o seus objetivos.

3.2 Sistemas Multiagente

Sistemas que possuem mais de um agente interagindo uns com os outros de alguma forma para alcançar um objetivo são chamados de sistemas multiagente.

Tais sistemas são interessantes algumas vezes, pois com eles podemos resolver problemas que seriam difíceis ou impossíveis com apenas um agente.

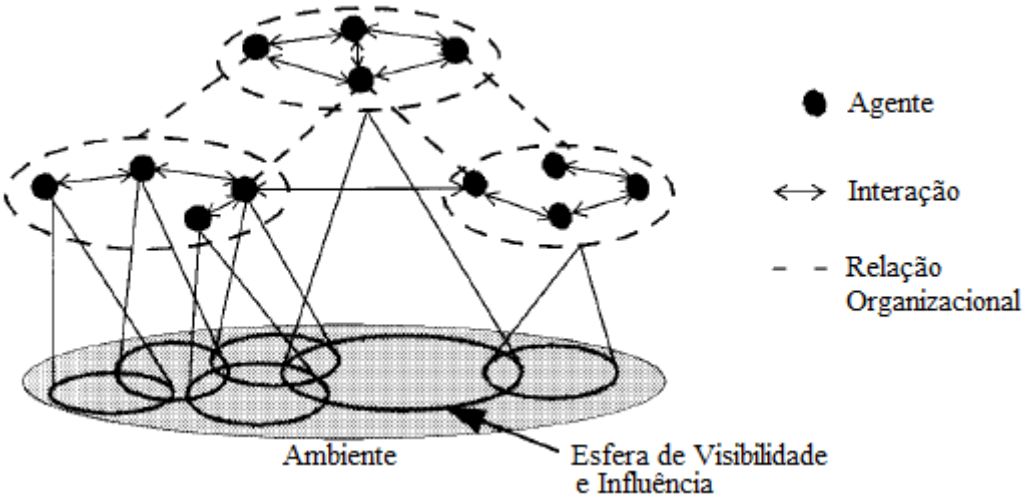

Figura 3.1: Estrutura típica de um sistema multiagente em [Jennings 2000]

A figura 3.1 deixa isso claro, uma vez que a chave para a resolução de um problema pode estar fora da esfera de visibilidade de um dos agentes. Porém, com a cooperação de um outro agente, este problema pode ser resolvido.

Mecanismos de comunicação entre os agentes são, portanto, fundamentais para o desenvolvimento de sistemas multiagente.

Segundo [Wooldridge 2009]: ”Os sistemas multiagente, antes vistos como raros ou não comuns, estão se tornando de fato a regra na computação.” A medida que os sistemas tornam-se mais complexos, surge a necessidade de dividi-los em subsistemas com o objetivo de realizar uma tarefa específica.

3.3 Ontologias e Agentes de Contexto

Além de representar o contexto em si, ontologias servem como um vocabulário comum entre agentes, promovendo assim a comunicação entre eles e a mesma interpretação dos dados.

Existem muitas ontologias para domínios específicos que podem ser usadas na construção de um sistema orientado a contexto. Ontologias como a CONON (Context Ontology), foram criadas para modelagem desses sistemas. Essas ontologias podem poupar muito trabalho e ainda podem ser extendidas para se adequarem a cada sistema.

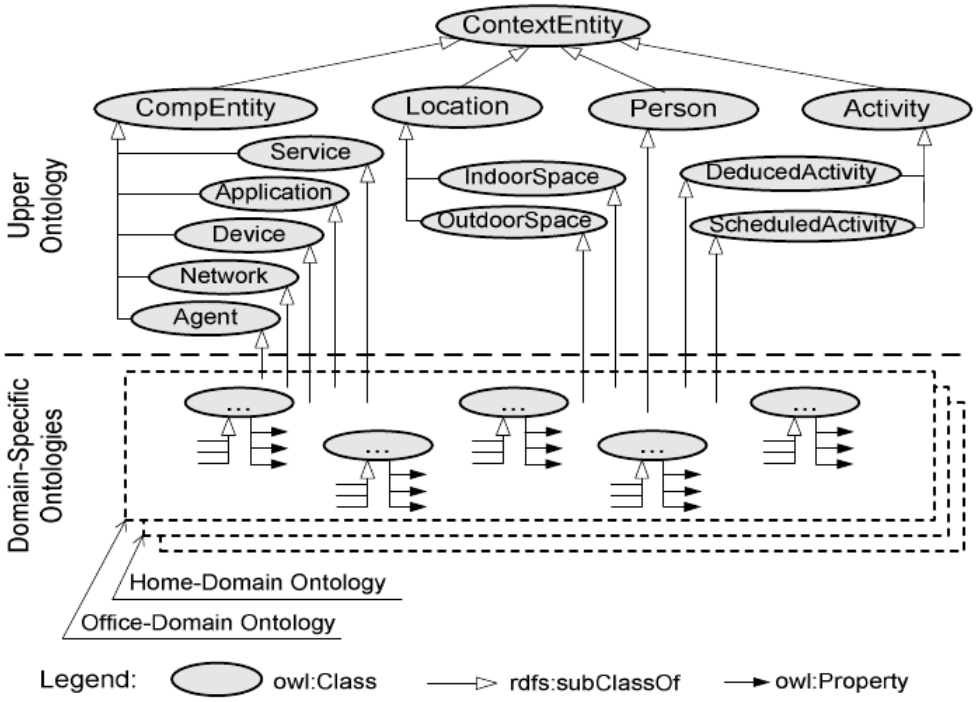

Figura 3.2: Exemplo de uma ontologia de contexto em [Wang et al. 2004]

A ontologia CONON foi codificada usando OWL (Web Ontology Language). A OWL é hoje um padrão recomendado pela [W3C] e foi feita para ser utilizada por aplicações que precisem processar o conteúdo da informação, ao contrário de apenas apresentá-lo ao usuário. A OWL disponibiliza uma forma comum para o processamento de conteúdo semântico na internet.

Na figura 3.2, podemos ver como funciona a construção de ontologias para domínios específicos. Na parte de cima da figura estão as classes pré-definidas da ontologia CONON, já na parte de baixo as possíveis ontologias que podem ser criadas a partir da existente.

3.4 Trabalhos Relacionados

Essa seção trará descrições de trabalhos relacionados que tratam sobre desenvolvimento de agentes inteligentes orientados a contexto utilizando diferentes abordagens de representação de contexto.

3.4.1 CoolAgent RS

O CoolAgent RS (Recommendation Service) é um sistema multiagente que foi desenvolvido por [Chen et al. 2003] para serviços de recomendação. O CoolAgent RS possui dois serviços distintos de recomendação orientada a contexto, um para recomendação de documentos em uma reunião, e outro para recomendação de comida em uma cafeteria.

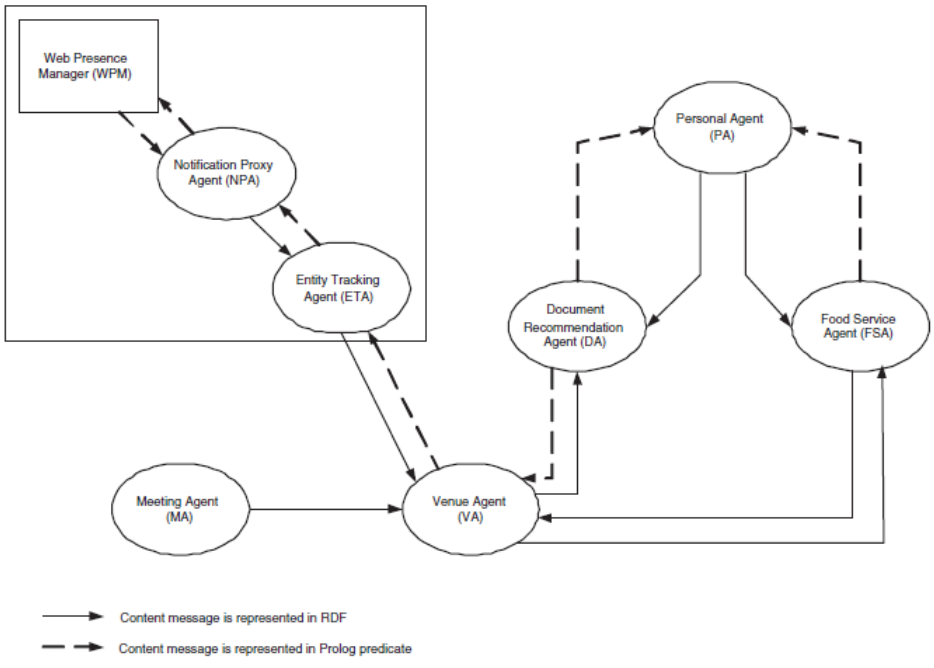

Figura 3.3: Arquitetura do CoolAgent RS

Como pode ser visto na figura 3.3, os vários agentes do sistema se comunicam através de ontologias codificadas em RDF e predicados da linguagem de programação em lógica Prolog. Cada um dos agentes possui um objetivo diferente descrito abaixo:

- Notification Agent Proxy (NAP): agente responsável por determinar a presença de objetos físicos em uma determinada localização através da interação com o CoolTown Web Presence Manager. O NAP provê um serviço de mapeamento de ontologias, traduzindo informações de contexto da ontologia CoolTown WPM para a ontologia CoolAgent RS e repassando para os agentes interessados.

- Entity Tracking Agent (ETA): esse agente é responsável por rastrear a presença de uma entidade numa localização específica. ETA se inscreve em NAP para ser notificado sempre que um objeto físico estiver presente em uma localização específica.

- Venue Agent (VA): agente responsável por obter informação contextual de uma localização específica, incluindo pessoas presentes, informações antecipadas de reunião e relações entre entidades do sistema. Essas informações são obtidas recebendo notificações de presença do ETA e comunicando-se com o Meeting Agent.

- Personal Agent (PA): esse agente é responsável por disponibilizar perfis pessoais e profisionais de usuários. O PA compartilha uma ontologia comum com os outros agentes.

- Meeting Agent (MA): cria uma página web codificada em RDF para descrever cada reunião. O VA extrai detalhes para as próximas reuniões examinando as páginas RDF criadas pelo MA.

- Document Recommendation Agent (DA): esse agente é responsável por fazer recomendações de documentos para os participantes da reunião baseado no contexto dos participantes.

- Food Service Agent (FA): responsável por fazer recomendações de pratos para os clientes da cafeteria baseado no contexto dos clientes.

O serviço de recomendação de documentos em uma reunião baseia-se em um conjunto de dados de contexto enumerado abaixo:

1. **Presença de um participante**: uma pessoa é um participante da reunião se ela está presente na sala de reunião no momento em que uma reunião está agendada para acontecer.

2. **O perfil do participante**: perfil pessoal e profissional do participante da reunião, qual o seu cargo, quais a área de pesquisa do participante, etc.

3. **O contexto da reunião**: informações sobre o tema da reunião, projetos e pesquisas relacionados ao tema da reunião, o assunto da reunião e os seus participantes.

4. **Informação organizacional**: dados sobre relações entre os participantes dentro da empresa. Por exemplo, se o empregado A é supervisor do empregado B, ou se B está trabalhando no mesmo projeto de A.

Já o serviço de recomendação de comidas baseia-se nas seguintes informações de contexto:

1. **Presença de um cliente**: informação sobre a presença ou ausência de um cliente na cafeteria.
2. **O perfil do cliente**: perfil pessoal do cliente a ser atendido na cafeteria.
3. **Os pratos do dia**: informações sobre os pratos sendo servidos no dia em questão, assim como detalhes sobre ingredientes e modo de preparo.

Para implementar o raciocínio dos agentes sobre o contexto [Chen et al. 2003] criaram regras em Prolog que expressam o conteúdo das ontologias em lógica de primeira ordem. As propriedades rdfs:subClassOf e rdf:type , por exemplo, foram usadas no Prolog utilizando as regras abaixo:

- rdfs:subClassOf
triple(A, subClassOf, B) $\Rightarrow$ subClassOf(A,B).
triple(A, subClassOf, B), subClassOf(B,C) $\Rightarrow$ subClassOf(A,C).

- rdf:type
triple(I, type, C) $\Rightarrow$ instanceOf(I,C).
subClassOf(B,C), instanceOf(I,B) $\Rightarrow$ instanceOf(I,C).

A comunicação entre os agentes foi realizada utilizando o protocolo [FIPA] Subscribe Interaction Protocol. Neste protocolo os agentes inscrevem-se uns com os outros para receber mensagens sobre seus temas de interesse.

3.4.2 Agentes BDI

Outra abordagem para produção de agentes orientados a contexto é proposta por [Vieira et al. 2010]. Trata-se de uma linguagem de programação para descrição de agentes inteligentes baseados na arquitetura BDI (Belief,Desire,Intention). A arquitetura BDI tenta modelar os conceitos de crença, desejo e intencionalidade em agentes.

Em [Vieira et al. 2010] foi proposta uma modificação na linguagem AgentSpeak(L) para implementar raciocínio e comunicação entre os agentes utilizando ontologias. A figura 3.4 mostra uma visão da arquitetura BDI implementada na linguagem AgentSpeak(L).

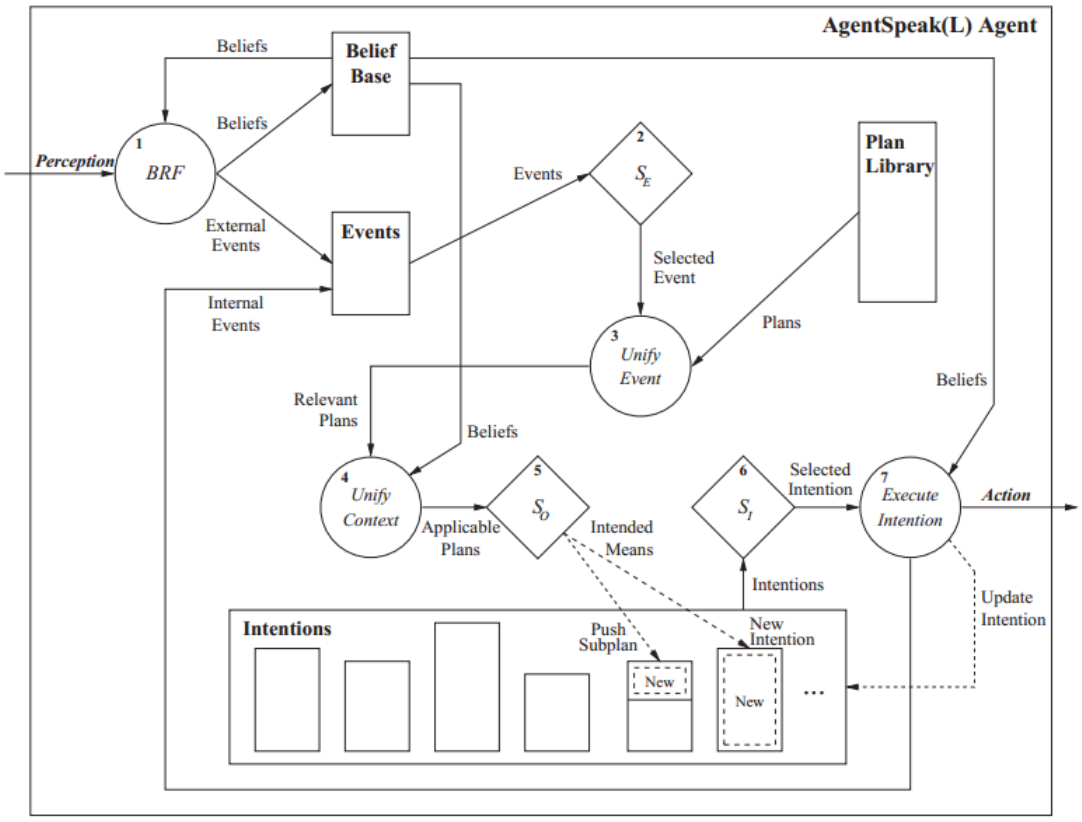

Figura 3.4: Arquitetura BDI na linguagem AgentSpeak(L)

Como pode ser visto na figura 3.4, um agente construído com a linguagem AgentSpeak(L) possui uma base de crenças (Belief Base) que pode ser atualizada de acordo com a percepção do agente. O agente possui também planos que ele tentará executar para atingir os seus desejos ou objetivos. As intenções também são levadas em conta na execução das ações do agente e podem ser modificadas assim como a base de crenças.

A modificação realizada por [Vieira et al. 2010] na linguagem AgentSpeak(L) pode ser vista examinando a sintaxe da linguagem antes e depois da modificação:

Antes:

$$
\begin{array}{lll}
ag & ::= bs \quad ps & \\
bs & ::= at_1 \ldots at_n & (n \geq 0) \\
at & ::= P(t_1, \ldots t_n) & (n \geq 0) \\
ps & ::= p_1 \ldots p_n & (n \geq 1) \\
p & ::= te : ct \leftarrow h & \\
te & ::= +at \mid -at \mid +g \mid -g & \\
ct & ::= at \mid \neg at \mid ct \wedge ct \mid \mathsf{T} & \\
h & ::= a \mid g \mid u \mid h; h & \\
g & ::= !at \mid ?at & \\
u & ::= +at \mid -at &
\end{array}
$$

Depois:

$$
\begin{array}{lll}
ag & ::= Ont \quad bs \quad ps & \\
Ont & ::= context\_ontology(url_1, \ldots url_n) & \\
bs & ::= at_1 \ldots at_n & (n \geq 0) \\
at & ::= C(t) \mid R(t_1, t_2) & \\
 & \mid C(t)[s_1, \ldots, s_n; url] & (n \geq 0) \\
 & \mid R(t_1, t_2)[s_1, \ldots, s_n; url] & (n \geq 0) \\
s & ::= \texttt{percept} \mid \texttt{self} \mid id & \\
ps & ::= p_1 \ldots p_n & (n \geq 1) \\
p & ::= te : ct \leftarrow h & \\
te & ::= +at \mid -at \mid +g \mid -g & \\
ct & ::= at \mid \neg at \mid ct \wedge ct \mid \mathsf{T} & \\
h & ::= a \mid g \mid u \mid h; h & \\
g & ::= !at \mid ?at & \\
u & ::= +at \mid -at &
\end{array}
$$

Figura 3.5: Sintaxe da linguagem AgentSpeak(L)

Podemos ver na figura 3.5 que foram adicionados predicados para a inclusão de ontologias de contexto nos agentes. Elas são incluídas especificando-se a url de cada ontologia.

Depois de incluídas no sistema, o grupo de [Vieira et al. 2010] trabalhou assumindo que o interpretador de AgentSpeak(L) tivesse suporte a troca de mensagens assíncronas entre os agentes. As mensagens poderiam ser trocadas então, executando a ação ".send" no corpo do plano de um dos agentes. O formato da mensagem é <mid,id,Ilf,at,url> no qual *mid* é um identificador de uma mensagem, *id* é o identificador do agente para o qual a mensagem está sendo encaminhada, *Ilf* é o tipo da mensagem, *at* é o conteúdo e *url* é a localização da ontologia.

Muitos tipos de mensagens foram criados, porém alguns deles são interessantes pois levam a raciocínio sobre as ontologias. Por exemplo, supondo que um agente A esteja mandando uma mensagem para B. As mensagens abaixo são exemplos de tipos de mensagens que poderiam ser trocadas:

- tell: o agente A informa B que a sentença da mensagem é verdadeira em A, ou seja, a sentença está na base de crenças de A.
- ask-if: o agente A quer saber se a sentença contida na mensagem é verdade para B, ou seja, se está na base de crenças de B.
- ask-all: o agente A quer todas as respostas de B para uma pergunta.
- ask-how: o agente A quer saber todos os planos de B para um evento.

3.4.3 Outros trabalhos

Um trabalho bastante interessante foi desenvolvido em [Bardram 2004] sobre a utilização de orientação a contexto em um ambiente hospitalar. Foram utilizados sensores de RFID [1] para identificar pacientes e medicações no hospital, possibilitando assim fazer verificações sobre as aplicações de medicamentos nos pacientes corretos dentro do hospital, assim como visualizar perfis de pacientes a partir de terminais instalados nas camas.

Entre os protótipos desenvolvidos durante a pesquisa de [Bardram 2004] estão uma caixa de remédios que detecta quando o paciente está próximo baseado em RFID e o sistema com reconhecimento de impressões digitais capaz de visualizar dados sobre os pacientes e suas respectivas medicações. Ambos podem ser vistos na figura 3.6.

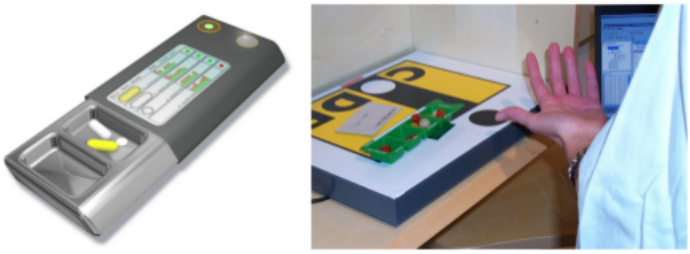

Figura 3.6: Protótipos desenvolvidos pela pesquisa de [Bardram 2004]

Outro trabalho relacionado é [Tesoriero et al. 2008], ele implementa um sistema que usa informações sobre localização do usuário para trazer informações em um PDA sobre uma exposição de arte. Novamente a tecnologia de RFID foi utilizada juntamente com outros sensores para criar um sistema inteligente capaz de trazer informações precisas ao usuário sobre o seu contexto. A imagem 3.7 mostra os PDAs que foram usados no projeto de [Tesoriero et al. 2008].

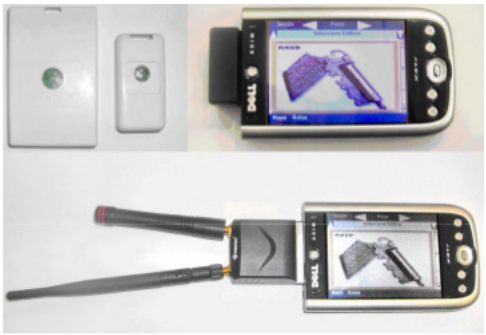

Figura 3.7: Protótipos desenvolvidos pela pesquisa de [Tesoriero et al. 2008]

[1] Identificação por radiofrequência ou RFID (do inglês "Radio-Frequency IDentification") é um método de identificação automática através de sinais de rádio

3.5 Comparação entre trabalhos relacionados

Na tabela 3.1 é apresentada uma comparação entre os quatro trabalhos relacionados referenciados. Os trabalhos foram comparados em função dos tipos de dados de contexto e sua abordagem de representação desses dados.

Trabalhos Relacionados	Representação do Contexto	Tipos de Contexto
[Chen et al. 2003]	Ontologias/RDF	Localização/Perfil
[Vieira et al. 2010]	Ontologias/Lógica	Qualquer
[Bardram 2004]	Independente	Localização/Biométrico/Perfil
[Tesoriero et al. 2008]	Gráfico/UML	Localização/Perfil

Tabela 3.1: Comparação entre os modelos de orientação a contexto de trabalhos relacionados.

Em [Chen et al. 2003], o tipo de representação de contexto escolhido foi a ontologia codificada com RDF. A pesquisa de [Chen et al. 2003] explorou dados de contexto sobre localização e perfis pessoais e profissionais com o objetivo de prestar um serviço de recomendação ao usuário.

O trabalho de [Vieira et al. 2010] foi criar uma extensão para um framework de desenvolvimento de agentes inteligentes, de modo que ontologias pudessem ser usadas em inferências. Logo, as representações de contexto utilizadas em [Vieira et al. 2010] foram ontologias e lógica, pois os dados de contexto também podem ser representados como fatos em uma expressão lógica.

Em [Bardram 2004] é apresentada uma infraestrutura voltada para descoberta de serviços de contexto, ou seja, essa abordagem é independente de modelo de representação. Os tipos de dados de contexto utilizados durantes a pesquisa foram dados de localização, dados de perfis de pacientes e dados de leitores biométricos.

O trabalho desenvolvido em [Tesoriero et al. 2008] consiste em um sistema desenvolvido para enriquecer a experiência de visitantes de um museu, utilizando-se de dispositivos móveis (PDAs) para mostrar informações adicionais sobre as obras expostas no museu. Baseando-se em informações de localização por radiofrequência (RFID), a localização do visitante é adquirida pelo sistema que envia informações sobre as obras próximas ao usuário.

Os trabalhos comparados na tabela 3.1 possuem diferentes abordagens de representação de contexto e ilustram bem o problema de desenvolver sistemas orientados a contexto. Não existem padrões unânimes para desenvolvimento desses sistemas e a melhor abordagem deve ser escolhida caso a caso.

3.6 Considerações

Neste capítulo foram apresentados alguns conceitos básicos sobre agentes inteligentes, sistemas multiagente e agentes BDI assim como alguns trabalhos relacionados que tratam de sistemas orientados a contexto.

Quanto aos trabalhos relacionados, foram incluídos aqueles que pudessem mostrar de forma representativa os conceitos apresentados nos dois últimos capítulos. Os trabalhos de [Bardram 2004] e [Tesoriero et al. 2008] se mostraram especialmente interessantes pelo fato de mostrarem com profundidade todas as etapas do desenvolvimento do sistema.

No próximo capítulo, será apresentada uma proposta de estudo de caso visando ao desenvolvimento de um sistema orientado a contexto utilizando os conceitos apresentados nos capítulos anteriores.

4 Proposta

O tema escolhido para o desenvolvimento do estudo de caso deste trabalho foi um sistema de sugestões. Este sistema permitiria a um cliente hipotético receber dicas e sugestões sobre o estabelecimento em que se encontra por meio de seu dispositivo móvel.

4.1 Arquitetura de Hardware

A arquitetura proposta para a realização do estudo de caso está ilustrada na figura 4.1. A idéia é ler os dados de sensores e enviá-los a um computador por meio de dois transmissores (um ligado aos sensores e outro ligado ao computador).

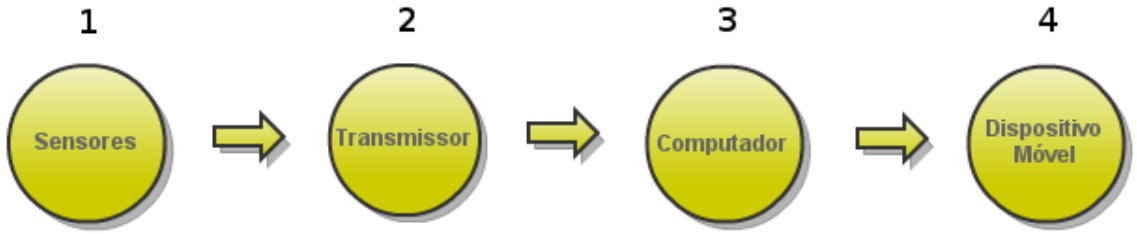

Figura 4.1: Planejamento da arquitetura do estudo de caso

Estes dados serão processados pelo computador que tomará decisões baseadas no contexto e notificará o usuário no seu dispositivo móvel quando julgar necessário.

1. Sensores obterão dados sobre o contexto e enviarão estes dados para um transmissor.
2. O transmissor ligado aos sensores enviará seus dados até o outro transmissor ligado a um computador.

3. Computador onde serão realizadas as verificações e decisões com base no contexto recebido dos sensores e de outras fontes de dados.

4. As notificações sobre mudança no contexto e outras informações relevantes serão transmitidas para o dispositivo móvel do usuário.

Nesta etapa, os componentes de hardware foram tratados de forma abstrata para facilitar a compreensão do sistema. No próximo capítulo, serão mostrados os componentes escolhidos para implementação da proposta e os motivos para a escolha destes componente de hardware.

4.2 Arquitetura de Software

Para realizar a implementação da parte de software deste estudo de caso, foi utilizado o paradigma de orientação a agentes.

Desta forma, os objetivos do sistema foram estabelecidos como pode ser visto na figura 4.2.

O objetivo principal aparece na parte superior da figura 4.2 e foi quebrado em sub-objetivos com o intuito de criar pequenas tarefas que pudessem ser realizadas por um agente individual.

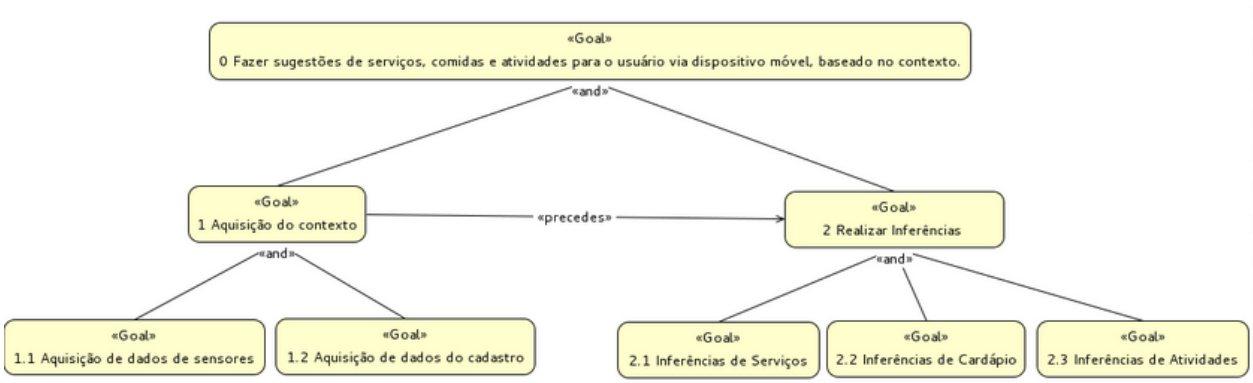

Figura 4.2: Diagrama de objetivo dos agentes no sistema.

O objetivo principal foi considerado : ”Fazer sugestões de serviços, comidas e atividades para o usuário via dispositivo móvel, baseado no contexto”.

A partir desse objetivo surgiram outros dois sub-objetivos: ”Aquisição do contexto” e ”Realizar Inferências”.

A tarefa relativa a adquirir dados de contexto deve preceder a de realização de inferências, pois obviamente os dados precisam existir no sistema antes de poderem ser utilizados, esta resalva esta representada na figura 4.2 por meio da aresta ”precedes”.

Os demais nodos da árvore de objetivos são também refinamentos dos objetivos superiores. A tarefa de adquirir contexto foi quebrada em duas outras: ”Aquisição de dados de sensores” e ”Aquisição de dados de cadastro”. Já a tarefa de realização de inferências foi quebrada em três: ”Inferências de Serviços”, ”Inferências de Cardápio” e ”Inferências de Atividades”.

Com a divisão dos objetivos principais em sub-objetivos, o problema de imaginar uma arquitetura orientada a agente tornou-se mais fácil, pois tarefas específicas foram surgindo ao longo deste processo.

A figura 4.3 mostra um diagrama de papéis imaginados para os diversos agentes exercerem as funções descritas nos objetivos do sistema.

Foram especificados cinco tipos diferentes de agentes para realizar as tarefas necessárias para que de forma conjunta possam atingir o objetivo principal, que seria ”Fazer sugestões de serviços, comidas e atividades para o usuário via dispositivo móvel, baseado no contexto”.

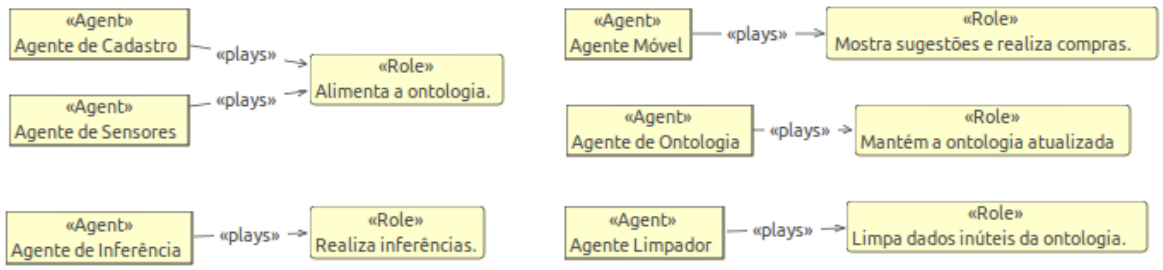

Figura 4.3: Diagrama de papéis dos agentes no sistema.

Na figura 4.3, os agentes estão representados nos retângulos com a marca <<Agent>> e os seus papéis descritos nos retângulos com a marca <<Role>> que está ligada por uma aresta com <<plays>>.

Os agentes de cadastro e de sensores tem basicamente o mesmo papel, embora eles sejam desempenhados de maneira diferente. O agente de cadastro deve receber os dados preenchidos em um formulário de cadastro, realizar as validações necessárias e notificar o agente de ontologia para persistir os dados, já o agente de sensores deve receber os dados enviados pela porta serial, processá-los e enviá-los também ao agente de ontologia.

O agente de ontologia é responsável por deixar a ontologia atualizada, realizando inserções e exclusões de acordo com a necessidade dos outros agentes.

O agente limpador é responsável por disparar eventos de exclusão de informações contidas na ontologia, para tanto ele deve se comunicar o agente de ontologia para que este realize as devidas exclusões.

Os agentes de inferência são responsáveis por disparar eventos de inferência e enviar possíveis sugestões sobre o hotel a clientes específicos. Os agentes de inferência se comunicam com o agente de ontologia para recuperar informações de contexto.

Por último, o agente móvel deve mostrar as informações das sugestões enviadas a ele por meio de sua interface e retornar possíveis interações do cliente com o sistema.

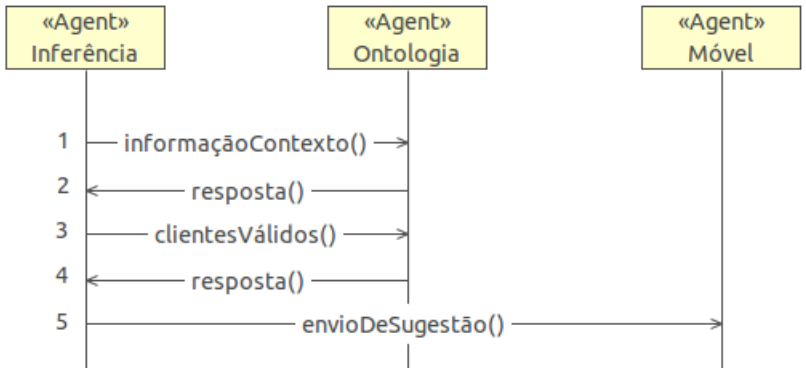

Figura 4.4: Diagrama de sequência ilustrando envio de sugestão.

Na figura 4.4 é ilustrada a sequência de mensagens trocadas pelos agentes do sistema para realizar uma sugestão.

1) Quando o agente de inferência acorda para mandar uma de suas sugestões, uma mensagem é enviada para o agente de ontologia com o objetivo de verificar o contexto climático.

2) Se o contexto não estiver se acordo com o contexto especificado no cadastro da sugestão a operação acaba aqui.

3) Caso o contexto climático atual seja compatível com o contexto especificado no cadastro da sugestão, então uma nova mensagem é enviada do agente de contexto para o agente de ontologia para que o usuário dos clientes com o perfil da sugestão sejam retornados.

4) Os usuários dos clientes que possuem o perfil compatível com o perfil especificado no cadastro da sugestão são retornados pelo agente de ontologia para o agente de inferência.

5) O agente de inferência envia uma mensagem contendo os dados da sugestão a cada um dos agentes móveis representando cada um dos clientes com perfil adequado.

Na figura 4.5 estão descritos as mensagens trocadas pelo agente de inferência e o agente móvel na operação de compra de um dos itens.

1) O agente de inferências envia a mensagem contendo a sugestão para o agente móvel que mostra os dados do item através de sua interface gráfica.

2) O usuário tem a opção de comprar o item clicando no botão ”comprar”. Se o item for comprado pelo cliente, uma mensagem é enviada do agente móvel para o agente de inferência, avisando sobre a compra do item.

3) Ao receber a mensagem de compra do item, o agente de inferência notifica o agente de ontologia para que este registre a compra realizada pelo cliente na ontologia do sistema.

4) O agente de ontologia, após registrar a compra, retorna uma mensagem de confirmação de registro para o agente de inferência.

5) Ao receber a confirmação de registro por parte do agente de ontologia, o agente de inferência envia uma mensagem ao agente móvel confirmando a compra.

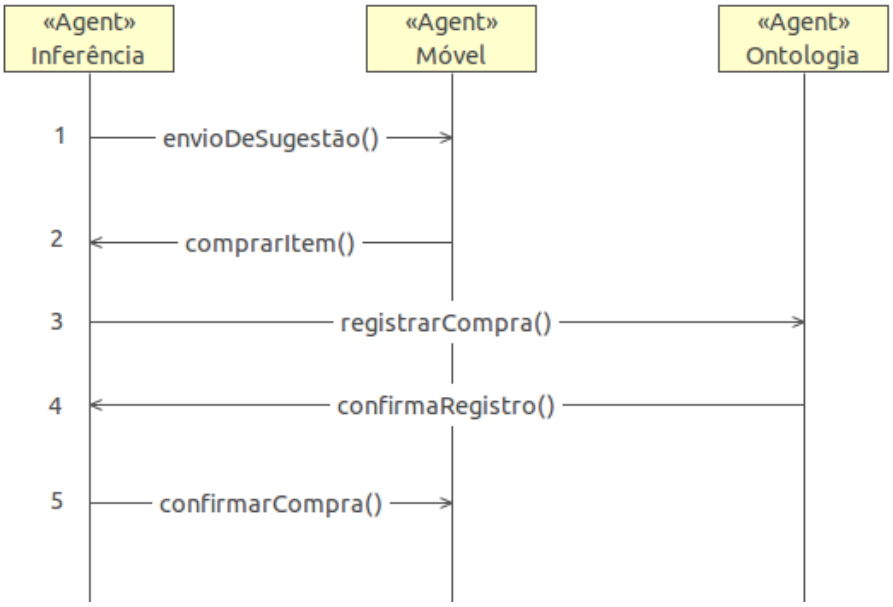

Figura 4.5: Diagrama de sequência ilustrando compra de um item sugerido.

4.3 Considerações da Proposta

A proposta deste trabalho foi idealizada com o objetivo de explorar o uso de ontologias e sistemas multiagente no desenvolvimento de sistemas orientados a contexto. Para tanto, a proposta foi pensada de forma que pudessem ser analisadas as principais etapas de um sistema orientado a contexto: aquisição, representação, interpretação e adaptação ao contexto.

A etapa de aquisição de contexto é realizada junto aos agentes de sensores e de cadastro. A etapa de representação do contexto será administrada pelo agente de ontologia. Já as etapas de interpretação e adaptação serão realizadas pelos agentes de inferência e pelo agente móvel, repectivamente.

No próximo capítulo serão descritas as formas de implementação da proposta, os artefatos de software desenvolvidos, os dispositivos de hardware utilizados e as referentes motivações para o uso de cada abordagem.

5 *Ambiente e Resultados Experimentais*

Como previsto anteriormente, foram desenvolvidos agentes para realizar as diversas tarefas dentro do sistema. O framework JADE foi escolhido para realizar a implementação destes agentes devido a sua simplicidade e sua integração com o sistema operacional Android. Desta forma foi possível fazer a comunicação entre os agentes do servidor com o cliente do celular de forma simples e confiável através da rede wireless.

5.1 Projeto de Hardware

A implementação da proposta de hardware foi realizada utilizando transmissores Garabee em conjunto com uma placa de prototipagem Arduino.

A escolha do Arduino e dos transmissores Garabee foi feita devido a baixa complexidade na manipulação desses componentes com base em experiências anteriores de utilização.

Foram usados quatro tipos de sensores ligados ao sistema: sensores de chuva, temperatura, umidade e luminosidade.

O projeto da parte que integra os sensores com o trasmissor Garabee é semelhante ao mostrado na figura 5.1, com os sensores ligados ao Arduino por meio de um protoboard e o circuito sendo alimentado eletricamente pelo próprio Arduino.

Na memória do Arduino será inserido um programa responsável por ler os valores obtidos por cada sensor e transformá-los em texto que será transmitido de forma serial pelo transmissor Garabee.

Desta forma, informações como luminosidade do ambiente, temperatura e umidade podem ser utilizados como dados para gerar possíveis inferências sobre o contexto.

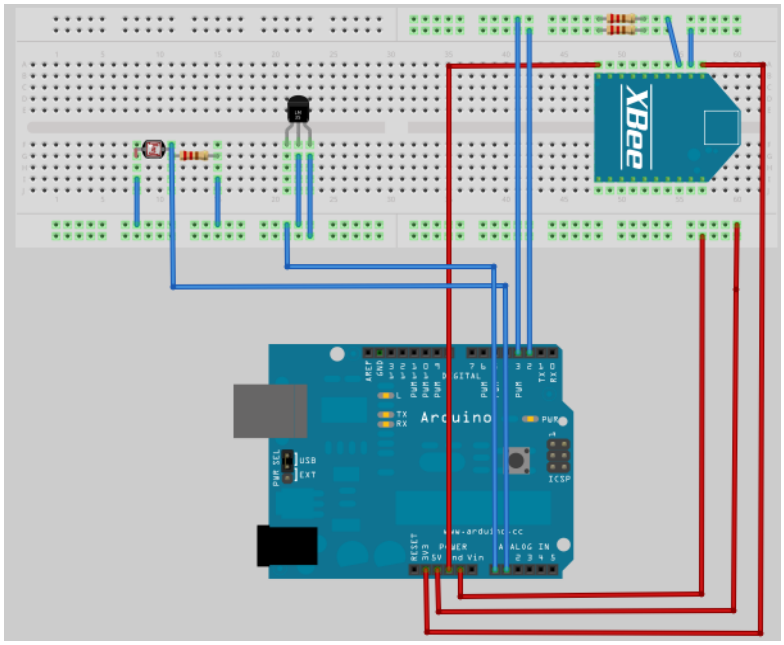

Figura 5.1: Projeto para captação de dados dos sensores.

Na figura 5.2 pode ser vista uma foto do ambiente computacional utilizado para a impletementação do sistema proposto. O notebook, à esquerda, foi utilizado para rodar as duas aplicações (cliente e servidor), os sensores podem ser vistos à direita ligados ao protoboard juntamente com o Arduíno e os módulos de transmissão sem fio Garabee. O cabo que está ligado ao Arduino e ao notebook está simplesmente o alimentando, uma vez que não ocorre transmissão de dados por este cabo. Toda a transmissão de dados é realizada pelos módulos Garabee presentes no protoboard e no notebook.

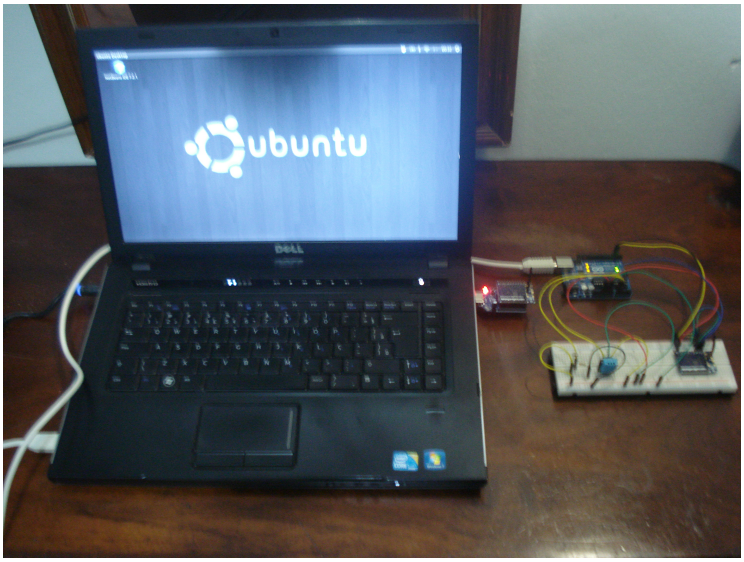

Figura 5.2: Ambiente computacional utilizado no experimento.

5.2 Projeto da Ontologia

A ontologia usada para representação do contexto foi desenvolvida no software de edição Protegé. Para realização das consultas sobre a ontologia foram criadas propriedades para relacionar os dados inseridos na ontologia e atribuir valor semântico a esses dados. Algumas dessas propriedades podem ser vistas na tabela abaixo:

Propriedade	Descrição
ontology:dataDeInicio	usado para especificar data de início de um evento.
ontology:dataDeFim	usado para especificar data de fim de um evento.
ontology:emprego	especifica a profissão de um cliente
ontology:estadoCivil	especifica o estado civil de um cliente
ontology:hora	especifica a hora de um evento
ontology:id_usuario	especifica o nome de usuário utilizado para acessar o sistema
ontology:idade	especifica a idade de um cliente
ontology:nome	especifica o nome de um cliente
ontology:preço	especifica o preço de um item de sugestão
ontology:sexo	especifica o sexo de um cliente
ontology:tempMax	usado para especificar a temperatura máxima de um contexto
ontology:tempMin	usado para especificar a temperatura mínima de um contexto

Tabela 5.1: Propriedades criadas na ontologia do sistema de sugestões.

Por outro lado, os indivíduos (como novos clientes e itens de sugestão) da ontologia foram criados de forma dinâmica, utilizando a linguagem de atualização do SPARQL por meio do agente de ontologia. Na próxima seção são descritas as consultas e atualizações desenvolvidas para utilizar a ontologia como forma de representação de contexto.

5.3 JENA e Consultas a Ontologia

Para realizar as consultas à ontologia foi usado o framework JENA. O motivo da escolha do JENA foi sua boa documentação e suporte a OWL.

O JENA permite a realização de consultas SPARQL e atualizações SPARUL na ontologia através do código Java. Uma das consultas realizadas pelo agente de ontologia pode ser vista na figura 5.3.

```
String queryString =
"PREFIX ontology: <" + ns + ">" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"SELECT ?item ?price ?description " +
"WHERE {" +
"      ?item      rdf:type              ontology:" + type + " ." +
"      ?item      ontology:preco        ?price ." +
"      ?item      ontology:descricao    ?description ." +
"      } ";

Query query = QueryFactory.create(queryString);
QueryExecution qe = QueryExecutionFactory.create(query, ontology);
ResultSet results2 = qe.execSelect();
```

Figura 5.3: Exemplo de consulta usada no agente de ontologia.

A consulta da figura 5.3 retorna os atributos ”item”, ”price” e ”description” para um item de um determinado tipo. No caso, o agente de ontologia recebe o tipo do item do qual deseja-se saber os dados, monta um string Java contendo a consulta SPARQL necessária e executa esta consulta.

A consulta SPARQL é montada utilizando propriedades do próprio RDF, como ”rdf:type”, e também propriedades criadas na ontologia desse projeto, como ”ontology:descricao”. Quando a consulta é executada, os padrões de triplas especificados na consultas são procurados na ontologia passada como parâmetro e os dados encontrados são retornados nas variáveis ”?item”, ”?price” e ”?description”.

Na figura 5.4 é vista uma atualização executada pelo agente de ontologia a pedido do agente limpador. Esta atualização apaga da ontologia todos os dados sobre um item de sugestão cujo tempo de validade foi atingido. Essas exclusões foram pensadas como necessárias para manutenção do tamanho ontologia e performance do sistema.

```
String update =
"PREFIX ontology: <" + ns + ">" +
"PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
"DELETE WHERE { ontology:" + type + " ?p ?o } " +
"DELETE WHERE { ontology:" + type + "_Context ?p ?o } ";

UpdateAction.parseExecute(update, ontology);
```

Figura 5.4: Exemplo de atualização usada no agente de ontologia.

5.4 JADE (Java Agent DEvelopment Framework)

Na figura 5.5 é possível visualizar os diferentes agentes do sistema através da ferramenta de gerenciamento existente no framework JADE. Os agentes mostrados abaixo da pasta ”Main_Container” são agentes criados pelo próprio JADE para auto-gerenciamento, já os agentes contidos em ”BE_hellsgate_1099_2” e ”BE_hellsgate_1099_3” são respectivamente os agentes do servidor e do cliente de celular.

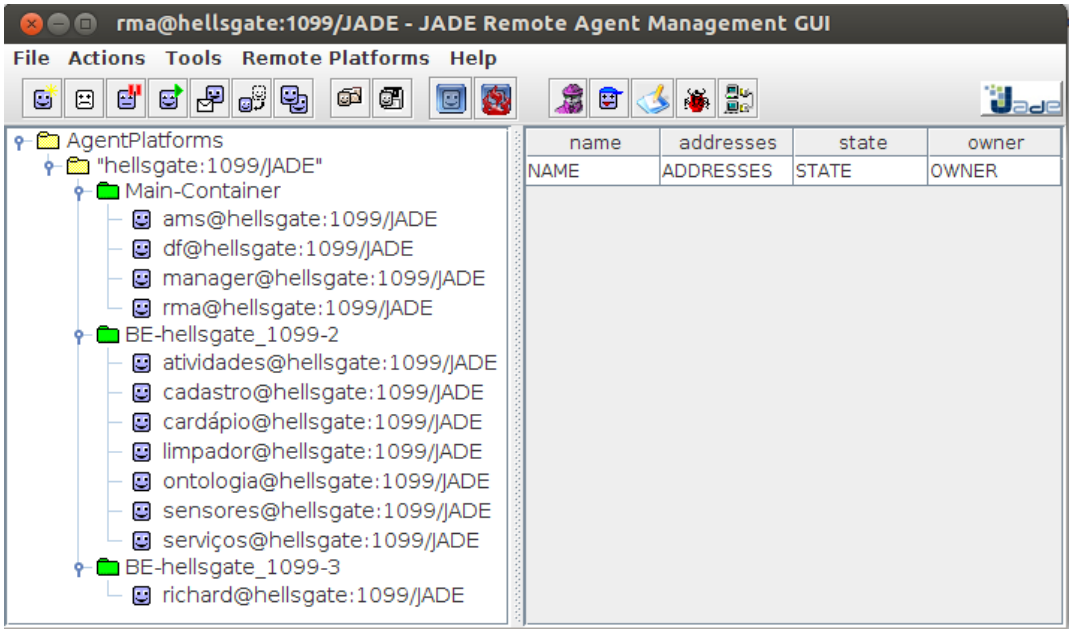

Figura 5.5: Plataforma de gerenciamento do framework JADE.

Para inserção de dados foi desenvolvida uma interface em Java Swing. Nesta interface é possível inserir dados pessoais como nome, idade e profissão do cliente, assim como dados do sistema como nome de usuário e senha para que o cliente possa acessar o sistema a partir do seu celular.

Os dados coletados durante a fase de cadastro do cliente podem ser usados como parte do contexto utilizado para realizar inferências e sugerir produtos oferecidos pelo hotel.

Uma vez que os dados do cliente foram preenchidos e o botão ”Cadastrar” foi acionado, os dados serão gravados na ontologia para serem utilizados posteriomente no processo de busca de clientes e envio de sugestões.

Como este estudo de caso ainda está sendo desenvolvido, algumas funções de validação não foram implementadas durante as operações de cadastro, de forma que dados errados podem ser inseridos, porém foram priorizadas outras funcionalidades do sistema em detrimento destas funções de validação.

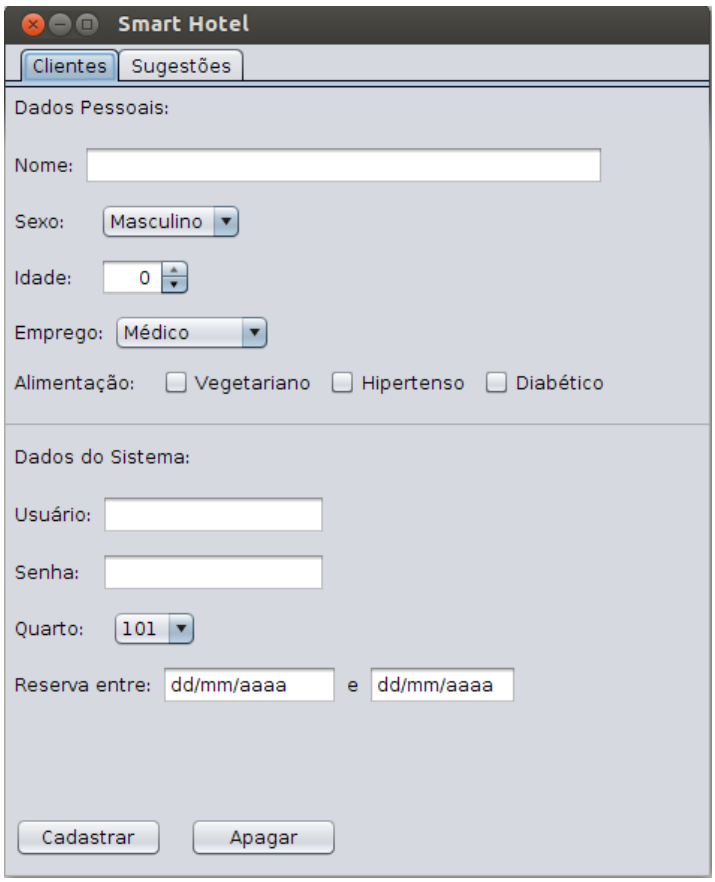

Figura 5.6: Tela de cadastro de clientes no sistema.

A segunda aba da interface Swing tem como objetivo inserir dados sobre atividades, cardápio e serviços oferecidos pelo hotel para que o sistema tente encontrar possíveis compradores desses itens entre os clientes cadastrados do hotel.

Na figura 5.7 pode ser visto que dados como nome, preço e descrição podem ser atribuídos a um item de sugestão, de forma que essas informações possam ser mostradas ao cliente que acessar o aplicativo de celular.

Também devem ser inseridos dados de contexto de cada sugestão cadastrada no sistema, desta forma os agentes de inferência poderão tentar encontrar possíveis clientes para um item sugerido.

Dados sobre as condições climáticas e o horário em que tais sugestões devem ser enviadas também podem ser configuradas a partir desta interface.

Cada um dos agentes de inferência registra o horário em que as suas sugestões devem ser enviadas. Quanto o momento de enviar a sugestão chega, o agente de inferência se comunicará com o agente de ontologia para saber se deve enviar e a quem enviar essa sugestão.

Figura 5.7: Tela de cadastro de sugestões no sistema.

O botão de ”apagar” presente tanto na aba de cadastro de usuários quanto na aba de cadastro de susgestões tem a função de meramente limpar os campos de texto do formulário, tornando a atividade de cadastrar vários clientes e sugestões mais fácil.

5.5 JADE Android

Para o desenvolvimento do cliente móvel foi utilizada a biblioteca Jade Android desenvolvida pelo grupo [Telecom Italia] , mesmo grupo de desenvolvedores que criaram o JADE. Com essa biblioteca torna-se possível a criação de agentes móveis na plataforma Android por meio da abordagem de container dividido. Nesta abordagem, as funcionalidades do agente ficam divididas entre o cliente e o servidor, tornando o consumo de recursos menor no dispositivo móvel em que o agente se encontra.

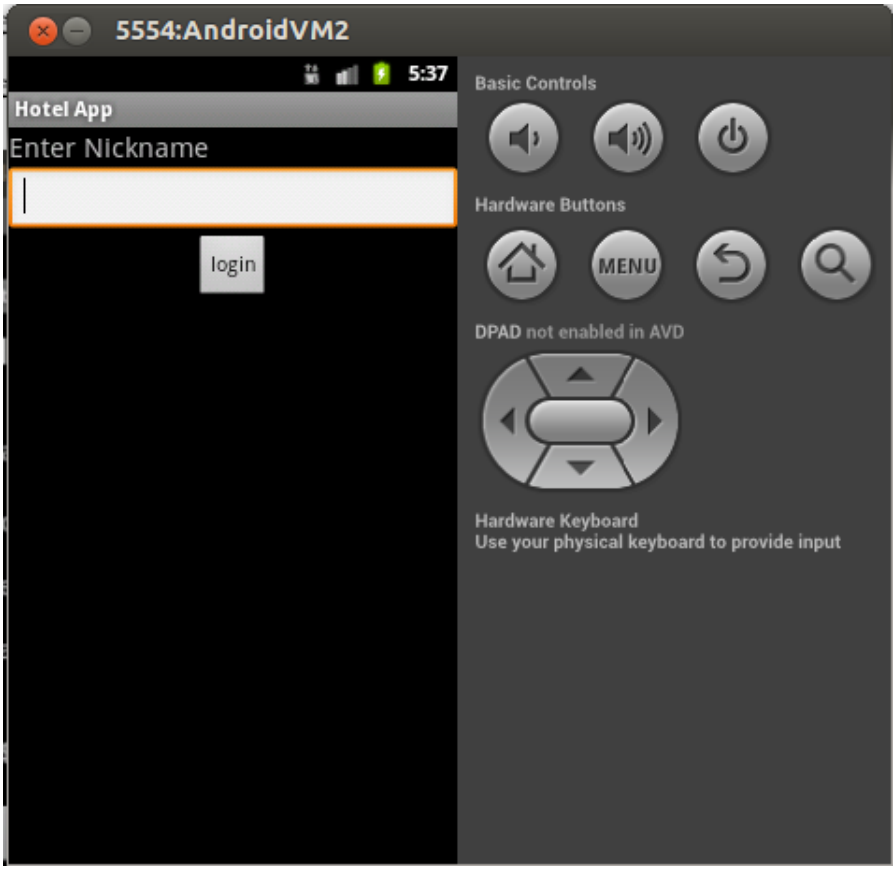

Figura 5.8: Tela de acesso do cliente móvel.

A primeira tela do aplicativo desenvolvido é a tela de login. Nesta tela o cliente deve fornecer o usuário cadastrado para ter acesso ao sistema de sugestões do hotel.

O nome de usuário utilizado para acesso ao sistema também foi utilizado para dar nome ao agente móvel de cada cliente. Desta forma as sugestões podem ser enviadas utilizando o nome do agente móvel existente na ontologia e disponível na plataforma JADE.

Dessa forma, um agente de inferência pode mandar uma mensagem para um cliente cadastrado com usuário ”richard” utilizando uma simples chamada send(mensagem, ”richard”), e a plataforma JADE se encarrega de manter dados sobre localização de dispositivos e entrega de

mensagens.

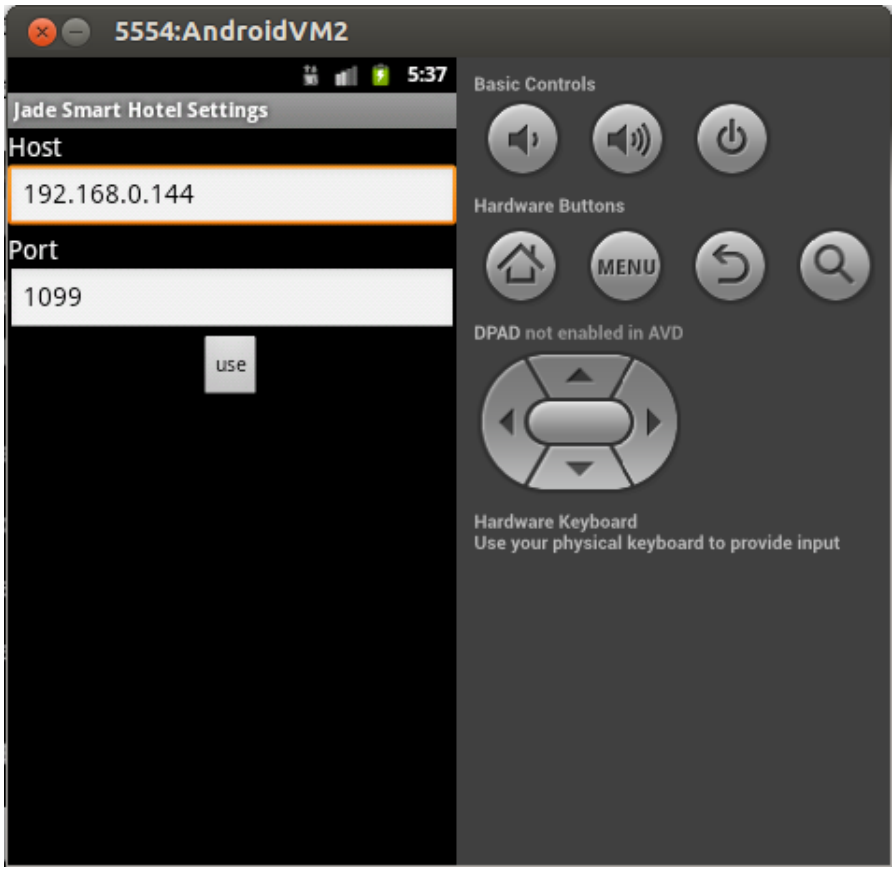

Figura 5.9: Tela de configuração de IP do cliente móvel.

A tela vista na figura 5.9 é acessada pressionando o botão ”menu” do dispositivo com Android. Nesta tela é possível editar o IP e porta do servidor rodando a plataforma JADE, para que seja possível a comunicação entre o agente móvel e o resto do sistema.

Essa tela de configuração não precisa ser conhecida pelo usuário caso o IP e a porta do servidor sejam fixos. Neste caso, os números de IP e porta podem ser configurados para um valor padrão.

Uma vez configurados os valores de IP e porta do servidor, o usuário deve pressionar o botão ”use” para voltar então na tela de acesso ao sistema.

Na figura 5.10 é possível ver a tela principal do aplicativo para Android. Existem três abas, ua para cada tipo de sugestão existente no sistema: atividades, cardápio e serviços. Com um clique em uma das abas é mostrada uma listagem das sugestões recebidas até o momento nesta classe de sugestões.

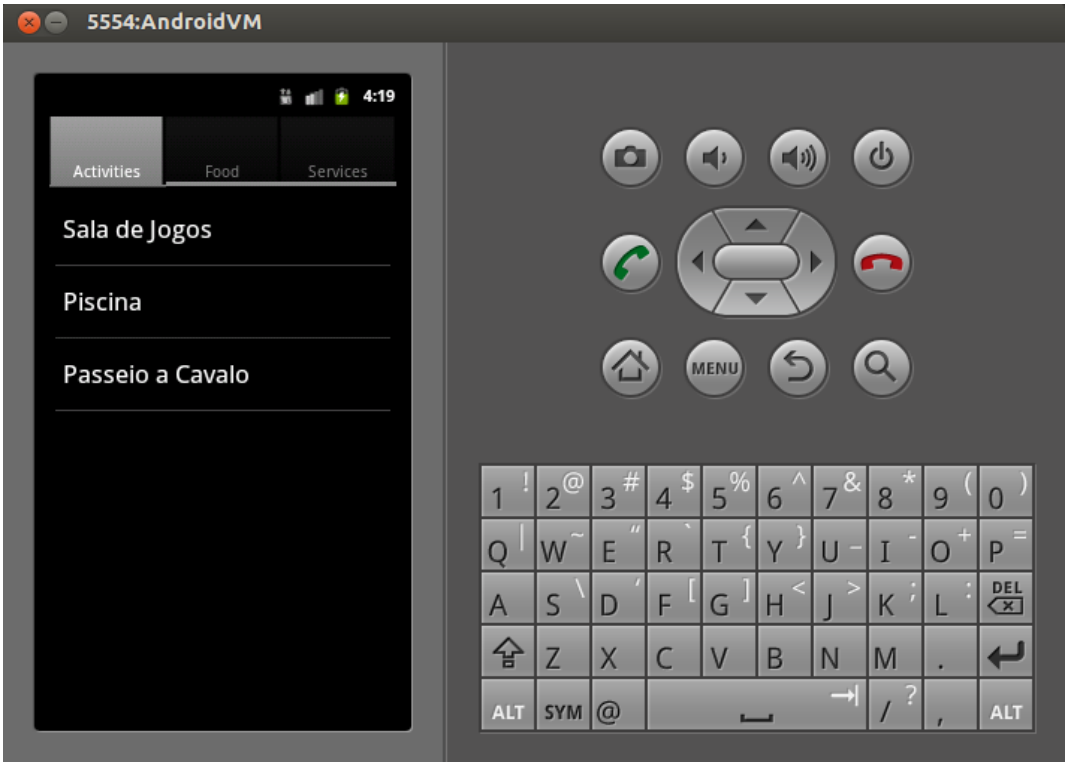

Figura 5.10: Exemplo de sugestões de atividades enviadas ao cliente móvel.

As sugestões são mostradas sem nenhum tipo de ordenação, cabe ao usuário selecionar quais sugestões lhe chamam atenção. Ao clicar no título de uma das sugestões, o aplicativo imediatamente apresenta uma outra tela com a descrição e o preço do item selecionado, como pode ser visto na figura 5.11.

Outro recurso deste aplicativo é a possibilidade de comprar o item sugerido pelo sistema. Neste caso, o usuário deve pressionar o botão ”comprar” presente na tela de descrição do item de interesse. Com esse gesto, uma mensagem é enviada pelo agente móvel ao agente de ontologia, que registra o pedido do cliente, retornando uma mensagem de confirmação.

O objetivo deste aplicativo é trazer ao conhecimento do cliente do estabelecimento os serviços oferecidos, aumentando assim a satisfação do cliente e o lucro do estabelecimento.

Figura 5.11: Exemplo da tela contendo a descrição da sugestão enviada ao cliente móvel.

5.6 Caso de Uso

Nessa seção é apresentado um caso de uso que demonstra o funcionamento do sistema desenvolvido utilizando dados de clientes e sugestões hipotéticos, como pode ser observado nas tabelas 5.2 e 5.3.

Foram criados clientes e sugestões com perfis diferentes de forma proposital, de forma que pudesse ser observada a interpretação do contexto realizada pelos agentes do sistema na entrega de sugestões de atividades.

Pessoa	Sexo	Idade	Formação	Alimentação
1	Masculino	63	Engenheiro	Sem restrições
2	Masculino	27	Estudante	Diabético
3	Feminino	52	Médica	Vegetariana
4	Feminino	34	Professora	Sem restrições

Tabela 5.2: Pessoas hipotéticas utilizados no caso de uso.

Sugestão	Nome	Contexto Climático	Perfil do Cliente
a	Palestra sobre medicina	Sem restrições	Emprego = Médico
b	Quadra de Esportes	Sem chuva	Idade < 50
c	Sala de Jogos	Sem restrições	Sem restrições

Tabela 5.3: Sugestões hipotéticas utilizados no caso de uso.

1) Dois novos clientes são cadastrados no sistema, de acordo com a tabela 5.2. Suas informações de perfil são armazenadas na ontologia.

2a) Uma sugestão de uma palestra sobre medicina é cadastrada no sistema. Essa sugestão não possui restrições climáticas mas deve ser enviada apenas para clientes médicos.

2b) É cadastrada no sistema uma sugestão sobre a quadra de esportes existente no hotel. Essa sugestão só deve ser enviada para clientes com menos de 50 anos e se não estiver chovendo, pois a quadra não é coberta.

2c) Uma sugestão é cadastrada no sistema sobre a sala de jogos existente no hotel. Essa sugestão não possui restrições climáticas e pode ser enviada para clientes em qualquer faixa etária.

3) O momento de enviar a sugestão cadastrada na etapa 2 chega, se o perfil climático desta sugestão não for compatível com o contexto climático atual, então a sugestão não é enviada.

4a) Os clientes com perfil compatível com a sugestão são buscados. A única pessoa que tem medicina como profissão é a pessoa 3, portanto ela é a única que recebe a sugestão sobre a palestra na área de medicina.

4b) Os clientes com perfil compatível com a sugestão são buscados. As únicas pessoas com menos de 50 anos de idade são as pessoas 2 e 4, portanto elas serão as únicas a receber a sugestão de visitar a quadra de esportes do hotel.

4c) Não existe nenhuma restrição de perfil associada a sugestão de visita a sala de jogos do hotel, portanto todos os clientes cadastrados no sistema receberão essa sugestão.

Dessa forma, o funcionamento do sistema deu-se de acordo com a especificação da proposta. O mecanismo de envio de sugestões usado para sugestões de cardápio e de serviços é análogo ao mostrado neste caso de uso.

5.7 Considerações do capítulo

Neste capítulo foi descrita a forma de implementação da proposta especificada anteriormente. O projeto foi desenvolvido visando o uso de ferramentas de código livre e boa documentação na internet.

Existem vários frameworks de orientação a contexto, mas não foi encontrado nenhum que tivesse boa documentação e fosse amplamente utilizado por desenvolvedores de software. Nesse caso, os frameworks JADE e JENA cumpriram bem o papel.

No próximo capítulo serão discutidas as conclusões obtidas a partir do experimento e os possíveis trabalhos futuros.

6 Conclusão

Dos trabalhos relacionados, o que mais se assemelha ao estudo de caso desenvolvido é o de [Chen et al. 2003], no qual foi desenvolvido um sistema de sugestões utilizando a própria ontologia para gerar novas triplas utilizando as triplas já existentes, caracterizando o processo de inferência. Diferentemente desta abordagem, no estudo de caso deste trabalho as inferências não são feitas diretamente na ontologia, mas sim nos agentes de inferência que tomam decisões sobre mandar ou não mandar sugestão com base nas informações obtidas da ontologia.

Durante o desenvolvimento do trabalho foi notado o potencial de ontologias em sistemas orientados a contexto. Com ontologias, é possível estabelecer relações e atribuir significados aos dados de contexto, essa possibilidade mostra-se muito interessante em sistemas orientados a contexto, pois permite que sejam explorados detalhes no modelo de dados que dificilmente poderiam ser explorados em bancos de dados relacionais, por exemplo.

Uma das dificuldades de se utilizar ontologias é se acostumar com ambientes para edição e a linguagem de consultas a ontologia. No estudo de caso foi utilizada uma implementação da linguagem SPARQL provida pelo framework JENA, este framework mostrou-se de fácil utilização e possui boa documentação.

Quanto a abordagem de desenvolvimento orientada a agentes, esta mostrou-se benéfica durante a fase de idealização da proposta, pois a divisão de objetivos e papéis dentro do sistema ajudou a diminuir a complexidade de implementação do sistema. Por outro lado, a utilização do framework JADE para implementação dos agentes mostrou-se desafiadora em alguns pontos uma vez que exige do programador atenção para aspectos do sistema como sincronização na troca de mensagens e outros detalhes de implementação que poderiam ser tratados de maneira automática pelo framework. De qualquer forma, o JADE é uma boa ferramenta para desenvolvimento de sistemas multiagente e seu uso talvez seja mais aconselhável em sistemas maiores, uma vez que pode adicionar uma complexidade desnecessária em projetos menores.

6.1 Trabalhos Futuros

Para continuidade do desenvolvimento do sistema de sugestões desenvolvido como estudo de caso deste trabalho, pretende-se implementar algumas outras funcionalidades como a integração com Google Maps para localização de locais próximos e a possibilidade de requisitar sugestões em um período de tempo determinado pelo cliente.

Outro aspecto da orientação a contexto que deve ser explorado é o histórico de contexto. No estudo de caso foram utilizados dados de sensores que sempre se sobrescreviam na ontologia, deixando sempre os valores de contexto climático atualizados. Portanto, o estudo da utilização de histórico de contexto e o seu impacto no desenvolvimento de software orientado a contexto permace como possibilidade de pesquisa.

Um estudo sobre privacidade em sistemas orientados a contexto também seria possível, pois como mostrado na tabela 2.1, os dados de entidades participantes do sistema são muitas vezes privados e devem ser protegidos. O estudo de abordagens para controle de privacidade em orientação a contexto pode servir como tema para trabalhos futuros uma vez que não foi levado em consideração durante o desenvolvimento do estudo de caso.

Referências Bibliográficas

[Ay 2007]AY, F. *Context Modeling and Reasoning using Ontologies*. [S.l.], jul. 2007. Disponível em: <http://www.ponnuki.de/cmaruo/cmaruo.pdf>.

[Bardram 2004]BARDRAM, J. Applications of context-aware computing in hospital work: examples and design principles. In: ACM. *Proceedings of the 2004 ACM symposium on Applied computing*. [S.l.], 2004. p. 1574–1579.

[Bauer 2003]BAUER, J. *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic*. Tese (Doutorado) — Technische Universitat Berlin, mar. 2003.

[Bellavista et al. 2012]BELLAVISTA, P. et al. Survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, 2012.

[Chen e Kotz 2000]CHEN, G.; KOTZ, D. *A Survey of Context-Aware Mobile Computing Research*. [S.l.], 2000.

[Chen et al. 2003]CHEN, H. et al. Creating context-aware software agents. In: TRUSZKOWSKI, W.; HINCHEY, M.; ROUFF, C. (Ed.). [S.l.]: Springer Berlin / Heidelberg, 2003, (Lecture Notes in Computer Science).

[Dey 2001]DEY, A. K. Understanding and using context. *Springer*, 2001.

[FIPA]FIPA. Disponível em: <http://www.fipa.org>. Acesso em: 16-03-2013.

[Greenfield 2006]GREENFIELD, A. *Everyware - The Dawning Age of Ubiquitous Computing*. [S.l.]: New Riders, 2006.

[Jennings 2000]JENNINGS, N. On agent-based software engineering. *Artificial intelligence*, Elsevier, v. 117, n. 2, p. 277–296, 2000.

[Jennings e Wooldridge 1996]JENNINGS, N.; WOOLDRIDGE, M. Software agents. *IEEE Review*, 1996.

[Malik, Mahmed e Javed 2007]MALIK, N.; MAHMED, U.; JAVED, Y. Future challenges in context-aware computing. *IADIS International Conference*, 2007.

[Nazario, Dantas e Todesco 2012]NAZARIO, D. C.; DANTAS, M. A. R.; TODESCO, J. L. Taxonomia das publicações sobre qualidade de contexto. *SBIJOURNAL*, n. 20, 2012.

[Padovitz, Loke e Zaslavsky 2008]PADOVITZ, A.; LOKE, S. W.; ZASLAVSKY, A. Multiple-agent perspectives in reasoning about situations for context-aware pervasive compting systems. *IEEE Transactions on Systems*, 2008.

[Resatsch 2010]RESATSCH, F. *Ubiquitous Computing - Developing and Evaluating Near Field Communication Applications*. [S.l.]: Springer, 2010.

[Rocha, Lima e Dantas 2011]ROCHA, C. C.; LIMA, J. C. D.; DANTAS, M. A. R. An adaptive authentication service based on mobile user's behavior and spatio-temporal context. *IEEE Symposium on Computers and Communications*, 2011.

[Russel e Norvig 2004]RUSSEL, S.; NORVIG, P. *Inteligência Artificial - Tradução da Segunda Edição*. [S.l.]: Elsevier, 2004.

[Schilit, Adams e Want 1994]SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. *IEEE*, 1994.

[Strang e Linnhoff-Popien 2004]STRANG, T.; LINNHOFF-POPIEN, C. A context modeling survey. In: . [S.l.: s.n.], 2004.

[Telecom Italia]TELECOM Italia. Disponível em: <http://www.telecomitalia.com/>. Acesso em: 16-03-2013.

[Tesoriero et al. 2008]TESORIERO, R. et al. A location-aware system using rfid and mobile devices for art museums. In: IEEE. *Autonomic and Autonomous Systems, 2008. ICAS 2008. Fourth International Conference on*. [S.l.], 2008. p. 76–81.

[Vieira et al. 2010]VIEIRA, R. et al. An agent-oriented programming language for computing in context. In: *IFIP International Federation for Information Processing*. [S.l.: s.n.], 2010. v. 218, n. 1.

[W3C]W3C. Disponível em: <http://www.w3.org>. Acesso em: 16-03-2013.

[Wang et al. 2004]WANG, X. H. et al. Pervasive computing and communications workshops, 2004. proceedings of the second ieee annual conference on. 2004.

[Weiser 1991]WEISER, M. The computer for the 21st century. *Scientific American*, 1991.

[Winograd 2001]WINOGRAD, T. Architectures for context. *Human-Computer Interaction*, 2001.

[Wooldridge 2009]WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: Wiley, 2009.

7 Anexos

7.1 Artigo sobre o TCC

Um Estudo Sobre Orientação a Contexto em Ambientes de Redes Sem Fio

Richard P. Silva[1], Mário A. R. Dantas[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

{richard.ps,mario}@inf.ufsc.br

Abstract. *Currently, the study of new ubiquitous systems building techniques is beeing stimulated by the success of the mobile devices in the electronic's market. One of the most noticeable themes in the last years is the development of context-aware software. However, there are many problems to be solved in this area, for example, the best approach of representation and interpretation of context. The purpose of this work is to do a study about the current techniques used in the context-aware software building and the production of a case study based on the studied techniques.*

Resumo. *Atualmente, o estudo de novas técnicas para construção de sistemas ubíquos vem sendo impulsionado pelo sucesso dos dispositivos móveis no mercado de eletrônicos. Um dos temas em evidência nos últimos anos é o desenvolvimento de software orientado a contexto. Contudo, ainda existem muitos problemas a serem resolvidos nessa área, como a forma ideal de representação e interpretação do contexto. Este trabalho tem como objetivo fazer um estudo das técnicas atuais utilizadas na construção de software orientado a contexto e a produção de um estudo de caso baseado nas técnicas estudadas.*

1. Introdução

A orientação a contexto é um tema atualmente em evidência devido ao grande sucesso dos dispositivos móveis no mercado de eletrônicos. O desenvolvimento de sistemas móveis modernos está relacionado com o avanço nas técnicas de utilização de dados de contexto, como a localização física do dispositivo, preferências do usuário ou qualquer informação relacionada ao contexto em que o sistema esteja inserido.

Muitas pesquisas têm sido realizadas sobre o desenvolvimento de software orientado a contexto, mas existe pouco consenso sobre as melhores estratégias existentes para isso. Este trabalho é um estudo sobre os conceitos, estratégias e tecnologias por trás do desenvolvimento de sistemas computacionais dependentes de contexto.

Um dos objetivos deste trabalho é o desenvolvimento de um estudo de caso utilizando o domínio de redes de sensores sem fio. Neste estudo de caso, será utilizado o método considerado mais adequado para a implementação de um aplicativo que utilize dados de contexto recebidos de uma rede de sensores.

2. Definição de Contexto

Um dos primeiros trabalhos sobre aplicações computacionais orientadas a contexto foi escrito por [Schilit 1994]. Este trabalho definiu contexto como sendo informações sobre localização, identidades de pessoas e objetos próximos entre si, assim como as mudanças nesses objetos.

Em [Dey 2001], foi proposta uma nova definição para contexto, esta definição mostrou-se mais geral e portanto tornou mais fácil para o desenvolvedor de aplicações enumerar os dados de contexto para uma aplicação específica. Segundo [Dey 2001], a definição antes proposta por [Schilit 1994] causava dúvida em desenvolvedores, pois eles não sabiam exatamente se os seus dados se encaixavam na definição de informações de contexto.

A definição de [Dey 2001] diz que: “Contexto é qualquer informação que possa ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é considerado relevante para a interação entre usuário e uma aplicação, incluindo o usuário e a aplicação.” A definição de contexto proposta por [Dey 2001] será utilizada para a realização deste trabalho.

3. Modelos de Representação de Contexto

Neste trabalho, o modelo de representação de contexto utilizado para implementação do estudo de caso foi baseado em ontologias. A existência de um estudo comparativo entre modelos de representação de contexto feito por [Strang e Linnhoff-Popien 2004] contribuiu para a escolha deste modelo.

A tabela 3.1 é uma tradução dos resultados obtidos por [Strang e Linnhoff-Popien 2004] na comparação de diversas técnicas de representação de contexto de acordo com vários critérios de avaliação.

Entre os critérios levados em conta estavam: composicao distribuída (dc), validacao parcial (pv), riqueza e qualidade da informacao (qua), incompleteza e ambiguidade (inc) e nível de formalizacao (for). De acordo com [Strang e Linnhoff-Popien 2004], a melhor nota atribuída foi ++ e a menor nota –-.

Abordagem - Requisitos	dc	pv	qua	inc	for	app
Modelos Chave-Valor	-	-	- -	- -	- -	+
Modelos Marcação	+	+ +	-	-	+	+ +
Modelos Gráficos	- -	-	+	-	+	+
Modelos Orientados a Objeto	+ +	+	+	+	+	+
Modelos Baseados em Lógica	+ +	-	-	-	+ +	-
Modelos Baseados em Ontologias	+ +	+ +	+	+	+ +	+

Tabela 1 - Avaliação dos modelos de representacao de dados de contexto

4. Proposta

O tema escolhido para o desenvolvimento do estudo de caso deste trabalho foi o de um sistema de sugestões para clientes de um hotel. Este sistema permitiria a um cliente de um hotel hipotético receber dicas e sugestões sobre atividades, comidas e serviços oferecidas pelo hotel em seu dispositivo móvel (seja ele um celular ou um tablet) através da rede wireless do hotel.

4.1 Arquitetura de Hardware

A arquitetura proposta para a realização do estudo de caso está ilustrada na figura 1. A ideia é ler os dados de sensores ligados a um Arduino e enviá-los ao computador por meio de dois transmissores sem fio, um ligado a um Arduino e outro ligado a porta USB de um computador. Estes dados serão processados pelo computador que as processará tomando decisões baseadas no contexto e notificará o usuário no seu dispositivo móvel quando julgar necessário.

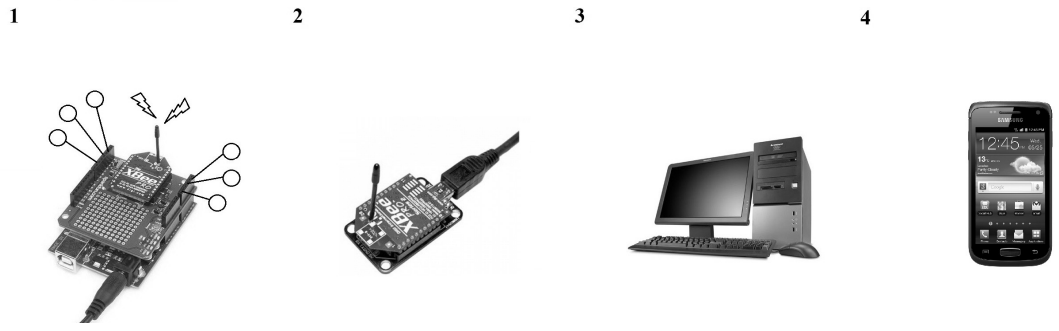

Figura 1 - Planejamento da arquitetura do estudo de caso

4.1 Arquitetura de Software

Foram especificados cinco tipos diferentes de agentes para realizar as tarefas necessárias para que de forma conjunta possam atingir o objetivo principal, que seria "Fazer sugestões de serviços, comidas e atividades para o usuário via dispositivo móvel, baseado no contexto".

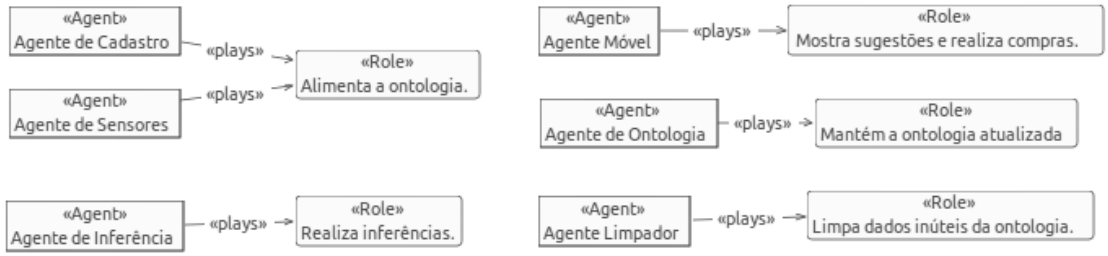

Figura 2 - Diagrama de papéis dos agentes no sistema.

Na figura 2, os agentes estão representados nos retângulos com a marca <<Agent>> e os seus papéis descritos nos retângulos com a marca <<Role>> que está ligada por uma aresta com <<plays>>.

5. Ambiente e Resultados Experimentais

Como previsto anteriormente, foram desenvolvidos agentes para realizar as diversas tarefas dentro do sistema. O framework JADE foi escolhido para realizar a implementação destes agentes devido a sua simplicidade e sua integração com o sistema operacional Android.

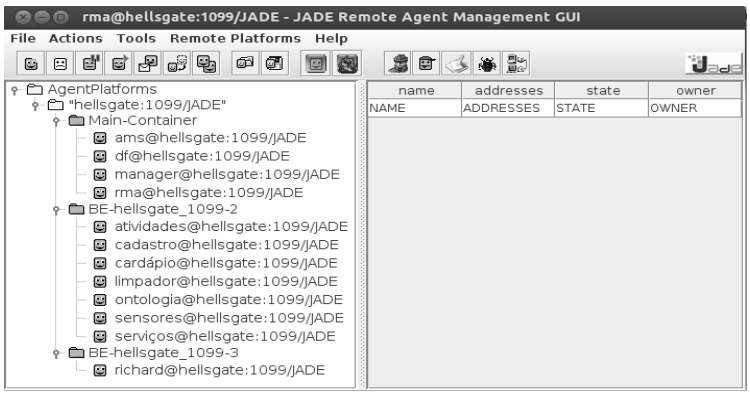

Figura 3 - Plataforma de gerenciamento do framework JADE.

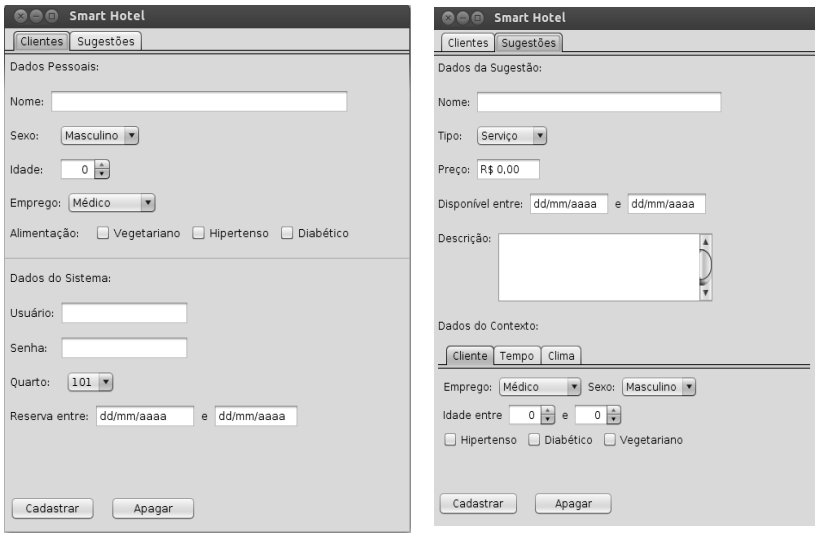

Figura 4 – Telas de cadastro de clientes e sugestões.

Foi desenvolvida um interface para cadastro de clientes e sugestões utilizando a biblioteca Swing e a linguagem de programação Java. No momento do cadastro de uma sugestão, é especificado em que tipo de contexto esta sugestão será enviada ao dispositivo móvel do cliente.

Na figura 5 é possível ver uma listagem de sugestões recebidas pelo protótipo de cliente de celular desenvolvido para este trabalho.

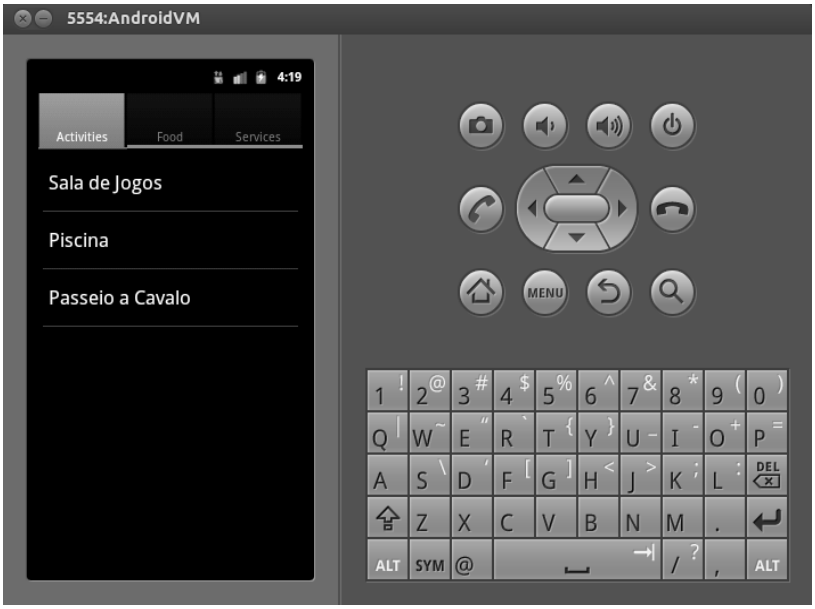

Figura 5 – Máquina virtual Android rodando aplicativo desenvolvido.

Figura 6 – Exemplo de sugestão visualizada na máquina virtual Android.

6. Trabalhos Futuros

Como continuação do trabalho atual, pretendemos ampliar o protótipo do sistema Smart Hotel. Entre as funções que podem ser integradas estão a visualização de mapas com Google Maps e utilização de outras ferramentas web para melhora na experiência do usuário e agregação de mais dados de contexto.

7. Conclusão

Este trabalho teve o objetivo de realizar um estudo sobre as principais formas de implementação de sistemas orientados a contexto na atualidade. Com o desenvolvimento do estudo de caso, foi possível observar a complexidade adicional de sistemas desse tipo, assim como o seu potencial para aplicações comerciais.

Referências

Bellavista, P. et al (2012) “Survey of context data distribution for mobile ubiquitous systems.” Editado por ACM Computing Surveys.

Dey, A. K. (2001) “Understanding and using context.”. Editado por Springer.

Greenfield, A. (2006) “Everyware - The Dawning Age of Ubiquitous Computing”. Editado por New Riders.

Schilit, B., Adams, N. e Want, R. (1994) “Context-aware computing applications” Editado por IEEE.

Strang, T. e Linnhoff-Popien, C. (2004) “A context modeling survey.”

7.2 Código do Estudo de Caso

1 Código do Projeto (Servidor)

1.1 Main

```
package main;

import jade.MicroBoot;
import jade.core.MicroRuntime;

public class Main {

    public static void main(String[] args) {
        MicroBoot.main(args);
        try {
            MicroRuntime.startAgent("cadastro", "agents.
                CadastreAgent", null);
            MicroRuntime.startAgent("sensores", "agents.
                SensorAgent", null);
            MicroRuntime.startAgent("servi os", "agents.
                InferenceAgent", null);
            MicroRuntime.startAgent("atividades", "agents.
                InferenceAgent", null);
            MicroRuntime.startAgent("card pio", "agents.
                InferenceAgent", null);
            MicroRuntime.startAgent("ontologia", "agents.
                OntologyAgent", null);
            MicroRuntime.startAgent("limpador", "agents.
                CleanerAgent", null);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

1.2 Agente de Ontologia

```
package agents;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import utils.ClientRegister;
import utils.Message;
import utils.SuggestionRegister;

```

```
17 import com.hp.hpl.jena.ontology.OntModel;
18 import com.hp.hpl.jena.ontology.OntModelSpec;
19 import com.hp.hpl.jena.query.Query;
20 import com.hp.hpl.jena.query.QueryExecution;
21 import com.hp.hpl.jena.query.QueryExecutionFactory;
22 import com.hp.hpl.jena.query.QueryFactory;
23 import com.hp.hpl.jena.query.QuerySolution;
24 import com.hp.hpl.jena.query.ResultSet;
25 import com.hp.hpl.jena.query.ResultSetFormatter;
26 import com.hp.hpl.jena.rdf.model.ModelFactory;
27 import com.hp.hpl.jena.update.UpdateAction;
28
29
30 import jade.core.AID;
31 import jade.core.Agent;
32 import jade.core.behaviours.CyclicBehaviour;
33 import jade.core.behaviours.OneShotBehaviour;
34 import jade.lang.acl.ACLMessage;
35 import jade.lang.acl.UnreadableException;
36
37 public class OntologyAgent extends Agent {
38
39     private final String ns = "http://www.semanticweb.org/
        ontologies/" +
40                               "2012/9/Ontology1350055633402.
                                     owl#";
41
42     private static final String SERVICES_LIST = "
        services_list";
43     private ACLMessage serviceMsg;
44     private OntModel ontology;
45     private ClientRegister clientRegister;
46     private SuggestionRegister suggestionRegister;
47
48     protected void setup() {
49             openOntology("res/ontologia.owl");
50             clientRegister = new ClientRegister();
51             suggestionRegister = new SuggestionRegister();
52
53             addBehaviour(new ReceiveRequest());
54     }
55
56     class ReceiveRequest extends CyclicBehaviour {
57
58         @Override
59         public void action() {
60             ACLMessage msg = myAgent.receive();
61
62             if (msg != null) {
63                 Message teste = null;
64
65                 try {
66                     teste = (Message) msg.getContentObject()
                            ;
```

```
67                  } catch (UnreadableException e1) {
68                      e1.printStackTrace();
69                  }
70
71                  switch(teste.getType()) {
72                  case "CLIENT":
73                      addBehaviour(new RegisterRequests(msg));
74                      break;
75                  case "SUGGESTION":
76                      addBehaviour(new RegisterRequests(msg));
77                      break;
78                  case "SENSOR":
79                      addBehaviour(new RegisterSensorData(msg)
                            );
80                      break;
81                  case "SEARCH":
82                      addBehaviour(new ReturnSuggestion(msg));
83                  break;
84                  case "DELETE":
85                      addBehaviour(new DeleteItem(msg));
86                  break;
87                  default:
88                      System.out.println("Wrong Parameter at
                            ReceiveRequests Behaviour");
89
90              }
91          }
92              else
93                  block();
94      }
95      }
96
97      class RegisterSensorData extends OneShotBehaviour {
98
99          ACLMessage msg;
100
101         public RegisterSensorData(ACLMessage msg) {
102             this.msg = msg;
103         }
104
105         @Override
106         public void action() {
107             String type = null;
108             Message message = null;
109             try {
110                 message = (Message) msg.getContentObject();
111             } catch (UnreadableException e) {
112                 e.printStackTrace();
113             }
114
115             String data = message.getContent();
116             data = data.trim();
117
118             System.out.println("Data 2: " + data);
```

```
119
120             String[] data2 = data.replaceAll(" ","").split("
                    =");
121
122             switch(data2[0]) {
123             case "temperatura":
124                 type = "Temperatura";
125                 break;
126             case "luminosidade":
127                 type = "Luminosidade";
128                 break;
129             case "umidade":
130                 type = "Umidade";
131                 break;
132             default:
133                 return;
134             }
135
136             String update =
137             "PREFIX ontology: <" + ns + ">" +
138             "DELETE { ontology:" + type + " ontology:valor
                    " + "?p " + " }  " +
139             "WHERE  { ontology:" + type + " ontology:valor
                    " + "?p " + "}";
140
141             String update2 =
142             "PREFIX ontology: <" + ns + ">" +
143             "INSERT { ontology:" + type + " ontology:valor
                    " + "\"" + data2[1] + "\"" + " }" + "WHERE {}
                    ";
144
145             UpdateAction.parseExecute(update, ontology);
146             UpdateAction.parseExecute(update2, ontology);
147
148             ontology.write(System.out);
149
150         }
151
152     }
153
154     class DeleteItem extends OneShotBehaviour {
155
156         ACLMessage msg;
157
158         public DeleteItem(ACLMessage msg) {
159             this.msg = msg;
160         }
161
162         @Override
163         public void action() {
164             String type = null;
165             Message message = null;
166             try {
167                 message = (Message) msg.getContentObject();
```

```
168             } catch (UnreadableException e) {
169                 e.printStackTrace();
170             }
171             Map<String, String> dados = message.getData();
172             type = dados.get("nome");
173 
174             String update =
175             "PREFIX ontology: <" + ns + ">" +
176             "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
                schema#>" +
177             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
                syntax-ns#>" +
178             "DELETE WHERE { ontology:" + type + " ?p ?o } "
                +
179             "DELETE WHERE { ontology:" + type + "_Context ?p
                 ?o } ";
180 
181             UpdateAction.parseExecute(update, ontology);
182 
183             ontology.write(System.out);
184 
185         }
186 
187     }
188 
189     class ReturnSuggestion extends OneShotBehaviour {
190 
191         ACLMessage msg;
192 
193         public ReturnSuggestion(ACLMessage msg) {
194             this.msg = msg;
195         }
196 
197         @Override
198         public void action() {
199             String type = null;
200             Message message = null;
201             try {
202                 message = (Message) msg.getContentObject();
203             } catch (UnreadableException e) {
204                 e.printStackTrace();
205             }
206             Map<String, String> dados = message.getData();
207             type = dados.get("tipo");
208 
209             List<String> clients = returnClients(dados);
210 
211             ACLMessage msg = this.msg.createReply();
212             Message reply = new Message();
213             reply.setList(clients);
214 
215             Map<String, String> sugestao = returnItem(
                message, type);
216 
```

```
217             msg.setConversationId("SEARCH");
218             reply.setData(sugestao);
219             try {
220                 msg.setContentObject(reply);
221             } catch (IOException e) {
222                 e.printStackTrace();
223             }
224             send(msg);
225         }
226 
227     }
228 
229     public Map<String, String> returnItem(Message message,
        String type) {
230         // Create a new query
231         String queryString =
232         "PREFIX ontology: <" + ns + ">" +
233         "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema
            #>" +
234         "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
            syntax-ns#>" +
235         "SELECT ?item ?price ?description " +
236         "WHERE {" +
237         "       ?item       rdf:type                ontology:" +
             type + " ." +
238         "       ?item       ontology:preco          ?price ." +
239         "       ?item       ontology:descricao      ?description
             ." +
240         "       } ";
241 
242         Query query = QueryFactory.create(queryString);
243         QueryExecution qe = QueryExecutionFactory.create(
            query, ontology);
244         ResultSet results2 = qe.execSelect();
245 
246         serviceMsg = new ACLMessage(ACLMessage.INFORM);
247         serviceMsg.setConversationId(SERVICES_LIST);
248         //List<QuerySolution> solutions = ResultSetFormatter
            .toList(results);
249 
250         Map<String, String> sugestao = new HashMap<String,
            String>();
251 
252         if (results2.hasNext()) {
253             QuerySolution result2 = results2.next();
254 
255             sugestao.put("nome", message.getData().get("nome
                "));
256             sugestao.put("preco", result2.getLiteral("?price
                ").getString());
257             sugestao.put("descricao", result2.getLiteral("?
                description").getString());
258         }
259 
```

```
260         qe.close();
261 
262         return sugestao;
263     }
264 
265     public List<String> returnClients(Map<String, String>
        dados) {
266 
267         String queryString =
268             "PREFIX ontology: <" + ns + ">" +
269             "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
                schema#>" +
270             "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
                syntax-ns#>" +
271             "SELECT ?idCliente " +
272             "WHERE {" +
273             "       ?cliente      rdf:type
                ontology:Cliente ." +
274             "       ?cliente      ontology:id_usuario    ?
                idCliente .";
275 
276 
277             queryString += " ?cliente  ontology:emprego \""
                + dados.get("emprego") + "\" .";
278             queryString += " ?cliente  ontology:sexo \"" +
                dados.get("sexo") + "\" .";
279 
280             queryString += "?cliente  ontology:idade   ?
                idade .";
281 
282             if (dados.containsKey("vegetariano")) {
283                 queryString += "?cliente    ontology:
                    pertence     \"Vegetariano\" .";
284                 }
285             if (dados.containsKey("hipertenso")) {
286                 queryString += "?cliente    ontology:
                    pertence     \"Hipertenso\" .";
287                 }
288             if (dados.containsKey("diabetico")) {
289                 queryString += "?cliente    ontology:
                    pertence     \"Diab tico\" .";
290                 }
291 
292             queryString += "FILTER ( ?idade >= " + dados.get
                ("idadeMin") + ")";
293             queryString += "FILTER ( ?idade <= " + dados.get
                ("idadeMax") +" ) }";
294 
295             Query query = QueryFactory.create(queryString);
296 
297             // Execute the query and obtain results
298             QueryExecution qe = QueryExecutionFactory.create
                (query, ontology);
299             ResultSet results = qe.execSelect();
```

```
300
301             List<QuerySolution> solutions =
                    ResultSetFormatter.toList(results);
302             List<String> clients = new ArrayList<String>();
303
304             for(QuerySolution r : solutions)
305                 clients.add(r.getLiteral("?idCliente").
                        getString());
306
307             qe.close();
308
309             return clients;
310     }
311
312     /**
313      * Recebe as requisi  es dos agentes de cadastro e de
            leitura
314      * dos sensores. Insere as informa  es recebidas na
            ontologia.
315      */
316     class RegisterRequests extends OneShotBehaviour {
317
318         private ACLMessage msg;
319
320         public RegisterRequests(ACLMessage msg) {
321             this.msg = msg;
322         }
323
324         @Override
325         public void action() {
326             Message data = null;
327             try {
328                 data = (Message) msg.getContentObject();
329             } catch (UnreadableException e) {
330                 e.printStackTrace();
331             }
332
333             switch(data.getType()) {
334             case "CLIENT":
335                 registerClient(data.getData());
336                 break;
337             case "SUGGESTION":
338                 registerSuggestion(data.getData());
339                 break;
340             default:
341                 System.out.println("Wrong Parameter at
                        ListenRequests Behaviour");
342         }
343
344         }
345     }
346
347     private void registerClient(Map<String, String> dados) {
348         clientRegister.registerClient(dados, ontology);
```

```
349         ontology.write(System.out);
350     }
351 
352     public void registerSuggestion(Map<String, String> dados
        ) {
353 
354         suggestionRegister.registerSuggestion(dados,
            ontology);
355 
356         /* enviar mensagem para o agente de inferencia para
            agendar
357          * a notificacao
358          */
359         sendMessage(dados);
360 
361         ontology.write(System.out);
362     }
363 
364     private void registerSensorData(Map<String, String>
        dados) {
365 
366     }
367 
368     private void openOntology(String path) {
369         InputStream in = null;
370         try {
371             in = new FileInputStream(new File(path));
372         } catch (FileNotFoundException e) {
373             e.printStackTrace();
374         }
375 
376         ontology = ModelFactory.createOntologyModel(
            OntModelSpec.OWL_MEM );
377         ontology.read(in,"RDF/XML"); // null base URI, since
             model URIs are absolute
378 
379 //      ontology.write(System.out);
380 
381         try {
382             in.close();
383         } catch (IOException e) {
384             e.printStackTrace();
385         }
386     }
387 
388     public void sendMessage(Map<String, String> dados) {
389         ACLMessage spokenMsg = new ACLMessage(ACLMessage.
            INFORM);
390         ACLMessage deleteMsg = new ACLMessage(ACLMessage.
            INFORM);
391 
392         spokenMsg.setConversationId("agendar_notificacao");
393         deleteMsg.setConversationId("agendar_exclusao");
394 
```

```
395         spokenMsg.clearAllReceiver();
396         deleteMsg.clearAllReceiver();
397 
398         Message msg = null;
399         switch(dados.get("tipo")) {
400         case "Servi o":
401             msg = new Message("services", dados);
402             spokenMsg.addReceiver(new AID("servi os", AID.
                ISLOCALNAME));
403             break;
404 
405         case "Atividade":
406             msg = new Message("atividades", dados);
407             spokenMsg.addReceiver(new AID("atividades", AID.
                ISLOCALNAME));
408             break;
409 
410         case "Card pio":
411             msg = new Message("card pio", dados);
412             spokenMsg.addReceiver(new AID("card pio", AID.
                ISLOCALNAME));
413             break;
414         default:
415             System.out.println("Op   o de item inv lida.")
                ;
416         }
417 
418         try {
419             spokenMsg.setContentObject(msg);
420             deleteMsg.setContentObject(msg);
421         } catch (IOException e) {
422             e.printStackTrace();
423         }
424 
425         send(spokenMsg);
426 
427         deleteMsg.addReceiver(new AID("limpador", AID.
            ISLOCALNAME));
428 
429         send(deleteMsg);
430 
431     }
432 }
```

1.3 Agente de Cadastro

```
1 package agents;
2 import gui.Cadastre;
3 
4 import java.io.IOException;
5 import java.util.Map;
6 import utils.Message;
7 
8 import jade.core.AID;
```

```
9  import jade.core.Agent;
10 import jade.core.behaviours.CyclicBehaviour;
11 import jade.core.behaviours.OneShotBehaviour;
12 import jade.lang.acl.ACLMessage;
13 import jade.lang.acl.UnreadableException;
14
15 public class CadastreAgent extends Agent {
16
17     private static final long serialVersionUID = 1L;
18     private ACLMessage spokenMsg;
19     private Cadastre cadastroUI;
20     private static final String CLIENT_REGISTER = "
           client_register";
21
22     protected void setup() {
23         cadastroUI= new Cadastre(this);
24         cadastroUI.setLookAndFeel();
25         spokenMsg = new ACLMessage(ACLMessage.INFORM);
26         spokenMsg.setConversationId("ontology");
27         addBehaviour(new ListenRequests());
28     }
29
30     class ListenRequests extends CyclicBehaviour {
31
32         @Override
33         public void action() {
34             ACLMessage msg = myAgent.receive();
35
36             if (msg != null) {
37                 Map<String, String> teste = null;
38
39                 try {
40                     teste = (Map<String, String>) msg.
                           getContentObject();
41                 } catch (UnreadableException e1) {
42                     e1.printStackTrace();
43                 }
44
45                 if (teste.get("cliente").equals("tarcisio"))
                       {
46                     System.out.println("\nCOMPRA REALIZADA!"
                           );
47                     System.out.println("Cliente: " + teste.
                           get("cliente"));
48                     System.out.print("Item comprado: " +
                           teste.get("nome").split(":")[1]);
49                     System.out.println(teste.get("preco"));
50                 }
51
52         }
53             else
54                 block();
55     }
56         }
```

```
57
58
59      /**
60       * Recebe os dados do cadastro e comunica o Agente de
             Ontologia
61       * para que ele insira os dados corretamente na
             ontologia
62       */
63      class RegisterData extends OneShotBehaviour {
64
65          private Map<String, String> data;
66          private String dataType;
67
68          public RegisterData(Map<String, String> data, String
                 dataType) {
69              this.data = data;
70              this.dataType = dataType;
71          }
72
73          @Override
74          public void action() {
75              spokenMsg.clearAllReceiver();
76              Message msg = null;
77
78              switch(this.dataType) {
79                  case "CLIENT":
80                      msg = new Message("CLIENT", this.data);
81                      break;
82                  case "SUGGESTION":
83                      msg = new Message("SUGGESTION", this.
                             data);
84                      break;
85                  default:
86                      System.out.println("Wrong Parameter at
                             RegisterData Behaviour");
87              }
88
89              try {
90                  spokenMsg.setContentObject(msg);
91              } catch (IOException e) {
92                  e.printStackTrace();
93              }
94              spokenMsg.addReceiver(new AID("ontologia", AID.
                     ISLOCALNAME));
95
96              send(spokenMsg);
97          }
98      }
99
100     /**
101      * Chamado pela interface quando ocorre um cadastro de
             cliente
102      * @param dados
103      */
```

```
104     public void registerClient(Map<String, String> dados) {
105         addBehaviour(new RegisterData(dados, "CLIENT"));
106     }
107 
108     public void registerSuggestion(Map<String, String> dados
        ) {
109         addBehaviour(new RegisterData(dados, "SUGGESTION"));
110 
111     }
112 
113 }
```

1.4 Agente Limpador

```
1  package agents;
2 
3  import java.io.IOException;
4  import java.text.DateFormat;
5  import java.text.ParseException;
6  import java.text.SimpleDateFormat;
7  import java.util.Calendar;
8  import java.util.Date;
9  import java.util.Map;
10 import utils.Message;
11 import jade.core.AID;
12 import jade.core.Agent;
13 import jade.core.behaviours.CyclicBehaviour;
14 import jade.core.behaviours.OneShotBehaviour;
15 import jade.core.behaviours.WakerBehaviour;
16 import jade.lang.acl.ACLMessage;
17 import jade.lang.acl.MessageTemplate;
18 import jade.lang.acl.UnreadableException;
19 
20 public class CleanerAgent extends Agent {
21 
22     private ACLMessage ontologyMsg;
23 
24     protected void setup() {
25         ontologyMsg = new ACLMessage(ACLMessage.INFORM);
26 
27         addBehaviour(new ListenRequests());
28 
29         }
30 
31     class ListenRequests extends CyclicBehaviour {
32 
33         private MessageTemplate template = MessageTemplate
34                 .MatchConversationId("agendar_exclusao");
35 
36         @Override
37         public void action() {
38             ACLMessage msg = myAgent.receive(template);
39 
40             if (msg != null) {
```

```
                Message content = null;
                try {
                    content = (Message) msg.getContentObject
                        ();
                } catch (UnreadableException e) {
                    e.printStackTrace();
                }

                deleteSchedule(content.getData());

            }
             else {
                block(); //important
            }
        }


    }

    public void deleteSchedule(final Map<String, String>
        content) {

        String dataFim = content.get("dataFim");
        DateFormat formatador = new SimpleDateFormat("dd/MM/
            yyyy");

        Date endDate = null;

        try {
            endDate = formatador.parse(dataFim);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        Calendar cal = Calendar.getInstance();
        cal.setTime(endDate);
        cal.add(Calendar.DATE, -1); // add 1 day

        endDate = cal.getTime();

        addBehaviour(new WakerBehaviour(this, endDate) {
            protected void handleElapsedTimeout() {
                addBehaviour(new SendRequests(content));
            }
            } );
        }

    class SendRequests extends OneShotBehaviour {

        Message context;

        public SendRequests(Map<String, String> content) {
            context = new Message("DELETE", content);
        }
```

```
92
93          @Override
94          public void action() {
95              // Manda mensagem perguntando pelos clientes
96              ontologyMsg.clearAllReceiver();
97              try {
98                  ontologyMsg.setContentObject((Message)
                        context);
99              } catch (IOException e) {
100                 e.printStackTrace();
101             }
102             ontologyMsg.addReceiver(new AID("ontologia", AID
                    .ISLOCALNAME));
103             send(ontologyMsg);
104
105         }
106         }
107
108     }
```

1.5 Agente de Inferência

```
1  package agents;
2
3  import java.io.IOException;
4  import java.io.Serializable;
5  import java.text.DateFormat;
6  import java.text.ParseException;
7  import java.text.SimpleDateFormat;
8  import java.util.Calendar;
9  import java.util.Date;
10 import java.util.List;
11 import java.util.Map;
12 import utils.Message;
13 import jade.core.AID;
14 import jade.core.Agent;
15 import jade.core.behaviours.CyclicBehaviour;
16 import jade.core.behaviours.OneShotBehaviour;
17 import jade.core.behaviours.WakerBehaviour;
18 import jade.lang.acl.ACLMessage;
19 import jade.lang.acl.MessageTemplate;
20 import jade.lang.acl.UnreadableException;
21
22 public class InferenceAgent extends Agent {
23
24     private ACLMessage contextMsg;
25     private ACLMessage cellMsg;
26     private static final String SEARCH_CLIENT = "
           search_clients";
27
28     protected void setup() {
29         cellMsg = new ACLMessage(ACLMessage.INFORM);
30
31         contextMsg = new ACLMessage(ACLMessage.INFORM);
```

```
32          contextMsg.setConversationId(SEARCH_CLIENT);
33
34          addBehaviour(new ListenRequests());
35
36          }
37
38      /**
39       * Executa um cyclic behaviour que faz as inferencias
             necess rias
40       * notificando os agentes de celular
41       */
42      class SendRequests extends OneShotBehaviour {
43
44          Message context;
45
46          public SendRequests(Map<String, String> content) {
47              context = new Message("SEARCH", content);
48          }
49
50          @Override
51          public void action() {
52              // Manda mensagem perguntando pelos clientes
53              contextMsg.clearAllReceiver();
54              try {
55                  contextMsg.setContentObject((Message)
                        context);
56              } catch (IOException e) {
57                  e.printStackTrace();
58              }
59              System.out.println("AQUI!");
60              contextMsg.addReceiver(new AID("ontologia", AID.
                    ISLOCALNAME));
61              System.out.println("AQUI! 2");
62              send(contextMsg);
63
64              addBehaviour(new SendSuggestion());
65
66          }
67          }
68
69      class SendSuggestion extends CyclicBehaviour {
70
71          @Override
72          public void action() {
73              MessageTemplate template = MessageTemplate.
                    MatchConversationId("SEARCH");
74              ACLMessage message = myAgent.receive(template);
75
76              Message content = null;
77
78              if (message != null) {
79                  try {
80                      content = (Message) message.
                            getContentObject();
```

```
81                  } catch (UnreadableException e) {
82                      e.printStackTrace();
83                  }
84                  System.out.println("MAP: " + content.getData
                        ());
85                  System.out.println("LIST: " + content.
                        getList());
86                  sendSuggestion(content.getData(), content.
                        getList());
87                  }
88               else {
89                  block(); //important
90              }
91          }
92
93      }
94
95      class ListenServiceAnswer extends CyclicBehaviour {
96
97          Message content;
98          MessageTemplate template = MessageTemplate.
                MatchConversationId
99                  ("SEARCH_SERVICES_ANSWER");
100
101         public ListenServiceAnswer(Message content) {
102             this.content = content;
103         }
104
105         @Override
106         public void action() {
107             ACLMessage message2 = myAgent.blockingReceive(
                    template);
108
109             Message resposta = null;
110             try {
111                 resposta = (Message) message2.
                        getContentObject();
112             } catch (UnreadableException e) {
113                 e.printStackTrace();
114             }
115             System.out.println("DATA: " + resposta.getData()
                    );
116             System.out.println("LIST: " + content.getList())
                    ;
117             sendSuggestion(resposta.getData(), content.
                    getList());
118
119         }
120
121     }
122
123     class ListenRequests extends CyclicBehaviour {
124
125         private MessageTemplate template = MessageTemplate
```

```
126                 .MatchConversationId("agendar_notificacao");
127
128         @Override
129         public void action() {
130             ACLMessage msg = myAgent.receive(template);
131
132             if (msg != null) {
133                 Message content = null;
134                 try {
135                     content = (Message) msg.getContentObject
                            ();
136                 } catch (UnreadableException e) {
137                     e.printStackTrace();
138                 }
139
140                 notificationSchedule(content.getData());
141
142             }
143              else {
144                 block(); //important
145             }
146
147         }
148     }
149
150     public void sendSuggestion(Map<String, String> result,
            List<String> clients) {
151         System.out.println("Services: " + result.get("
                servico"));
152
153         cellMsg.clearAllReceiver();
154         cellMsg.setConversationId("send_sugestion");
155
156         try {
157             cellMsg.setContentObject((Serializable) result);
158         } catch (IOException e) {
159             e.printStackTrace();
160         }
161
162         //TODO: trocar pelo metodo broadcast
163         for (String client: clients) {
164             cellMsg.addReceiver(new AID(client, AID.
                    ISLOCALNAME));
165             System.out.println("Enviando a mensagem para: "
                    + client);
166         }
167         send(cellMsg);
168     }
169
170     /**
171      * Schedule the notification of the service using a
             WakerBehaviour
172      * @param content
173      */
```

```
174     public void notificationSchedule(final Map<String,
            String> content) {
175
176         String dataInicio = content.get("dataInicio");
177         String dataFim = content.get("dataFim");
178         String[] horario = content.get("hora").split(":");
179         String hora = horario[0];
180         String minuto = horario[1];
181         DateFormat formatador = new SimpleDateFormat("dd/MM/
                yyyy");
182
183         Date startDate = null;
184         Date endDate = null;
185
186         try {
187             startDate = formatador.parse(dataInicio);
188             endDate = formatador.parse(dataFim);
189         } catch (ParseException e) {
190             e.printStackTrace();
191         }
192
193         startDate.setHours(Integer.parseInt(hora));
194         startDate.setMinutes(Integer.parseInt(minuto));
195
196         while (startDate.before(endDate)) {
197             addBehaviour(new WakerBehaviour(this, startDate)
                    {
198                 protected void handleElapsedTimeout() {
199                     switch(content.get("tipo")) {
200                     case("Servi o"): content.put("tipo", "
                            Servi o"); break;
201                     case("Atividade"): content.put("tipo", "
                            Atividade"); break;
202                     case("Item de Card pio"): content.put("
                            tipo", "Card pio"); break;
203                     }
204                     addBehaviour(new SendRequests(content));
205                 }
206             } );
207
208             Calendar cal = Calendar.getInstance();
209             cal.setTime(startDate);
210             cal.add(Calendar.DATE, 1); // add 1 day
211
212             startDate = cal.getTime();
213         }
214     }
215     }
```

1.6 Agente de Sensores

```
1  package agents;
2
3 import java.io.IOException;
```

```
4  import java.io.InputStream;
5  import java.io.OutputStream;
6  import java.util.Enumeration;
7
8  import utils.Message;
9
10 import gnu.io.CommPortIdentifier;
11 import gnu.io.SerialPort;
12 import gnu.io.SerialPortEvent;
13 import gnu.io.SerialPortEventListener;
14 import jade.core.AID;
15 import jade.core.Agent;
16 import jade.core.behaviours.OneShotBehaviour;
17 import jade.lang.acl.ACLMessage;
18
19 public class SensorAgent extends Agent implements
       SerialPortEventListener {
20
21     SerialPort serialPort;
22     /** The port we're normally going to use. */
23     private static final String PORT_NAMES[] = {
24         "/dev/tty.usbserial-A9007UX1", // Mac OS X
25         "/dev/ttyACM1", // Linux
26         "COM3", // Windows
27         };
28
29     private String data = "";
30
31     /** Buffered input stream from the port */
32     private InputStream input;
33     /** The output stream to the port */
34     private OutputStream output;
35     /** Milliseconds to block while waiting for port open */
36     private static final int TIME_OUT = 2000;
37     /** Default bits per second for COM port. */
38     private static final int DATA_RATE = 19200;
39
40     protected void setup() {
41         CommPortIdentifier portId = null;
42         Enumeration portEnum = CommPortIdentifier.
               getPortIdentifiers();
43
44         // iterate through, looking for the port
45         while (portEnum.hasMoreElements()) {
46             CommPortIdentifier currPortId = (
                   CommPortIdentifier) portEnum.nextElement();
47             for (String portName : PORT_NAMES) {
48                 if (currPortId.getName().equals(portName)) {
49                     portId = currPortId;
50                     break;
51                 }
52             }
53         }
54
```

```
55          if (portId == null) {
56              System.out.println("Could not find COM port.");
57              return;
58          }
59
60          try {
61              // open serial port, and use class name for the
                    appName.
62              serialPort = (SerialPort) portId.open(this.
                    getClass().getName(),
63                      TIME_OUT);
64
65              // set port parameters
66              serialPort.setSerialPortParams(DATA_RATE,
67                      SerialPort.DATABITS_8,
68                      SerialPort.STOPBITS_1,
69                      SerialPort.PARITY_NONE);
70
71              // open the streams
72              input = serialPort.getInputStream();
73              output = serialPort.getOutputStream();
74
75              // add event listeners
76              serialPort.addEventListener(this);
77              serialPort.notifyOnDataAvailable(true);
78          } catch (Exception e) {
79              System.err.println(e.toString());
80          }
81
82      }
83
84
85      class SendData extends OneShotBehaviour {
86
87          private ACLMessage spokenMsg;
88          private String data;
89
90          public SendData(String data) {
91              this.data = data;
92              this.spokenMsg = new ACLMessage(ACLMessage.
                    INFORM);
93          }
94
95          @Override
96          public void action() {
97              spokenMsg.clearAllReceiver();
98              Message msg = null;
99              msg = new Message("SENSOR", this.data);
100
101             try {
102                 spokenMsg.setContentObject(msg);
103             } catch (IOException e) {
104                 e.printStackTrace();
105             }
```

```
106
107             spokenMsg.addReceiver(new AID("ontologia", AID.
                    ISLOCALNAME));
108
109             send(spokenMsg);
110
111         }
112     }
113
114     /**
115      * This should be called when you stop using the port.
116      * This will prevent port locking on platforms like
             Linux.
117      */
118     public synchronized void close() {
119         if (serialPort != null) {
120             serialPort.removeEventListener();
121             serialPort.close();
122         }
123     }
124
125     /**
126      * Handle an event on the serial port. Read the data and
             print it.
127      */
128     public synchronized void serialEvent(SerialPortEvent
            oEvent) {
129         if (oEvent.getEventType() == SerialPortEvent.
                DATA_AVAILABLE) {
130             try {
131                 char in = (char) input.read();
132
133                 if(in != '\r'){
134                     data += in;
135                 }else{
136                     addBehaviour(new SendData(data));
137                     data = "";
138                 }
139
140             } catch (Exception e) {
141                 System.err.println(e.toString());
142             }
143         }
144         // Ignore all the other eventTypes, but you should
                consider the other ones.
145     }
146
147
148 }
```

1.7 Classes Utilitárias

1.7.1 Registrador de Clientes

```
1 package utils;
```

```
2
3  import java.util.Map;
4  import com.hp.hpl.jena.ontology.OntModel;
5  import com.hp.hpl.jena.rdf.model.Literal;
6  import com.hp.hpl.jena.rdf.model.Resource;
7  import com.hp.hpl.jena.rdf.model.ResourceFactory;
8  import com.hp.hpl.jena.vocabulary.RDF;
9
10 public class ClientRegister {
11
12     private final String ns = "http://www.semanticweb.org/
           ontologies/" +
13                 "2012/9/Ontology1350055633402.owl#";
14
15     public void registerClient(Map<String, String> dados,
           OntModel ontology) {
16         Resource client = ResourceFactory.createResource(ns
               + "Cliente");
17         Resource reserve = ResourceFactory.createResource(ns
                + "Reserva");
18
19         Literal age = ResourceFactory.createTypedLiteral(
               Integer.parseInt(dados.get("idade")));
20         Literal sex = ResourceFactory.createTypedLiteral(
               dados.get("sexo"));
21         Literal job = ResourceFactory.createTypedLiteral(
               dados.get("emprego"));
22         Literal id = ResourceFactory.createTypedLiteral(
               dados.get("usuarioCliente"));
23         Literal dataDeInicio = ResourceFactory.
               createTypedLiteral(dados.get("inicioReserva"));
24         Literal dataDeFim = ResourceFactory.
               createTypedLiteral(dados.get("fimReserva"));
25
26         Resource ClientResource = ontology.createResource(ns
               +dados.get("nome"));
27         Resource ReserveResource = ontology.createResource(
               ns+"Reserva_"+dados.get("nome"));
28
29         setAlimentation(dados, ClientResource, ontology);
30         setClientProperties(client, age, sex, job, id,
               ClientResource, ontology);
31         setReservePropeties(reserve, dataDeInicio, dataDeFim
               , ReserveResource, ontology);
32     }
33
34     private void setReservePropeties(Resource reserve,
           Literal dataDeInicio,
35             Literal dataDeFim, Resource ReserveResource,
                   OntModel ontology) {
36         ReserveResource.addProperty(RDF.type, reserve);
37         ReserveResource.addProperty(ontology.getProperty(ns+
               "dataDeInicio"), dataDeInicio);
38         ReserveResource.addProperty(ontology.getProperty(ns+
```

```
            "dataDeFim"), dataDeFim);
39      }
40
41      private void setClientProperties(Resource client,
            Literal age, Literal sex,
42              Literal job, Literal id, Resource ClientResource
                    , OntModel ontology) {
43          ClientResource.addProperty(RDF.type, client);
44          ClientResource.addProperty(ontology.getProperty(ns+"
                idade"), age);
45          ClientResource.addProperty(ontology.getProperty(ns+"
                sexo"), sex);
46          ClientResource.addProperty(ontology.getProperty(ns+"
                emprego"), job);
47          ClientResource.addProperty(ontology.getProperty(ns+"
                id_usuario"), id);
48      }
49
50      private void setAlimentation(Map<String, String> dados,
51              Resource ClientResource, OntModel ontology) {
52          if (dados.containsKey("vegetariano")) {
53              String vg = dados.get("vegetariano");
54              ClientResource.addProperty(ontology.getProperty(
                    ns+"pertence"), vg );
55          }
56          if (dados.containsKey("diabetico")) {
57              String db = dados.get("diabetico");
58              ClientResource.addProperty(ontology.getProperty(
                    ns+"pertence"), db );
59          }
60          if (dados.containsKey("hipertenso")) {
61              String hp = dados.get("hipertenso");
62              ClientResource.addProperty(ontology.getProperty(
                    ns+"pertence"), hp );
63          }
64      }
65
66  }
```

1.7.2 Registrador de Sugestões

```
1   package utils;
2
3   import java.util.Map;
4
5   import com.hp.hpl.jena.ontology.OntModel;
6   import com.hp.hpl.jena.rdf.model.Literal;
7   import com.hp.hpl.jena.rdf.model.Resource;
8   import com.hp.hpl.jena.rdf.model.ResourceFactory;
9   import com.hp.hpl.jena.vocabulary.RDF;
10
11  public class SuggestionRegister {
12
13      private final String ns = "http://www.semanticweb.org/
            ontologies/" +
```

```
14              "2012/9/Ontology1350055633402.owl#";
15
16      public void registerSuggestion(Map<String, String> dados
            , OntModel ontology) {
17          switch(dados.get("tipo")) {
18          case("Servi o"): dados.put("tipo", "Servi o");
                break;
19          case("Atividade"): dados.put("tipo", "Atividade");
                break;
20          case("Item de Card pio"): dados.put("tipo", "
                Card pio"); break;
21          }
22          Resource tipo = ResourceFactory.createResource(ns +
                dados.get("tipo"));
23
24          Literal des = ResourceFactory.createTypedLiteral(
                dados.get("descricao"));
25          Literal pre = ResourceFactory.createTypedLiteral(
                dados.get("preco"));
26          Literal inicio = ResourceFactory.createTypedLiteral(
                dados.get("dataInicio"));
27          Literal fim = ResourceFactory.createTypedLiteral(
                dados.get("dataFim"));
28
29          Resource sugestao = ontology.createResource(ns+dados
                .get("nome"));
30
31          sugestao.addProperty(RDF.type, tipo);
32          sugestao.addProperty(ontology.getProperty(ns+"
                descricao"), des);
33          sugestao.addProperty(ontology.getProperty(ns+"preco"
                ), pre);
34          sugestao.addProperty(ontology.getProperty(ns+"
                dataInicio"), inicio);
35          sugestao.addProperty(ontology.getProperty(ns+"
                dataFim"), fim);
36
37          Resource context = ontology.createResource(ns+dados.
                get("nome")+"_Context");
38          context.addProperty(RDF.type, ontology.getProperty(
                ns+"Contexto"));
39          context.addProperty(ontology.getProperty(ns+"trigger
                "), sugestao);
40
41          registerContext(dados, context, ontology);
42
43          ontology.write(System.out);
44      }
45
46      private void registerContext(Map<String, String> dados,
            Resource contexto, OntModel ontology) {
47          registerWeatherContext(dados, contexto, ontology);
48          registerClientContext(dados, contexto, ontology);
49          registerTimeContext(dados, contexto, ontology);
```

```
50      }
51
52      private void registerTimeContext(Map<String, String>
            dados, Resource contexto, OntModel ontology) {
53          Literal inicio = ResourceFactory.createTypedLiteral(
                dados.get("dataInicio"));
54          Literal fim = ResourceFactory.createTypedLiteral(
                dados.get("dataFim"));
55          Literal hora = ResourceFactory.createTypedLiteral(
                dados.get("hora"));
56
57          contexto.addProperty(ontology.getProperty(ns+"
                dataDeInicio"), inicio);
58          contexto.addProperty(ontology.getProperty(ns+"
                dataDeFim"), fim);
59          contexto.addProperty(ontology.getProperty(ns+"hora")
                , hora);
60
61      }
62
63      private void registerClientContext(Map<String, String>
            dados, Resource contexto, OntModel ontology) {
64          Literal emprego = ResourceFactory.createTypedLiteral
                (dados.get("emprego"));
65          Literal sexo = ResourceFactory.createTypedLiteral(
                dados.get("sexo"));
66          Literal idadeMin = ResourceFactory.
                createTypedLiteral(dados.get("idadeMin"));
67          Literal idadeMax = ResourceFactory.
                createTypedLiteral(dados.get("idadeMax"));
68
69          contexto.addProperty(ontology.getProperty(ns+"
                emprego"), emprego);
70          contexto.addProperty(ontology.getProperty(ns+"sexo")
                , sexo);
71          contexto.addProperty(ontology.getProperty(ns+"idade"
                ), idadeMin);
72          contexto.addProperty(ontology.getProperty(ns+"idade"
                ), idadeMax);
73          if (dados.containsKey("vegetariano")) {
74              Literal vegetariano = ResourceFactory.
                    createTypedLiteral(dados.get("vegetariano"));
75              contexto.addProperty(ontology.getProperty(ns+"
                    vegetariano"), vegetariano);
76          }
77          if (dados.containsKey("hipertenso")) {
78              Literal hipertenso = ResourceFactory.
                    createTypedLiteral(dados.get("hipertenso"));
79              contexto.addProperty(ontology.getProperty(ns+"
                    diab tico"), hipertenso);
80          }
81          if (dados.containsKey("diabetico")) {
82              Literal diabetico = ResourceFactory.
                    createTypedLiteral(dados.get("diabetico"));
```

```
            contexto.addProperty(ontology.getProperty(ns+"
                hipertenso"), diabetico);
        }

    }

    private void registerWeatherContext(Map<String, String>
        dados, Resource contexto, OntModel ontology) {
        Literal tempMin = ResourceFactory.createTypedLiteral
            (dados.get("tempMin"));
        Literal tempMax = ResourceFactory.createTypedLiteral
            (dados.get("tempMax"));
        contexto.addProperty(ontology.getProperty(ns+"
            tempMin"), tempMin);
        contexto.addProperty(ontology.getProperty(ns+"
            tempMax"), tempMax);

        if (dados.containsKey("clima")) {
            Literal clima = ResourceFactory.
                createTypedLiteral(dados.get("clima"));
            contexto.addProperty(ontology.getProperty(ns+"
                clima"), clima);
        }
    }

}
```

1.7.3 Mensagem

```
 package utils;

import java.io.Serializable;
import java.util.List;
import java.util.Map;

public class Message implements Serializable {

    private static final long serialVersionUID = 1L;

    private String type;
    private Map<String, String> data;
    private List<String> list;
    private String content;

    public Message() {
    }

    public Message(String type, Map<String, String> data) {
        this.type = type;
        this.data = data;
    }

    public Message(String type, String content) {
        this.type = type;
        this.content = content;
```

```
    }

    public Message(List<String> list) {
        this.list = list;
    }

    public String getType() {
        return type;
    }

    public String getContent() {
        return content;
    }

    public Map<String, String> getData() {
        return data;
    }

    public List<String> getList() {
        return list;
    }

    public void setData(Map<String, String> data) {
        this.data = data;
    }

    public void setList(List<String> list) {
        this.list = list;
    }
}
```

2 Código do Projeto (Cliente Android)

2.1 Interface do Agente Móvel

```
 package client.agent;

import java.util.Map;

public interface CellAgentInterface {
    public void handleSpoken(Map<String, String> s);
}
```

2.2 Agente Móvel

```
 package client.agent;

import java.io.IOException;
import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;

import jade.core.AID;
```

```
10 import jade.core.Agent;
11 import jade.core.behaviours.CyclicBehaviour;
12 import jade.core.behaviours.OneShotBehaviour;
13 import jade.lang.acl.ACLMessage;
14 import jade.lang.acl.MessageTemplate;
15 import jade.lang.acl.UnreadableException;
16 import jade.util.Logger;
17 import android.content.Intent;
18 import android.content.Context;
19
20 public class CellPhoneAgent extends Agent implements
       CellAgentInterface {
21     private static final long serialVersionUID =
           1594371294421614291L;
22
23     private Logger logger = Logger.getJADELogger(this.
           getClass().getName());
24
25     private static final String CHAT_ID = "__chat__";
26     private static final String BUY = "__buy__";
27
28     private ACLMessage spokenMsg;
29     private ACLMessage buyMsg;
30     private Context context;
31
32     protected void setup() {
33         Object[] args = getArguments();
34         if (args != null && args.length > 0) {
35             if (args[0] instanceof Context) {
36                 context = (Context) args[0];
37             }
38         }
39
40         // Add initial behaviours
41         addBehaviour(new ReceiveSuggestions(this));
42
43         // Initialize the message used to convey spoken
               sentences
44         spokenMsg = new ACLMessage(ACLMessage.INFORM);
45         spokenMsg.setConversationId(CHAT_ID);
46
47         buyMsg = new ACLMessage(ACLMessage.INFORM);
48         buyMsg.setConversationId(BUY);
49
50         // Activate the GUI
51         registerO2AInterface(CellAgentInterface.class, this)
               ;
52
53         Intent broadcast = new Intent();
54         broadcast.setAction("jade.demo.hotel.SHOW_TABS");
55         logger.log(Level.INFO, "Sending broadcast " +
               broadcast.getAction());
56         context.sendBroadcast(broadcast);
57     }
```

```
58
59
60      class ReceiveSuggestions extends CyclicBehaviour {
61
62          private MessageTemplate template = MessageTemplate
63                  .MatchConversationId("send_sugestion");
64
65          ReceiveSuggestions(Agent a) {
66              super(a);
67          }
68
69          @Override
70          public void action() {
71              ACLMessage msg = myAgent.receive(template);
72
73              if (msg != null) {
74                  try {
75                      String sender = msg.getSender().
                            getLocalName();
76                      notifySpoken(sender, msg.
                            getContentObject());
77                  } catch (UnreadableException e) {
78                      e.printStackTrace();
79                  }
80              } else {
81                  block(); //important
82              }
83          }
84
85      }
86
87      private class BuySpeaker extends OneShotBehaviour {
88          private static final long serialVersionUID =
                -1426033904935339194L;
89          private Map<String, String> sentence;
90
91          private BuySpeaker(Agent a, Map<String, String> s) {
92              super(a);
93              sentence = s;
94          }
95
96          public void action() {
97              buyMsg.clearAllReceiver();
98              buyMsg.addReceiver(new AID("cadastro", AID.
                    ISLOCALNAME));
99              try {
100                 sentence.put("cliente", myAgent.getLocalName
                        ());
101                 buyMsg.setContentObject((Serializable)
                        sentence);
102             } catch (IOException e) {
103                 e.printStackTrace();
104             }
105             /*notifySpoken(myAgent.getLocalName(), sentence)
```

```
                ;*/
106             send(buyMsg);
107         }
108     } // END of inner class ChatSpeaker
109
110     private void notifySpoken(String sender, Serializable
            msg) {
111         Intent broadcast = new Intent();
112         HashMap<String, String> data = (HashMap<String,
                String>) msg;
113
114         if (sender.equals("servi os")) {
115             broadcast.setAction("jade.demo.hotel.SERVICES");
116         } else if (sender.equals("atividades")) {
117             broadcast.setAction("jade.demo.hotel.ACTIVITIES"
                    );
118         } else if (sender.equals("card pio")) {
119             broadcast.setAction("jade.demo.hotel.MENU");
120         }
121
122         broadcast.putExtra("nome", data.get("nome"));
123         broadcast.putExtra("preco", data.get("preco"));
124         broadcast.putExtra("descricao", data.get("descricao"
                ));
125         broadcast.putExtra("agente", getLocalName());
126
127
128         logger.log(Level.INFO, "Sending broadcast " +
                broadcast.getAction());
129         context.sendBroadcast(broadcast);
130     }
131
132     public void handleSpoken(Map<String, String> s) {
133         //addBehaviour(new ChatSpeaker(this, s));
134         logger.log(Level.INFO, "Mandando mensagem! ");
135
136         addBehaviour(new BuySpeaker(this, s));
137
138     }
139
140     protected void takeDown() {
141
142     }
143
144 }
```

2.3 Interface de Atividade

```
1  package client.gui;
2
3 import jade.util.Logger;
4 import java.util.ArrayList;
5 import java.util.HashMap;
6 import java.util.Map;
```

```
7  import java.util.logging.Level;
8  import android.os.Bundle;
9  import android.app.ListActivity;
10 import android.content.BroadcastReceiver;
11 import android.content.Context;
12 import android.content.Intent;
13 import android.content.IntentFilter;
14 import android.view.Menu;
15 import android.view.MenuInflater;
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.widget.AdapterView;
19 import android.widget.ArrayAdapter;
20 import android.widget.ListView;
21 import android.widget.TextView;
22 import android.widget.AdapterView.OnItemClickListener;
23
24 public class ActivityActivity extends ListActivity {
25
26     private Logger logger = Logger.getJADELogger(this.
        getClass().getName());
27     private MyReceiver myReceiver;
28
29     static final int SETTINGS_REQUEST = 1;
30
31     //LIST OF ARRAY STRINGS WHICH WILL SERVE AS LIST ITEMS
32     ArrayList<String> listItems=new ArrayList<String>();
33
34     //DEFINING STRING ADAPTER WHICH WILL HANDLE DATA OF
        LISTVIEW
35     ArrayAdapter<String> adapter;
36
37     private Map<String, String> nameDescription;
38     private Map<String, String> namePrice;
39
40     @Override
41     public void onCreate(Bundle savedInstanceState) {
42         super.onCreate(savedInstanceState);
43
44         myReceiver = new MyReceiver();
45
46         nameDescription = new HashMap<String, String>();
47         namePrice = new HashMap<String, String>();
48
49         IntentFilter refreshFilter = new IntentFilter();
50         refreshFilter.addAction("jade.demo.hotel.ACTIVITIES"
               );
51         registerReceiver(myReceiver, refreshFilter);
52
53         setContentView(R.layout.activity_activity);
54
55         ListView lv = getListView();
56
57         setHandler(lv);
```

```
58
59          adapter=new ArrayAdapter<String>(this,
60                  android.R.layout.simple_list_item_1,
61                  listItems);
62
63          setListAdapter(adapter);
64      }
65
66      private void setHandler(ListView lv) {
67          lv.setOnItemClickListener(new OnItemClickListener()
                {
68              public void onItemClick(AdapterView<?> parent,
                    View view,
69                  int position, long id) {
70
71                  // selected item
72                  String name = ((TextView) view).getText().
                        toString();
73                  String description = nameDescription.get(
                        name);
74                  String price = namePrice.get(name);
75
76                  // Launching new Activity on selecting
                        single List Item
77                  Intent i = new Intent(getApplicationContext
                        (), ListItemActivity.class);
78                  // sending data to new activity
79                  i.putExtra("type", "Atividade");
80                  i.putExtra("name", name);
81                  i.putExtra("description", description);
82                  i.putExtra("price", price);
83                  startActivity(i);
84
85              }
86          });
87      }
88
89      @Override
90      public boolean onCreateOptionsMenu(Menu menu) {
91          MenuInflater inflater = getMenuInflater();
92          inflater.inflate(R.menu.activity_food, menu);
93          return true;
94      }
95
96      @Override
97      public boolean onOptionsItemSelected(MenuItem item) {
98          switch (item.getItemId()) {
99          case R.id.menu_settings:
100             Intent showSettings = new Intent(
                    ActivityActivity.this,
101                         SettingsActivity.class);
102             ActivityActivity.this.startActivityForResult(
                    showSettings,
103                         SETTINGS_REQUEST);
```

```
            return true;
        case R.id.menu_exit:
            finish();
        default:
            return super.onOptionsItemSelected(item);
        }
    }

    private class MyReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent
            ) {
            String action = intent.getAction();
            logger.log(Level.INFO, "Received intent " +
                action);

            if (action.equalsIgnoreCase("jade.demo.hotel.
                ACTIVITIES")) {
                String name = intent.getExtras().getString("
                    nome");
                String description = intent.getExtras().
                    getString("descricao");
                String price = intent.getExtras().getString(
                    "preco");

                nameDescription.put(name, description);
                namePrice.put(name, price);

                listItems.add(name);

                runOnUiThread(new Runnable() {
                    public void run() {
                        adapter.notifyDataSetChanged();
                    }
                });
                //adapter.notifyDataSetChanged();
            }
        }

    }
}
```

2.4 Interface de Cardápio

```
 package client.gui;

import jade.util.Logger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import android.os.Bundle;
import android.app.ListActivity;
```

```
10 import android.content.BroadcastReceiver;
11 import android.content.Context;
12 import android.content.Intent;
13 import android.content.IntentFilter;
14 import android.view.Menu;
15 import android.view.MenuInflater;
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.widget.AdapterView;
19 import android.widget.ArrayAdapter;
20 import android.widget.ListView;
21 import android.widget.TextView;
22 import android.widget.AdapterView.OnItemClickListener;
23
24 public class FoodActivity extends ListActivity {
25
26     private Logger logger = Logger.getJADELogger(this.
        getClass().getName());
27     private MyReceiver myReceiver;
28
29     static final int SETTINGS_REQUEST = 1;
30
31     //LIST OF ARRAY STRINGS WHICH WILL SERVE AS LIST ITEMS
32     ArrayList<String> listItems=new ArrayList<String>();
33
34     //DEFINING STRING ADAPTER WHICH WILL HANDLE DATA OF
        LISTVIEW
35     ArrayAdapter<String> adapter;
36
37     private Map<String, String> nameDescription;
38     private Map<String, String> namePrice;
39
40     @Override
41     public void onCreate(Bundle savedInstanceState) {
42         super.onCreate(savedInstanceState);
43
44         myReceiver = new MyReceiver();
45
46         nameDescription = new HashMap<String, String>();
47         namePrice = new HashMap<String, String>();
48
49         IntentFilter refreshFilter = new IntentFilter();
50         refreshFilter.addAction("jade.demo.hotel.MENU");
51         registerReceiver(myReceiver, refreshFilter);
52
53         setContentView(R.layout.activity_food);
54
55         ListView lv = getListView();
56
57         setHandler(lv);
58
59         adapter=new ArrayAdapter<String>(this,
60                 android.R.layout.simple_list_item_1,
61                 listItems);
```

```
62
63          setListAdapter(adapter);
64      }
65
66      private void setHandler(ListView lv) {
67          lv.setOnItemClickListener(new OnItemClickListener()
                {
68              public void onItemClick(AdapterView<?> parent,
                    View view,
69                  int position, long id) {
70
71                  // selected item
72                  String name = ((TextView) view).getText().
                        toString();
73                  String description = nameDescription.get(
                        name);
74                  String price = namePrice.get(name);
75
76                  // Launching new Activity on selecting
                        single List Item
77                  Intent i = new Intent(getApplicationContext
                        (), ListItemActivity.class);
78                  // sending data to new activity
79                  i.putExtra("type", "Card pio");
80                  i.putExtra("name", name);
81                  i.putExtra("description", description);
82                  i.putExtra("price", price);
83                  startActivity(i);
84
85              }
86            });
87      }
88
89      @Override
90      public boolean onCreateOptionsMenu(Menu menu) {
91          MenuInflater inflater = getMenuInflater();
92          inflater.inflate(R.menu.activity_food, menu);
93          return true;
94      }
95
96      @Override
97      public boolean onOptionsItemSelected(MenuItem item) {
98          switch (item.getItemId()) {
99          case R.id.menu_settings:
100             Intent showSettings = new Intent(FoodActivity.
                    this,
101                         SettingsActivity.class);
102             FoodActivity.this.startActivityForResult(
                    showSettings,
103                         SETTINGS_REQUEST);
104             return true;
105         case R.id.menu_exit:
106             finish();
107         default:
```

```
            return super.onOptionsItemSelected(item);
        }
    }

    private class MyReceiver extends BroadcastReceiver {

        @Override
        public void onReceive(Context context, Intent intent
            ) {
            String action = intent.getAction();
            logger.log(Level.INFO, "Received intent " +
                action);

            if (action.equalsIgnoreCase("jade.demo.hotel.
                MENU")) {
                String name = intent.getExtras().getString("
                    nome");
                String description = intent.getExtras().
                    getString("descricao");
                String price = intent.getExtras().getString(
                    "preco");

                nameDescription.put(name, description);
                namePrice.put(name, price);

                listItems.add(name);

                runOnUiThread(new Runnable() {
                    public void run() {
                        adapter.notifyDataSetChanged();
                    }
                });
                //adapter.notifyDataSetChanged();
            }
        }

    }
}
```

2.5 Interface de Serviços

```
 package client.gui;

import jade.util.Logger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import android.os.Bundle;
import android.app.ListActivity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
```

```
14 import android.view.Menu;
15 import android.view.MenuInflater;
16 import android.view.MenuItem;
17 import android.view.View;
18 import android.widget.AdapterView;
19 import android.widget.AdapterView.OnItemClickListener;
20 import android.widget.ArrayAdapter;
21 import android.widget.ListView;
22 import android.widget.TextView;
23
24 public class ServicesActivity extends ListActivity {
25
26     private Logger logger = Logger.getJADELogger(this.
        getClass().getName());
27     private MyReceiver myReceiver;
28
29     static final int SETTINGS_REQUEST = 1;
30
31     //LIST OF ARRAY STRINGS WHICH WILL SERVE AS LIST ITEMS
32     ArrayList<String> listItems=new ArrayList<String>();
33
34     //DEFINING STRING ADAPTER WHICH WILL HANDLE DATA OF
        LISTVIEW
35     ArrayAdapter<String> adapter;
36
37     private Map<String, String> nameDescription;
38     private Map<String, String> namePrice;
39
40     @Override
41     public void onCreate(Bundle savedInstanceState) {
42         super.onCreate(savedInstanceState);
43
44         myReceiver = new MyReceiver();
45
46         nameDescription = new HashMap<String, String>();
47         namePrice = new HashMap<String, String>();
48
49         IntentFilter refreshFilter = new IntentFilter();
50         refreshFilter.addAction("jade.demo.hotel.SERVICES");
51         registerReceiver(myReceiver, refreshFilter);
52
53         setContentView(R.layout.activity_services);
54
55         ListView lv = getListView();
56
57         setHandler(lv);
58
59         adapter=new ArrayAdapter<String>(this,
60                 android.R.layout.simple_list_item_1,
61                 listItems);
62
63         setListAdapter(adapter);
64     }
65
```

```
66      private void setHandler(ListView lv) {
67          lv.setOnItemClickListener(new OnItemClickListener()
                {
68              public void onItemClick(AdapterView<?> parent,
                    View view,
69                  int position, long id) {
70
71                  // selected item
72                  String name = ((TextView) view).getText().
                        toString();
73                  String description = nameDescription.get(
                        name);
74                  String price = namePrice.get(name);
75
76                  // Launching new Activity on selecting
                        single List Item
77                  Intent i = new Intent(getApplicationContext
                        (), ListItemActivity.class);
78                  // sending data to new activity
79                  i.putExtra("type", "Servi o");
80                  i.putExtra("name", name);
81                  i.putExtra("description", description);
82                  i.putExtra("price", price);
83                  startActivity(i);
84
85              }
86          });
87      }
88
89      @Override
90      public boolean onCreateOptionsMenu(Menu menu) {
91          MenuInflater inflater = getMenuInflater();
92          inflater.inflate(R.menu.activity_services, menu);
93          return true;
94      }
95
96      @Override
97      public boolean onOptionsItemSelected(MenuItem item) {
98          switch (item.getItemId()) {
99          case R.id.menu_settings:
100             Intent showSettings = new Intent(
                    ServicesActivity.this,
101                     SettingsActivity.class);
102             ServicesActivity.this.startActivityForResult(
                    showSettings,
103                     SETTINGS_REQUEST);
104             return true;
105         case R.id.menu_exit:
106             finish();
107         default:
108             return super.onOptionsItemSelected(item);
109         }
110     }
111
```

```
112     private class MyReceiver extends BroadcastReceiver {
113 
114         @Override
115         public void onReceive(Context context, Intent intent
                ) {
116             String action = intent.getAction();
117             logger.log(Level.INFO, "Received intent " +
                    action);
118 
119             if (action.equalsIgnoreCase("jade.demo.hotel.
                    SERVICES")) {
120                 String name = intent.getExtras().getString("
                        nome");
121                 String description = intent.getExtras().
                        getString("descricao");
122                 String price = intent.getExtras().getString(
                        "preco");
123 
124                 nameDescription.put(name, description);
125                 namePrice.put(name, price);
126 
127                 listItems.add(name);
128 
129                 runOnUiThread(new Runnable() {
130                     public void run() {
131                         adapter.notifyDataSetChanged();
132                     }
133                 });
134                 //adapter.notifyDataSetChanged();
135             }
136         }
137 
138 
139     }
140 
141 }
```

2.6 Preferências do Hotel

```
1  package client.gui;
2 
3 import java.util.logging.Level;
4 
5 import jade.util.Logger;
6 import android.app.Application;
7 import android.content.SharedPreferences;
8 
9 
10 public class HotelApplication extends Application {
11     private Logger logger = Logger.getJADELogger(this.
           getClass().getName());
12 
13     @Override
14     public void onCreate() {
```

```
15          super.onCreate();
16
17          SharedPreferences settings = getSharedPreferences("
                jadeChatPrefsFile", 0);
18
19          String defaultHost = settings.getString("defaultHost
                ", "");
20          String defaultPort = settings.getString("defaultPort
                ", "");
21          if (defaultHost.isEmpty() || defaultPort.isEmpty())
                {
22              logger.log(Level.INFO, "Create default
                    properties");
23              SharedPreferences.Editor editor = settings.edit
                    ();
24              editor.putString("defaultHost", "192.168.0.144")
                    ;
25              editor.putString("defaultPort", "1099");
26              editor.commit();
27          }
28      }
29  }
```

2.7 Interface de Listas

```
1   package client.gui;
2
3   import java.util.HashMap;
4   import java.util.Map;
5   import java.util.logging.Level;
6
7   import jade.core.MicroRuntime;
8   import jade.util.Logger;
9   import jade.wrapper.ControllerException;
10  import jade.wrapper.O2AException;
11  import jade.wrapper.StaleProxyException;
12  import client.agent.CellAgentInterface;
13  import android.app.Activity;
14  import android.app.AlertDialog;
15  import android.content.DialogInterface;
16  import android.content.Intent;
17  import android.os.Bundle;
18  import android.view.View;
19  import android.widget.Button;
20  import android.widget.TextView;
21
22  public class ListItemActivity extends Activity{
23
24      private Logger logger = Logger.getJADELogger(this.
            getClass().getName());
25      private CellAgentInterface cellAgentInterface;
26      private String nickname = "tarcisio";
27
28      @Override
```

```
29      public void onCreate(Bundle savedInstanceState) {
30          super.onCreate(savedInstanceState);
31
32          //
33  //          Bundle extras = this.getIntent().getExtras();
34  //          if (extras != null) {
35  //              nickname = extras.getString("agente");
36  //              logger.log(Level.INFO, "Aqui " + nickname);
37  //          }
38
39          try {
40              //PROBLEM HERE
41              /*cellAgentInterface = MicroRuntime.getAgent(
                    nickname)
42                          .getO2AInterface(CellAgentInterface.
                            class);*/
43              logger.log(Level.INFO, "Received intent " +
                    nickname);
44
45              cellAgentInterface = MicroRuntime.getAgent(
                    nickname)
46                          .getO2AInterface(CellAgentInterface.
                            class);
47
48          } catch (StaleProxyException e) {
49              showAlertDialog(getString(R.string.
                    msg_interface_exc), true);
50          } catch (ControllerException e) {
51              showAlertDialog(getString(R.string.
                    msg_controller_exc), true);
52          }
53          //
54
55          this.setContentView(R.layout.activity_list_item);
56
57          TextView type = (TextView) findViewById(R.id.type);
58          TextView name = (TextView) findViewById(R.id.name);
59          TextView description = (TextView) findViewById(R.id.
                description);
60          TextView price = (TextView) findViewById(R.id.price)
                ;
61
62          Intent i = getIntent();
63
64          String s_type = i.getStringExtra("type") + "\n";
65          String s_name = "Nome: " + i.getStringExtra("name")
                + "\n";
66          String s_description = "Descri  o: " + i.
                getStringExtra("description") + "\n";
67          String s_price = "Pre o: " + i.getStringExtra("
                price") + "\n";
68
69          type.setText(s_type);
70          name.setText(s_name);
```

```
71          description.setText(s_description);
72          price.setText(s_price);
73
74          final Button button = (Button) findViewById(R.id.buy
                );
75          button.setOnClickListener(new View.OnClickListener()
                {
76              public void onClick(View v) {
77                  // Perform action on click
78                  Map<String, String> msg = new HashMap<String
                        , String>();
79                  TextView view1 = (TextView) findViewById(R.
                        id.name);
80                  TextView view2 = (TextView) findViewById(R.
                        id.description);
81                  TextView view3 = (TextView) findViewById(R.
                        id.price);
82                  msg.put("nome", (String) view1.getText());
83                  msg.put("descricao", (String) view2.getText
                        ());
84                  msg.put("preco", (String) view3.getText());
85
86                  try {
87                      cellAgentInterface.handleSpoken(msg);
88                  } catch (O2AException e) {
89                      showAlertDialog(e.getMessage(), false);
90                  }
91
92                  String[] produto = view1.getText().toString
                        ().split(":");
93
94                  showAlertDialog("Voc  comprou " + produto
                        [1].trim(), true);
95
96              }
97          });
98
99      }
100
101     private void showAlertDialog(String message, final
            boolean fatal) {
102         AlertDialog.Builder builder = new AlertDialog.
                Builder(
103                 ListItemActivity.this);
104         builder.setMessage(message)
105                 .setCancelable(false)
106                 .setPositiveButton("Ok",
107                         new DialogInterface.OnClickListener
                                () {
108                             public void onClick(
109                                     DialogInterface dialog,
                                            int id) {
110                                 dialog.cancel();
111                                 if(fatal) finish();
```

```
                            }
                        });
        AlertDialog alert = builder.create();
        alert.show();
    }


}
```

2.8 Interface Principal

```
 package client.gui;

import java.util.logging.Level;

import client.agent.CellPhoneAgent;
import client.gui.R;
import jade.android.AndroidHelper;
import jade.android.MicroRuntimeService;
import jade.android.MicroRuntimeServiceBinder;
import jade.android.RuntimeCallback;
import jade.core.MicroRuntime;
import jade.core.Profile;
import jade.util.Logger;
import jade.util.leap.Properties;
import jade.wrapper.AgentController;
import jade.wrapper.ControllerException;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {
    private Logger logger = Logger.getJADELogger(this.
        getClass().getName());

```

```
43      private MicroRuntimeServiceBinder
            microRuntimeServiceBinder;
44      private ServiceConnection serviceConnection;
45
46      static final int CHAT_REQUEST = 0;
47      static final int SETTINGS_REQUEST = 1;
48
49      private MyReceiver myReceiver;
50      private MyHandler myHandler;
51
52      private TextView infoTextView;
53
54      private String nickname;
55
56      @Override
57      public void onCreate(Bundle savedInstanceState) {
58          super.onCreate(savedInstanceState);
59
60          myReceiver = new MyReceiver();
61
62          IntentFilter killFilter = new IntentFilter();
63          killFilter.addAction("jade.demo.chat.KILL");
64          registerReceiver(myReceiver, killFilter);
65
66          IntentFilter showChatFilter = new IntentFilter();
67          showChatFilter.addAction("jade.demo.hotel.SHOW_TABS"
                );
68          registerReceiver(myReceiver, showChatFilter);
69
70          myHandler = new MyHandler();
71
72          setContentView(R.layout.main);
73
74          Button button = (Button) findViewById(R.id.
                button_chat);
75          button.setOnClickListener(buttonChatListener);
76
77          infoTextView = (TextView) findViewById(R.id.
                infoTextView);
78          infoTextView.setText("");
79      }
80
81      @Override
82      protected void onDestroy() {
83          super.onDestroy();
84
85          unregisterReceiver(myReceiver);
86
87          logger.log(Level.INFO, "Destroy activity!");
88      }
89
90      private static boolean checkName(String name) {
91          if (name == null || name.trim().equals("")) {
92              return false;
```

```
93          }
94          // FIXME: should also check that name is composed
95          // of letters and digits only
96          return true;
97      }
98
99      private OnClickListener buttonChatListener = new
            OnClickListener() {
100         public void onClick(View v) {
101             final EditText nameField = (EditText)
                    findViewById(R.id.edit_nickname);
102             nickname = nameField.getText().toString();
103             if (!checkName(nickname)) {
104                 logger.log(Level.INFO, "Invalid nickname!");
105                 myHandler.postError(getString(R.string.
                        msg_nickname_not_valid));
106             } else {
107                 try {
108                     SharedPreferences settings =
                            getSharedPreferences(
109                                 "jadeChatPrefsFile", 0);
110                     String host = settings.getString("
                            defaultHost", "");
111                     String port = settings.getString("
                            defaultPort", "");
112                     infoTextView.setText(getString(R.string.
                            msg_connecting_to)
113                             + " " + host + ":" + port + "...
                                ");
114                     startChat(nickname, host, port,
                            agentStartupCallback);
115                 } catch (Exception ex) {
116                     logger.log(Level.SEVERE, "Unexpected
                            exception creating chat agent!");
117                     infoTextView.setText(getString(R.string.
                            msg_unexpected));
118                 }
119             }
120         }
121     };
122
123     @Override
124     public boolean onCreateOptionsMenu(Menu menu) {
125         MenuInflater inflater = getMenuInflater();
126         inflater.inflate(R.menu.main_menu, menu);
127         return true;
128     }
129
130     @Override
131     public boolean onOptionsItemSelected(MenuItem item) {
132         switch (item.getItemId()) {
133         case R.id.menu_settings:
134             Intent showSettings = new Intent(MainActivity.
                    this,
```

```
135                     SettingsActivity.class);
136             MainActivity.this.startActivityForResult(
                    showSettings,
137                     SETTINGS_REQUEST);
138             return true;
139         case R.id.menu_exit:
140             finish();
141         default:
142             return super.onOptionsItemSelected(item);
143         }
144     }
145
146     @Override
147     protected void onActivityResult(int requestCode, int
            resultCode, Intent data) {
148         if (requestCode == CHAT_REQUEST) {
149             if (resultCode == RESULT_CANCELED) {
150                 // The chat activity was closed.
151                 infoTextView.setText("");
152                 logger.log(Level.INFO, "Stopping Jade...");
153                 microRuntimeServiceBinder
154                         .stopAgentContainer(new
                                RuntimeCallback<Void>() {
155                             @Override
156                             public void onSuccess(Void
                                    thisIsNull) {
157                             }
158
159                             @Override
160                             public void onFailure(Throwable
                                    throwable) {
161                                 logger.log(Level.SEVERE, "
                                        Failed to stop the "
162                                         + CellPhoneAgent.
                                            class.getName()
163                                         + "...");
164                                 agentStartupCallback.
                                        onFailure(throwable);
165                             }
166                         });
167             }
168         }
169     }
170
171     private RuntimeCallback<AgentController>
            agentStartupCallback = new RuntimeCallback<
            AgentController>() {
172         @Override
173         public void onSuccess(AgentController agent) {
174         }
175
176         @Override
177         public void onFailure(Throwable throwable) {
178             logger.log(Level.INFO, "Nickname already in use!
```

```
                ");
179             myHandler.postError(getString(R.string.
                    msg_nickname_in_use));
180         }
181     };
182 
183     public void ShowDialog(String message) {
184         AlertDialog.Builder builder = new AlertDialog.
                Builder(MainActivity.this);
185         builder.setMessage(message).setCancelable(false)
186                 .setPositiveButton("Ok", new DialogInterface
                        .OnClickListener() {
187                     public void onClick(DialogInterface
                            dialog, int id) {
188                         dialog.cancel();
189                     }
190                 });
191         AlertDialog alert = builder.create();
192         alert.show();
193     }
194 
195     private class MyReceiver extends BroadcastReceiver {
196 
197         @Override
198         public void onReceive(Context context, Intent intent
                ) {
199             String action = intent.getAction();
200             logger.log(Level.INFO, "Received intent " +
                    action);
201             if (action.equalsIgnoreCase("jade.demo.chat.KILL
                    ")) {
202                 finish();
203             }
204             if (action.equalsIgnoreCase("jade.demo.hotel.
                    SHOW_TABS")) {
205                 Intent showTabs = new Intent(MainActivity.
                        this,
206                         TabWidgetActivity.class);
207                 //showTabs.putExtra("nickname", nickname);
208                 MainActivity.this
209                         .startActivityForResult(showTabs,
                                CHAT_REQUEST);
210             }
211         }
212     }
213 
214     private class MyHandler extends Handler {
215         @Override
216         public void handleMessage(Message msg) {
217             Bundle bundle = msg.getData();
218             if (bundle.containsKey("error")) {
219                 infoTextView.setText("");
220                 String message = bundle.getString("error");
221                 ShowDialog(message);
```

```
222             }
223         }
224 
225         public void postError(String error) {
226             Message msg = obtainMessage();
227             Bundle b = new Bundle();
228             b.putString("error", error);
229             msg.setData(b);
230             sendMessage(msg);
231         }
232     }
233 
234     public void startChat(final String nickname, final
        String host,
235             final String port,
236             final RuntimeCallback<AgentController>
                agentStartupCallback) {
237 
238         final Properties profile = new Properties();
239         profile.setProperty(Profile.MAIN_HOST, host);
240         profile.setProperty(Profile.MAIN_PORT, port);
241         profile.setProperty(Profile.MAIN, Boolean.FALSE.
            toString());
242         profile.setProperty(Profile.JVM, Profile.ANDROID);
243 
244         if (AndroidHelper.isEmulator()) {
245             // Emulator: this is needed to work with
                emulated devices
246             profile.setProperty(Profile.LOCAL_HOST,
                AndroidHelper.LOOPBACK);
247         } else {
248             profile.setProperty(Profile.LOCAL_HOST,
249                     AndroidHelper.getLocalIPAddress());
250         }
251         // Emulator: this is not really needed on a real
            device
252         profile.setProperty(Profile.LOCAL_PORT, "2000");
253 
254         if (microRuntimeServiceBinder == null) {
255             serviceConnection = new ServiceConnection() {
256                 public void onServiceConnected(ComponentName
                     className,
257                         IBinder service) {
258                     microRuntimeServiceBinder = (
                        MicroRuntimeServiceBinder) service;
259                     logger.log(Level.INFO, "Gateway
                        successfully bound to
                        MicroRuntimeService");
260                     startContainer(nickname, profile,
                        agentStartupCallback);
261                 };
262 
263                 public void onServiceDisconnected(
                    ComponentName className) {
```

```
264                     microRuntimeServiceBinder = null;
265                     logger.log(Level.INFO, "Gateway unbound
                            from MicroRuntimeService");
266                 }
267             };
268             logger.log(Level.INFO, "Binding Gateway to
                    MicroRuntimeService...");
269             bindService(new Intent(getApplicationContext(),
270                     MicroRuntimeService.class),
                            serviceConnection,
271                     Context.BIND_AUTO_CREATE);
272         } else {
273             logger.log(Level.INFO, "MicroRumtimeGateway
                    already binded to service");
274             startContainer(nickname, profile,
                    agentStartupCallback);
275         }
276     }
277
278     private void startContainer(final String nickname,
            Properties profile,
279             final RuntimeCallback<AgentController>
                    agentStartupCallback) {
280         if (!MicroRuntime.isRunning()) {
281             microRuntimeServiceBinder.startAgentContainer(
                    profile,
282                     new RuntimeCallback<Void>() {
283                         @Override
284                         public void onSuccess(Void
                                thisIsNull) {
285                             logger.log(Level.INFO, "
                                    Successfully start of the
                                    container...");
286                             startAgent(nickname,
                                    agentStartupCallback);
287                         }
288
289                         @Override
290                         public void onFailure(Throwable
                                throwable) {
291                             logger.log(Level.SEVERE, "Failed
                                     to start the container...");
292                         }
293                     });
294         } else {
295             startAgent(nickname, agentStartupCallback);
296         }
297     }
298
299     private void startAgent(final String nickname,
300             final RuntimeCallback<AgentController>
                    agentStartupCallback) {
301         microRuntimeServiceBinder.startAgent(nickname,
302                 CellPhoneAgent.class.getName(),
```

```
                new Object[] { getApplicationContext() },
                new RuntimeCallback<Void>() {
                    @Override
                    public void onSuccess(Void thisIsNull) {
                        logger.log(Level.INFO, "Successfully start of the "
                                + CellPhoneAgent.class.getName() + "...");
                        try {
                            agentStartupCallback.onSuccess(MicroRuntime
                                    .getAgent(nickname));
                        } catch (ControllerException e) {
                            // Should never happen
                            agentStartupCallback.onFailure(e);
                        }
                    }

                    @Override
                    public void onFailure(Throwable throwable) {
                        logger.log(Level.SEVERE, "Failed to start the "
                                + CellPhoneAgent.class.getName() + "...");
                        agentStartupCallback.onFailure(throwable);
                    }
                });
    }

}
```

2.8.1 Interface de Edição de IP

```
 package client.gui;

import jade.util.leap.Properties;
import android.app.Activity;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class SettingsActivity extends Activity {
    Properties properties;
    EditText hostField;
    EditText portField;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
20
21          SharedPreferences settings = getSharedPreferences("
                jadeChatPrefsFile",
22                  0);
23
24          String host = settings.getString("defaultHost", "");
25          String port = settings.getString("defaultPort", "");
26
27          setContentView(R.layout.settings);
28
29          hostField = (EditText) findViewById(R.id.edit_host);
30          hostField.setText(host);
31
32          portField = (EditText) findViewById(R.id.edit_port);
33          portField.setText(port);
34
35          Button button = (Button) findViewById(R.id.
                button_use);
36          button.setOnClickListener(buttonUseListener);
37      }
38
39      private OnClickListener buttonUseListener = new
            OnClickListener() {
40          public void onClick(View v) {
41              SharedPreferences settings =
                    getSharedPreferences(
42                          "jadeChatPrefsFile", 0);
43
44              // TODO: Verify that edited parameters was
                    formally correct
45              SharedPreferences.Editor editor = settings.edit
                    ();
46              editor.putString("defaultHost", hostField.
                    getText().toString());
47              editor.putString("defaultPort", portField.
                    getText().toString());
48              editor.commit();
49
50              finish();
51          }
52      };
53  }
```

2.8.2 Interface de Abas

```
1   package client.gui;
2
3   import client.gui.R;
4   import android.app.TabActivity;
5   import android.content.Intent;
6   import android.os.Bundle;
7   import android.widget.TabHost;
8
9   public class TabWidgetActivity extends TabActivity {
10
```

```
11      TabHost tbHost;
12
13      @Override
14      public void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_tab_widget);
17
18          tbHost = getTabHost();
19          TabHost.TabSpec Spec;
20          Intent intent;
21
22          intent = new Intent().setClass(this,
                ActivityActivity.class);
23          Spec = tbHost.newTabSpec("tab1").setIndicator("
                Activities")
24          .setContent(intent);
25          tbHost.addTab(Spec);
26
27          intent = new Intent().setClass(this, FoodActivity.
                class);
28          Spec = tbHost.newTabSpec("tab2").setIndicator("Food"
                )
29          .setContent(intent);
30          tbHost.addTab(Spec);
31
32          intent = new Intent().setClass(this,
                ServicesActivity.class);
33          Spec = tbHost.newTabSpec("tab3").setIndicator("
                Services")
34          .setContent(intent);
35          tbHost.addTab(Spec);
36
37          initTabs();
38      }
39
40      private void initTabs() {
41          tbHost.setCurrentTab(0);
42          tbHost.setCurrentTab(1);
43          tbHost.setCurrentTab(2);
44          tbHost.setCurrentTab(0);
45      }
46  }
```

7.3 Ontologia

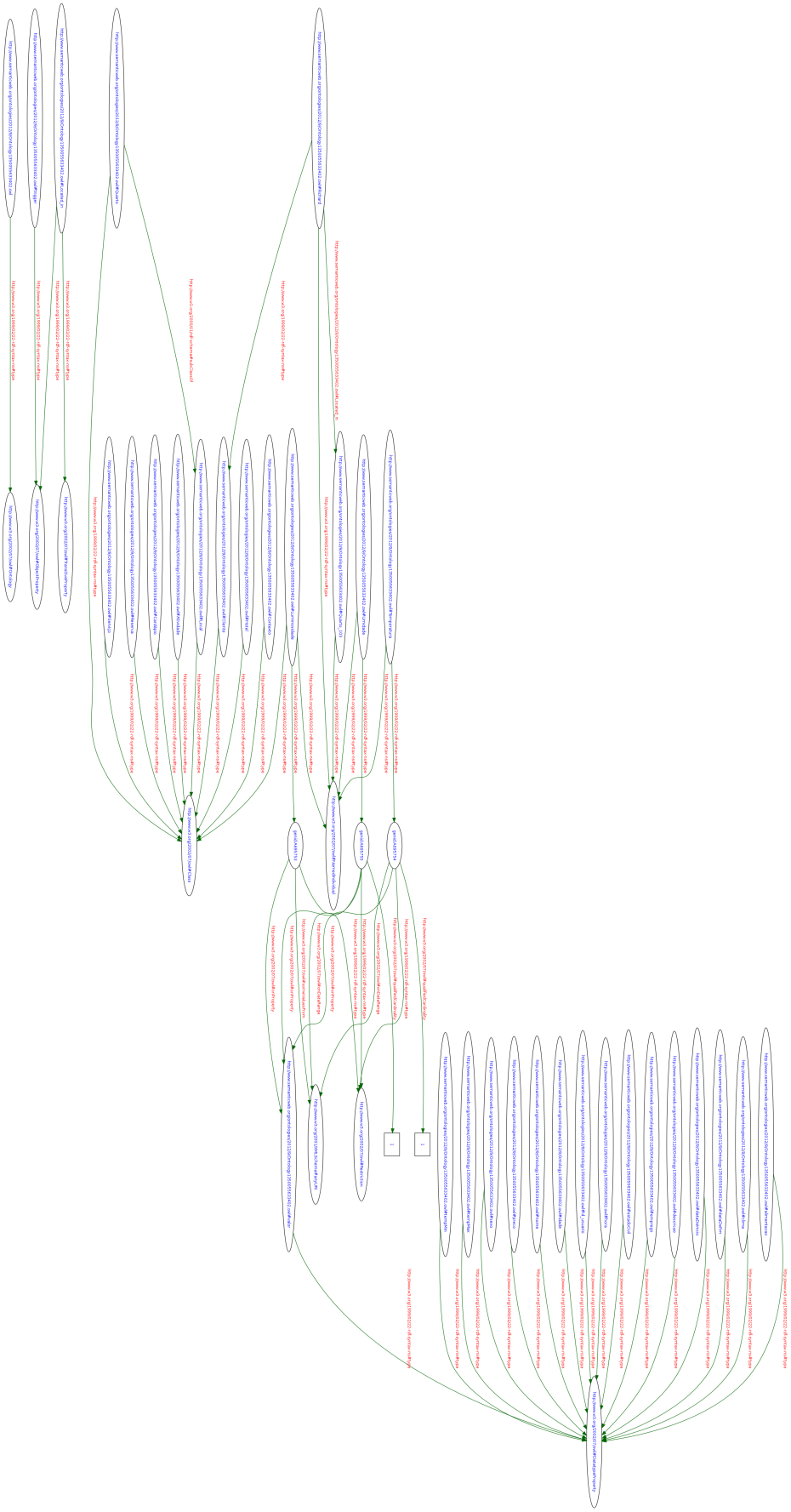

Figura 7.1: Ontologia criada para o estudo de caso.