



Programa Interdisciplinar de Pós-Graduação em  
**Computação Aplicada**  
Mestrado Acadêmico

Paulo César Büttendender

Uma Plataforma para Computação Sensível a Contexto  
Baseada em Tecnologias Web e Armazenamento Distribuído  
de Ontologias

São Leopoldo, 2013

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS  
CIÊNCIAS EXATAS E TECNOLÓGICAS  
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO EM  
COMPUTAÇÃO APLICADA  
NÍVEL MESTRADO

PAULO CÉSAR BÜTTENBENDER

UMA PLATAFORMA PARA COMPUTAÇÃO SENSÍVEL A CONTEXTO  
BASEADA EM TECNOLOGIAS WEB E ARMAZENAMENTO  
DISTRIBUÍDO DE ONTOLOGIAS

SÃO LEOPOLDO

2013

Paulo César Büttenbender

UMA PLATAFORMA PARA COMPUTAÇÃO SENSÍVEL A CONTEXTO  
BASEADA EM TECNOLOGIAS WEB E ARMAZENAMENTO  
DISTRIBUÍDO DE ONTOLOGIAS

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Computação Aplicada, pelo Programa Interdisciplinar de Pós-graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos.

Orientador: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto

SÃO LEOPOLDO

2013

Ficha catalográfica

B988p Büttendbender, Paulo César

Uma plataforma para computação sensível a contexto baseada em tecnologias web e armazenamento distribuído de ontologias / por Paulo César Büttendbender. – 2013.

131 f. : il., 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2013.

Orientação: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto.

1. Sistemas orientados a contextos. 2. Agentes de software. 3. Ontologias. 4. Computação na nuvem. I. Título.

CDU 004.5

Catálogo na Fonte:  
Bibliotecária Vanessa Borges Nunes - CRB 10/1556

Paulo César Büttenbender

UMA PLATAFORMA PARA COMPUTAÇÃO SENSÍVEL A CONTEXTO  
BASEADA EM TECNOLOGIAS WEB E ARMAZENAMENTO  
DISTRIBUÍDO DE ONTOLOGIAS

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Computação Aplicada, pelo Programa Interdisciplinar de Pós-graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos.

Orientador: Prof. Dr. Sérgio Crespo Coelho da Silva Pinto

Aprovado em \_\_/\_\_/\_\_\_\_.

BANCA EXAMINADORA

---

Prof. Dr. Sérgio Crespo – Orientador

---

Prof. Dr. Jorge Barbosa – Unisinos

---

Prof. Dr. Crediné Menezes – UFES

*Esta Dissertação de Mestrado é dedicada  
à minha esposa Mirian e ao meu filho Theo,  
pelo apoio incondicional e pelo tempo que deixamos de estar juntos...  
Aos meus pais, Anaide e Antônio, pelo incentivo e compreensão.*

## **AGRADECIMENTOS**

Ao Prof. Dr. Sérgio Crespo, por acreditar na ideia e conduzir os trabalhos rumo aos resultados obtidos.

Ao Prof. Dr. Jorge Barbosa pela troca de ideias, e por incentivar a interação com hardware neste trabalho.

Aos meus colegas de pós-graduação e pesquisa que tornaram este período de dedicação em algo divertido.

## RESUMO

A internet das coisas tem por objetivo utilizar a infraestrutura existente da internet para conectar sistemas, pessoas e objetos, permitindo um monitoramento e interação contínua. Essa visão está diretamente relacionada aos trabalhos já existentes na área de computação pervasiva.

Uma característica comum aos trabalhos de computação pervasiva é que as suas respectivas plataformas são fechadas e exclusivas a um trabalho ou uma instituição e, portanto a aplicação dos conceitos de computação pervasiva exige dos autores uma construção do software de uma forma específica para o domínio de conhecimento onde será aplicado, o que acarreta em pouca ou nenhuma forma de compartilhamento das informações obtidas.

Este trabalho tem como objetivo prover uma plataforma para melhorar a comunicação entre ambientes inteligentes e pessoas, utilizando dispositivos móveis como meio de interação e possibilitando a exposição de contextos pela internet para dispositivos inteligentes externos ao ambiente inteligente utilizando a infraestrutura existente de computação na nuvem e a capacidade de compartilhamento de conhecimento através de ontologias.

Palavras-chave: Sistemas orientados a contexto. Agentes de software. Ontologias. Computação na nuvem.



## **ABSTRACT**

The main objective of the internet of things is to use the existing internet infrastructure to connect systems, people and objects, allowing constant interaction and monitoring. This vision is directly related to existing research within pervasive computing.

A common characteristic among pervasive computing research papers is that usually their respective platforms are closed and build exclusively to a specific objective or for a given institution, and therefore applying pervasive computing concepts usually demand from the authors to develop the software constrained to the knowledge domain where pervasive computing is going to be applied, which incur in few or none ways of sharing information with other projects.

The objective of this research is to provide a common platform to enhance the communication between intelligent environments and people, using mobile devices as a way of interaction, allowing user contexts to be exposed in the Internet to intelligent devices which are not contained within the intelligent environment where the user is actually located by using existing cloud computing infrastructure and web standards ontology to share knowledge.

**Keywords:** Context-aware systems. Software agents. Ontology. Cloud computing.

## LISTA DE FIGURAS

Figura 1 - Ontologia de vinho cedida pela W3C .....	27
Figura 2 - Fragmento de ontologia de famílias OWL 2 .....	30
Figura 3 - Arquitetura do Jena .....	33
Figura 4 - Relação entre agente e ambiente .....	34
Figura 5 - Arquitetura do HDFS.....	41
Figura 6 - Exemplo de tabela para armazenamento de páginas da Web, adaptado de (Chang, Dean and Ghemawat, 2008) .....	43
Figura 7 - Definições e exemplo de ontologia no modelo proposto .....	45
Figura 8 - Arquitetura do <i>EasyMeeting</i> , fonte (Chen, Finin and Joshi, 2004).....	48
Figura 9 - Arquitetura do sistema e o papel do middleware, fonte (Song, Alvaro and Masuoka, 2012).....	49
Figura 10 - Infraestrutura de contexto do Gaia, fonte (Ranganathan and Campbell, 2003).....	51
Figura 11 - Modelo e exemplo de armazenamento de triplas RDF no HBase, adaptado de (Khadilkar and Kantarcioglu, 2012).....	55
Figura 12 – Relacionamento entre ontologias na SOUPA-core e parte da SOUPA-extension, adaptado de (Chen, Finin and Joshi, 2005).....	56
Figura 13 - Visão do programa "desktop semântico" .....	58
Figura 14 - Ilustração para o cenário de uso online.....	60
Figura 15 - Arquitetura do Histórico de Contexto na Nuvem .....	63
Figura 16 - Ontologia de Tempo.....	65
Figura 17 - Ontologia de Localização .....	66
Figura 18 - Ontologia de Agentes.....	67
Figura 19 - Ontologia de Contexto .....	68
Figura 20 - Diagrama de classe do processo de inicialização.....	69
Figura 21 - Diagrama de sequência do processo de inicialização .....	70
Figura 22 - Adaptação do formato simples de armazenamento de triplas .....	71
Figura 23 - Exemplo com fragmento de tabela de sujeitos .....	71
Figura 24 - Diagrama de classes de armazenamento de grafos RDF no HBase .....	73
Figura 25 - Diagrama de classe das entidades do serviço de contexto .....	77
Figura 26 - Arquitetura do Motor de Execução de Agentes.....	81
Figura 27 - Estrutura das instruções da API Agente JS.....	84
Figura 28 – Diagrama de classe dos componentes da API.....	85
Figura 29 - Diagrama das classes de emissão de sinal.....	97
Figura 30 - Classes relacionadas a execução de agentes.....	102
Figura 31 - Serviço UPnP para descoberta de agentes .....	102
Figura 32 - Diagrama de classe do módulo de serviços UPnP.....	103
Figura 33 - Diagrama das classes envolvidas na atualização de contexto .....	104
Figura 34 - Resultado das operações de busca no banco de dados distribuído.....	107
Figura 35 - Validação do serviço de consulta SPARQL.....	108
Figura 36 - Tela para manutenção de agentes .....	109
Figura 37 - (a) Preferência do histórico de contextos, (b) Busca SPARQL para listar aplicativos executados após um intervalo pré-definido.....	110
Figura 38 - Informação no emulador do contexto sendo emitido para o servidor de aplicação local.....	110
Figura 39 - Resultado da seleção de aplicativos em execução em um determinado intervalo .....	111

Figura 40 - Agentes existentes no servidor de aplicação e instrução do agente “Notificador” .....	111
Figura 41 - (a) Lista de agentes no cartão SD e no servidor de aplicação, (b) Resultado da interpretação do agente Notificador .....	112
Figura 42 - (a) Listagem de agentes descobertos via UPnP, (b) Resultado da interpretação do agente descoberto via UPnP .....	113
Figura 43 - Rota simulada pelo emulador em branco, marcadores com todas as pizzarias próximas a rota .....	113
Figura 44 - (a) Código do agente de Busca por pizzaria, (b) Última notificação emitida, correspondente a Pizzaria 4 .....	114
Figura 45 - (a) Código do agente de Sms Música, (b) Música iniciada logo após o recebimento do SMS de alerta .....	114
Figura 46 - Código do agente de validação do componente de HTTP.....	115
Figura 47 - Resultado da requisição no log do dispositivo móvel.....	115
Figura 48 – (a) Código do teste de inicialização de aplicativos por agentes, (b) aplicativo aberto após a interpretação do agente .....	116
Figura 49 - Motorola Dext listando os agentes encontrados na rede.....	117
Figura 50 - Raspberry Pi.....	118
Figura 51 - Instâncias Amazon EC2 alocadas em São Paulo .....	119
Figura 52 - Consulta SPARQL no servidor de aplicação .....	119
Figura 53 - Tempo de resposta por Quantidade de contextos.....	123
Figura 54 - Comparativo entre consulta SPARQL otimizada e consulta comum .....	124

## LISTA DE TABELAS

Tabela 1 - Componentes básicos do diagrama de blocos.....	44
Tabela 2 - Comparativo entre trabalhos relacionados e trabalho proposto.....	53
Tabela 3 - Interface do serviço de consulta SPARQL.....	74
Tabela 4 - Interface para o serviço de criação de agentes.....	75
Tabela 5 - Interface para o serviço de atualização de contexto.....	77
Tabela 6 - Instruções compartilhadas entre componentes .....	86
Tabela 7 - Instruções do componente de API <i>music</i> .....	87
Tabela 8 - Instruções do componente de API <i>sms</i> .....	88
Tabela 9 - Instruções do componente de API <i>notification</i> .....	89
Tabela 10 - Instruções do componente de API <i>network</i> .....	90
Tabela 11 - Instruções do componente de API <i>location</i> .....	91
Tabela 12 - Instruções do componente de API <i>http</i> .....	93
Tabela 13 - Instruções do componente de API <i>apps</i> .....	95
Tabela 14 - Consulta C1 .....	120
Tabela 15 - Consulta C2 .....	120
Tabela 16 - Consulta C3 .....	121
Tabela 17 - Consulta C4 .....	122

## LISTA DE ABREVIATURAS

API	<i>Application programming interface</i>
FIPA	<i>Foundation for Intelligent Physical Agents</i>
FOAF	<i>Friend of a Friend</i>
HTML	<i>Hypertext Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
JADE	<i>Java Agent Development Framework</i>
OWL	<i>Web Ontology Language</i>
PIPCA	<i>Programa Interdisciplinar de Pós-Graduação em Computação Aplicada</i>
RDF	<i>Resource Description Framework</i>
RDFS	<i>RDF Schema</i>
SOAP	<i>Simple Object Access Protocol</i>
SPARQL	<i>SPARQL Protocol and RDF Query Language</i>
SQL	<i>Structured Query Language</i>
TAM	<i>Technical Architecture Modeling</i>
UDP	<i>User Datagram Protocol</i>
URI	<i>Uniform Resource Identifier</i>
URL	<i>Uniform Resource Locator</i>
XML	<i>Extensible Markup Language</i>
W3C	<i>World Wide Web Consortium</i>

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>16</b>
1.1	CONTEXTO E MOTIVAÇÃO	17
1.2	QUESTÃO DE PESQUISA	17
1.3	OBJETIVO DA PESQUISA	18
1.4	METODOLOGIA	18
1.5	ORGANIZAÇÃO DO DOCUMENTO	19
<b>2</b>	<b>ARCABOUÇO TEÓRICO</b>	<b>20</b>
2.1	COMPUTAÇÃO PERVASIVA	20
2.1.1	<i>Localização indoor em dispositivos móveis</i>	21
2.1.2	<i>Sistemas orientados a contexto</i>	22
2.1.3	<i>Descoberta de recursos e serviços</i>	23
2.2	ONTOLOGIAS	26
2.2.1	<i>RDF</i>	27
2.2.2	<i>OWL</i>	28
2.2.3	<i>SPARQL</i>	31
2.2.4	<i>Jena</i>	32
2.3	AGENTES DE SOFTWARE	34
2.3.1	<i>JADE</i>	35
2.4	COMPUTAÇÃO NA NUVEM	36
2.5	PLATAFORMAS, LINGUAGENS E ARMAZENAMENTO	38
2.5.1	<i>Android</i>	38
2.5.2	<i>Javascript</i>	39
2.5.3	<i>Apache Hadoop</i>	40
2.5.4	<i>Apache HBase</i>	42
2.6	LINGUAGEM DE MODELAGEM	43
2.6.1	<i>Linguagem de modelagem para arquitetura e implementação</i>	43
2.6.2	<i>Padrão simplificado para representação gráfica de ontologias OWL</i>	44
2.7	FECHAMENTO DO CAPÍTULO	45
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>47</b>
3.1	EASYMEETING E CONTEXT BROKER ARCHITECTURE	47
3.2	MIDDLEWARE SEMÂNTICO PARA INTERNET DAS COISAS	48
3.3	MIDDLEWARE SEMÂNTICO INTELIGENTE PARA INTERNET DAS COISAS	49
3.4	MIDDLEWARE PARA AGENTES SENSÍVEIS A CONTEXTO EM AMBIENTES DE COMPUTAÇÃO UBÍQUA	50
3.5	UM MIDDLEWARE PARA AMBIENTES INTELIGENTES E INTERNET DAS COISAS	51
3.6	COMPARATIVO ENTRE TRABALHOS RELACIONADOS	52
3.7	UMA FORMA DE ARMAZENAMENTO DE TRIPLAS RDF DISTRIBUIDA, ESCALAVEL E EFICIENTE BASEADA EM JENA E HBASE	53
3.8	A ONTOLOGIA SOUPA PARA COMPUTAÇÃO PERVASIVA	55
3.9	APLICATIVO MICROSOFT ON{X}	56
<b>4</b>	<b>PROPOSTA DE TRABALHO</b>	<b>58</b>
4.1	PROGRAMA DESKTOP SEMÂNTICO	58
4.2	VISÃO GERAL DA PROPOSTA DE TRABALHO	60
4.3	HISTÓRICO DE CONTEXTOS NA NUVEM	60
4.3.1	<i>Requisitos do componente</i>	61
4.3.2	<i>Arquitetura do componente</i>	62
4.3.3	<i>Modelagem do contexto em ontologias</i>	64
4.3.4	<i>Inicialização do servidor e leitura dos modelos</i>	68

4.3.5	<i>Armazenamento e leitura de dados no HBase</i> .....	70
4.3.6	<i>Serviços REST do histórico de contexto</i> .....	73
4.4	<b>MOTOR DE EXECUÇÃO DE AGENTES</b> .....	78
4.4.1	<i>Requisitos do sistema</i> .....	79
4.4.2	<i>Arquitetura do componente</i> .....	80
4.4.3	<i>A API Agente JS</i> .....	83
4.4.4	<i>Emissor de sinal</i> .....	96
4.4.5	<i>Controle de execução de agentes</i> .....	99
4.4.6	<i>Descoberta de agentes em via UPnP</i> .....	102
4.4.7	<i>Atualizador de contexto</i> .....	104
<b>5</b>	<b>VALIDAÇÃO E TESTES DA PLATAFORMA</b> .....	<b>105</b>
5.1	PLANEJAMENTO DA VALIDAÇÃO E LIMITAÇÃO DA METODOLOGIA .....	105
5.2	TESTE DE FUNCIONALIDADE EM AMBIENTE VIRTUAL .....	105
5.3	TESTE DE FUNCIONALIDADE EM AMBIENTE REAL .....	116
5.4	TESTE DE DESEMPENHO.....	120
<b>6</b>	<b>CONCLUSÃO</b> .....	<b>125</b>
6.1	PRINCIPAIS CONTRIBUIÇÕES .....	125
6.2	TRABALHOS FUTUROS .....	126

## 1 INTRODUÇÃO

Conforme identificado por (Haller, 2010), o termo “internet das coisas” ainda não possui uma definição universal, mas possui em comum a característica de ser utilizado para se referir a todo o conjunto de tecnologias necessárias para integrar a comunicação entre dispositivos físicos com serviços e aplicações disponíveis na internet, em algo que é referenciado em seu artigo como o futuro da internet.

Neste mesmo trabalho, o autor apresenta uma descrição dos termos relevantes à internet das coisas, que é apresentada como sendo formada por entidades de interesse, também referenciado como “coisas” (*things*), dispositivos (*devices*), recursos (*resources*) e serviços (*services*).

Um dispositivo pode ser considerado um subtipo de entidade de interesse, mas com a diferença de que uma entidade de interesse é um objeto que tem algum tipo de valor ao observador (podendo ser um observador máquina ou pessoa), enquanto um dispositivo é um componente técnico necessário para permitir que um objeto seja monitorado ou interaja com outras entidades de interesse (via serviços, e pela internet).

Recursos são as informações contidas ou monitoradas por dispositivos (como por exemplo, a temperatura de um ambiente, ou as coordenadas de um container em um determinado momento), e acessíveis via serviços.

Estes conceitos são a base do que é a visão da “internet das coisas”, cujo objetivo é utilizar a infraestrutura existente da internet para conectar não somente sistemas e pessoas, bem como objetos, permitindo assim um monitoramento e interação contínua.

Parte desta visão já vem sendo explorada e pesquisada pela área de computação pervasiva, especialmente nas áreas de sensibilidade a contexto e localização como pode ser verificado nos trabalhos apresentados em (Satyanarayanan, 2001) e um passo adiante com o uso de web semântica para contextualização em (Moore, Hu and Wan, 2008).

Este trabalho tem como objetivo prover uma plataforma para melhorar a comunicação entre ambientes inteligentes e pessoas, utilizando dispositivos móveis como meio de interação e possibilitando a exposição de contextos pela internet para dispositivos inteligentes externos ao ambiente inteligente utilizando a infraestrutura existente de computação na nuvem e a capacidade de compartilhamento de conhecimento através de ontologias.



## 1.1 CONTEXTO E MOTIVAÇÃO

Segundo a Gartner, o uso de dispositivos móveis deve ultrapassar o uso de PC's no acesso a Web, e até 2015 será possível observar que 80% dos dispositivos móveis nos mercados maduros serão smartphones, sendo o mercado basicamente dividido entre Windows Phones, Apple iOS e Android. (Savitz, 2012)

Segundo a IDC, em 2015 o número de smartphones no mundo deve chegar a 804 milhões de dispositivos, e segundo a ARS técnica, o número de usuários que farão acesso a internet apenas por dispositivos móveis deve chegar a 788 milhões neste mesmo ano. (Lopez, 2012)

Analisando os números e posições de diversos analistas de mercado é possível verificar que a adoção de dispositivos móveis é uma tendência universal e que utilizar dispositivos móveis inteligentes de forma a captar informações e padrões de uso do usuário, e disponibilizar estas informações a seus dispositivos é uma forma promissora de explorar a infraestrutura de web e permitir a comunicação entre pessoas e máquinas.

Em (Paridel *et al.*, 2010) os autores descrevem alguns dos desafios ao projetar um *middleware* e os problemas das soluções atuais. Estes desafios e problemas estão diretamente relacionados a grande escala de dispositivos e eventos, cujo volume dificulta a escalabilidade e resposta do sistema, nesta proposta o uso da computação em nuvem visa resolver parte destes problemas.

A motivação deste trabalho é construir uma plataforma para internet das coisas utilizando dispositivos móveis como meio de capturar contexto e disponibilizar para agentes internos ao dispositivo, para agentes no ambiente inteligente, ou para agentes existentes em dispositivos inteligentes externos ao ambiente, com escalabilidade e disponibilidade.

## 1.2 QUESTÃO DE PESQUISA

A questão central deste trabalho é: Como permitir a interação de um dispositivo móvel com um ambiente inteligente e armazenar informações de contexto para expor mudanças de contexto para agentes e dispositivos externos com escalabilidade e disponibilidade?

### 1.3 OBJETIVO DA PESQUISA

Conforme já destacado anteriormente, este trabalho objetiva a construção de uma plataforma para a internet das coisas e é composto por um motor de execução de agentes sensíveis a contexto para dispositivos móveis e de um armazenamento de histórico de contextos com escalabilidade horizontal baseado em ontologias. Este objetivo é subdividido nos seguintes objetivos específicos:

- criar um modelo para representação de contexto baseado em padrões existentes;
- projetar e implementar uma plataforma para execução de agentes sensíveis a contexto de dispositivos móveis;
- projetar e implementar serviços web para armazenamento do contexto em ontologias;
- projetar e implementar serviços web para pesquisa dos dados nas ontologias;
- adaptar formas de armazenamento distribuído de triplas RDF contendo dados das ontologias modeladas;
- projetar e implementar agentes para alimentar o histórico de contextos dos dispositivos móveis.

### 1.4 METODOLOGIA

Para garantir que os objetivos descritos na seção anterior sejam atingidos uma metodologia para realização deste trabalho deve ser definida. Esta metodologia de trabalho pode ser descrita pelos seguintes itens:

- analisar trabalhos relacionados e alternativos a arquitetura proposta a partir do estudo de produções científicas da área;
- estabelecer a visão geral da arquitetura baseado no objeto de pesquisa deste trabalho;
- projetar e implementar o software indicados nos objetivos, utilizando as ferramentas e tecnologias selecionadas e detalhadas na seção de revisão de literatura;
- validar a arquitetura proposta em um cenário de exemplo de sensibilidade a contexto;

- documentar todo projeto e elaboração da arquitetura e solução, bem como resultados e análise comparativa com outras soluções.

## 1.5 ORGANIZAÇÃO DO DOCUMENTO

Esta proposta está dividida da seguinte forma:

Capítulo 2: Realiza uma revisão de literatura das tecnologias envolvidas neste trabalho, bem como as limitações e decisões de projeto realizadas.

Capítulo 3: Apresenta trabalhos relacionados com o desta proposta, suas principais diferenças e trabalhos que formaram a base para implementação desta proposta.

Capítulo 4: Descreve os detalhes dos aspectos de modelagem, de projeto e de implementação do trabalho proposto, bem como alguns detalhes específicos de estrutura de dados e de interface de comunicação entre módulos.

Capítulo 5: Descreve os detalhes dos resultados obtidos a partir da implementação discutida no capítulo anterior, bem como as validações realizadas sobre o trabalho.

Capítulo 6: Considerações finais e sugestões para trabalhos futuros.

## 2 ARCABOUÇO TEÓRICO

Nas próximas seções será realizada uma revisão de literatura nos assuntos diretamente relacionados a esta proposta. Na primeira seção é revisado o conceito de computação pervasiva, e qual de suas características será o foco desta proposta, seguido pelo problema da localização *indoor* de dispositivos e da descoberta automática de recursos e serviços em uma rede local.

Na segunda e terceira seção é tratado respectivamente o assunto de ontologias, que é proposto como forma de representação do conhecimento, e agentes de software, como forma de criar informações e inferir padrões a partir do conhecimento adquirido pelos sensores e agentes de software.

A quarta seção contém informações relacionadas à computação na nuvem, como forma de lidar com o volume de informações e alta disponibilidade necessária para esta proposta, evitando o desperdício de recursos sem uso.

A quinta seção trata das plataformas e linguagens utilizadas na construção do software, bem como o motivo da escolha de cada uma das tecnologias.

A sétima e última seção trata de como cada tecnologia citada na revisão de literatura será utilizada no software desta proposta.

### 2.1 COMPUTAÇÃO PERVASIVA

A visão de computação ubíqua conforme apresentada por (Weiser, 1991), e depois refinada para o termo computação pervasiva é essencialmente a ideia de ambientes inteligentes e integrados ao cotidiano de seus usuários, reduzindo assim o esforço para a realização das atividades e de comunicação entre objetos do dia-a-dia (Satyanarayanan, 2001).

Atualmente a computação pervasiva é uma evolução em relação a experimentos realizados em meados de 1970, e é a união das áreas de estudo de sistemas distribuídos e computação móvel, fazendo uso destas tecnologias para atingir seus principais objetivos, que conforme (Satyanarayanan, 2001) podem ser caracterizados como:

- *Espaços Inteligentes*. É definido como um espaço com tecnologia embarcada, que permite a sua comunicação com objetos dentro de seus limites, ou com outros espaços inteligentes.

- *Invisibilidade*. É a característica que é alcançada quando a tecnologia embarcada se torna invisível para a consciência dos usuários, permitindo que o uso e interação natural com o espaço inteligente impliquem em respostas naturais e inteligentes do ambiente.
- *Escalabilidade localizada*. É a capacidade dos espaços inteligentes fornecerem capacidade computacional para dispositivos com menos capacidade, seja de processamento, banda ou energia. Atualmente a capacidade dos dispositivos móveis tem sido limitada a quantidade de energia disponível, e essa característica da computação pervasiva conforme indicada por (Satyanarayanan, 2001) pode ser menos relevante quando comparada as demais.
- *Esconder as diferentes condições*. Característica na qual ambientes podem ter condições distintas de “inteligência”, o que pode afetar a característica de invisibilidade, uma vez que o usuário perceberia uma diferença entre ambientes. O autor sugere a possibilidade de ser compensada pelos dispositivos móveis inteligentes.

De todas as características, esta proposta irá se focar em como a plataforma proposta vai permitir a construção de espaços inteligentes, bem como a interação entre dispositivos.

### 2.1.1 Localização *indoor* em dispositivos móveis

Com o advento e expansão significativa do mercado de dispositivos móveis inteligentes na última década e a inclusão de uma série de tecnologias de localização (GPS) e comunicação via radiofrequência (802.11, Bluetooth e mais recentemente, NFC) nestes dispositivos abrem novas possibilidades quanto à solução de desafios típicos da computação pervasiva.

Em termos de localização em campo aberto o GPS é efetivo e suficientemente preciso e está presente na maior parte dos dispositivos móveis inteligentes modernos, e quando utilizado em conjunto com tecnologias de comunicação de longa distância (GSM, 3G ou WiMax) permite a determinação de contextos e utilização de serviços a partir da Internet.

No entanto, quando se trata de determinação de localização e contexto dentro de construções ou prédios (neste trabalho referido como *indoor*) os sinais de rádio dos satélites que compõem o GPS falham em adentrar estas construções, diminuindo significativamente a precisão, além do fato que o sistema de GPS permite a geo-localização, mas não consegue

determinar altitude e, portanto não consegue diferenciar andares em um prédio, e é para isso que é necessário uma forma alternativa e precisa para determinação de localização.

Em (Gu, Lo and Niemegeers, 2009) o autor realiza uma revisão das diversas propostas para localização indoor, e é possível destacar o uso de luz infravermelha (IR), ultrassom, RFID, rede local sem fio 802.11 (WLAN), Bluetooth, sinais magnéticos, análise de imagens e emissão sonora pouco abaixo do espectro audível.

A limitação nas diversas propostas é a disponibilidade da tecnologia em questão em dispositivos móveis, que atualmente é limitada a Bluetooth, IEEE 802.11 e em alguns casos NFC (baseado nas frequências do RFID). Atualmente as propostas de melhor precisão para localização indoor para dispositivos móveis são baseadas em Bluetooth, chegando a uma precisão de 1 metro, e com sensibilidade de posição vertical dada a natureza do posicionamento, conforme pode ser visto nos trabalhos de localização indoor utilizando Bluetooth por força de sinal, trilateração ou nomenclatura do ponto mais próximo (Altini *et al.*, 2010; Fischer, Dietrich and Winkler, 2004; Hallberg, Nilsson and Synnes, 2003; Hay and Harle, 2009; Zhou and Pollard, 2006).

Nesta proposta a localização geográfica será determinada utilizando a funcionalidade de busca pelo melhor provedor, existente na plataforma Android, que determina a localização a partir dos recursos disponíveis no dispositivo móvel (por exemplo, utilizar o GLONASS como alternativa de localização quando o sistema de GPS não estiver ativo), com apoio de redes IEEE 802.11 para melhor precisão. Sensores baseados em proximidade geográfica são disponibilizados na API para construção de agentes.

### 2.1.2 Sistemas orientados a contexto

A definição de sistemas orientados a contexto é dada como sistemas capazes de examinar e reagir a mudanças no contexto do usuário do sistema, sendo o contexto todas as informações relevantes ao usuário em um determinado momento do tempo, também levando em conta localização do usuário, bem como pessoas e dispositivos próximos ao usuário. (Schilit, Adams and Want, 1994)

Em (Brooks, 2003) o autor propõe cinco questões para guiar o projeto de dispositivos computacionais sensíveis a contexto, questões estas que quando respondidas proveem informações suficientes para inferência do contexto do usuário do sistema, estas cinco questões são: Quando, Onde, O Que, Quem e Por que.

Estas questões podem ser divididas em categorias de maturidade devido às tecnologias existentes e difundidas. “Quando” e “Onde” podem ser definidas com precisão aceitável e sem consumo de muitos recursos por dispositivos móveis com GPS e sincronização de data e hora via rede GSM.

“O que” está relacionado a o que o usuário está fazendo, e é classificado como algo ainda em desenvolvimento, e neste trabalho será definido em conjunto com outro projeto para captura das interações do usuário com o dispositivo móvel.

“Quem” é uma questão considerada desafiadora pelo autor, uma vez que não é apenas a classificação de nome e idade, mas sim a relação entre o usuário e outras pessoas a sua volta. Esta característica será parcialmente determinada pela conexão da conta em uma rede social permitindo ao sistema determinar relações e informações do usuário.

Por fim, a questão “Por que” é definida como possivelmente a mais complexa de todas, uma vez que a motivação por trás de uma ação pode ser desconhecida pelo próprio usuário, e é uma questão que ainda continua como um desafio a ser pesquisado em trabalhos futuros.

A definição semântica de um contexto neste trabalho utiliza as quatro principais questões (Quando, Onde, O que e Quem) como base do modelo de contexto, que será utilizado por agentes nesta arquitetura para determinar suas respostas sensíveis a contexto.

### 2.1.3 Descoberta de recursos e serviços

Conforme foi apontado por (Edwards, 2006), uma das visões da computação ubíqua é tornar os sistemas computacionais invisíveis, como uma ferramenta de trabalho, onde seu uso é realizado sem necessidade de um esforço de administração e configuração por parte do usuário. Para atingir tal efeito existe a necessidade de que sistemas e recursos tenham conhecimento de novos recursos, serviços ou dispositivos de forma espontânea e automática, e é nesse ponto que os protocolos para descoberta de recursos e serviços agregam como parte da infraestrutura para computação ubíqua.

Neste trabalho foram analisados os três protocolos mais disseminados para a descoberta de recursos e serviços em redes locais, e como podem ser estendidos para permitir a sensibilidade a contexto.

### 2.1.3.1 *Simple Service Discovery Protocol (SSDP) e Universal Plug and Play (UPnP)*

A *Universal Plug and Play (UPnP)* é uma tecnologia definida por um consórcio de empresas que faz uso de tecnologias web (como o HTTP, SOAP e XML) para prover acesso automático a recursos e serviços. (Edwards, 2006)

O *Simple Service Discovery Protocol (SSDP)* é o protocolo utilizado pelo UPnP para conseguir fazer a descoberta automática dos recursos e serviços disponíveis em uma determinada rede local, e reflete a visão centrada na web, uma vez que é construída em cima do HTTP, e identifica recursos a partir de URI's. (Edwards, 2006)

Como exemplo a UPnP consegue definir uma impressora multifuncional como um dispositivo, e as funcionalidades de impressão, cópia e fax como serviços do dispositivo em questão. (Edwards, 2006)

Devido ao fato de que o SSDP não faz uso de um diretório central para controle dos recursos, os anúncios de dispositivos e serviços são realizados via mensagens HTTPMU (HTTP através de um *multicast* em uma porta UDP) assim que um determinado dispositivo é conectado a rede, assim como a descoberta de recursos, que também é realizada via HTTPMU a partir de uma URI que representa o recurso desejado. (Edwards, 2006)

O anúncio de um determinado dispositivo segue uma estrutura comum, onde a mensagem HTTPMU contém a URI que identifica o tipo do recurso disponível, o ID e a URL que aponta para um documento XML que prove detalhes sobre o dispositivo em questão. (Edwards, 2006; Hu, Huang and Liao, 2007)

A arquitetura do UPnP define algumas outras funcionalidades além do anúncio do dispositivo, como a aquisição de endereço IP via DHCP, a forma como o dispositivo deve responder a solicitações de controle de outros dispositivos, o sistema de notificação de eventos para que outros dispositivos previamente registrados via o método HTTP SUBSCRIBE possam receber informações sobre mudanças de estado do dispositivo e por fim a camada de apresentação e configuração do dispositivo, que deve ser baseada em HTML. (Hu, Huang and Liao, 2007)

Neste trabalho o UPnP foi escolhido como forma de descoberta automática de agentes em uma rede local, permitindo a execução de agentes locais quando disponíveis, e para essa descoberta automática foi definido um serviço UPnP que pode ser implementado por qualquer dispositivo que tenha intenção de prover agentes em uma rede local.



### 2.1.3.2 *Service Location Protocol (SLP)*

O *Service Location Protocol (SLP)* é um protocolo de descoberta de serviços sugerido pela *Internet Engineering Task Force* projetado de forma escalável na descoberta de serviços e recursos em uma rede local ou no nível de toda uma rede empresarial, uma vez que a descoberta de serviços pode funcionar via *multicast* e, portanto atingir apenas a rede local, ou via um servidor como agente de diretório que fica responsável por centralizar as informações dos recursos e serviços disponíveis a partir do registro destes dispositivos (via descoberta *multicast* ou via configuração explícita). (Edwards, 2006)

No SLP os serviços disponíveis são nomeados a partir de URLs, portanto uma impressora em uma rede local receberia o endereço de *service:printer://host*, e uma impressora com um serviço especializado *lpr* receberia a identificação *service:printer:lpr://host*. (Edwards, 2006; Friday *et al.*, 2004)

Ao fazer o anúncio dos serviços disponíveis, o dispositivo envia à rede local a mensagem de registro contendo o seu tipo de serviço, a URL identificadora, e um ou mais atributos relevantes, sendo os atributos representados por pares de chave e valor. (Edwards, 2006)

Para realizar a descoberta do serviço, o dispositivo cliente pode realizar a busca a partir da URL do dispositivo (por exemplo, *service:printer:lpr*) ou a partir de atributos que ele deseja procurar. (Edwards, 2006)

O SLP provê mecanismos de busca bastante ricos em termos de funcionalidades e métodos de comparação de atributos, o que permite uma busca bastante refinada e mostra a forte inclinação do SLP para redes empresariais com muitos recursos em diversos níveis. (Edwards, 2006; Friday *et al.*, 2004)

Como resposta da descoberta de serviços o dispositivo cliente recebe as URLs dos serviços que se enquadram nos filtros da busca, o tempo de vida da informação para controle de *cache* do resultado da busca e informações para autenticação no recurso. (Edwards, 2006)

### 2.1.3.3 *Bonjour*

O Bonjour é um protocolo mantido pela Apple que teve sua origem no grupo Zeroconf da *Internet Engineering Task Force* cujo principal objetivo é permitir que dois dispositivos se comuniquem em uma rede sem sequer a necessidade de existência de servidores DHCP e DNS, e sem a necessidade de configuração por parte do usuário. (Edwards, 2006)

O protocolo combina *multicast DNS* (mDNS) com *DNS service discovery* (DNS-SD) para aproveitar os protocolos de Internet existentes na descoberta de recursos, e a partir de um novo domínio de topo (.local.) permite que hosts arbitrários resolvam nomes destes domínios ao invés de precisar de um servidor DNS para tal atividade. (Edwards, 2006)

DNS-SD é basicamente um conjunto de nomeação de serviços em forma de registros de DNS, que não define novas operações nem novos tipos de registro de DNS, e, portanto é completamente compatível com o protocolo existente. (Edwards, 2006)

Para procurar um serviço na rede local, o dispositivo cliente envia uma requisição por registros DNS de ponteiro (PTR) indicando o tipo de serviço desejado segundo o padrão *\_tipo.\_protocolo.dominio*, portanto para encontrar todos os servidores web em “unisinos.br” o cliente enviaria a requisição para *\_http.\_tcp.unisinos.br*, que basicamente significa a busca por servidores que sabem o protocolo HTTP, que rodam sobre o protocolo TCP e que fazem parte do domínio unisinos.br, que é o destino da requisição. (Edwards, 2006)

O registro PTR retornado pela requisição é um nome amigável para o recurso encontrado (e devido a este fato o Bonjour não consegue diferenciar dois recursos com o mesmo nome), e para recuperar a instância específica para os serviços do recurso em questão é necessário enviar outra requisição do tipo SVC (*DNS service*), que por sua vez retorna o host e a porta onde o serviço se encontra. (Edwards, 2006)

A partir do uso de padrões de Internet existentes e conhecidos o Bonjour consegue oferecer a descoberta de recursos em uma rede local com nenhum esforço de configuração por parte do usuário.

## 2.2 ONTOLOGIAS

No contexto da ciência da computação, uma ontologia é definida como uma série de primitivas utilizadas para modelar um domínio de conhecimento, tipicamente representadas por classes, atributos e relações. (Gruber, 2009)

Ontologias são geralmente definidas de forma a abstrair as estruturas de dados e estratégias de implementações, e por este motivo são ditas como parte do nível “semântico” da informação, ao invés das camadas lógicas ou físicas das bases de dados, e é esta característica independente do modelo físico que permite a ontologias conectar bases de dados heterogêneas de sistemas distintos, especificando as interfaces baseadas na representação do conhecimento. (Gruber, 2009)

Ontologias fazem parte dos padrões da W3C para a web semântica, na qual a mesma é utilizada para especificar um padrão de vocabulários conceituais para o compartilhamento de informações entre sistemas, prover serviços para inferência sobre bases de dados e para publicação de bases de conhecimento reutilizáveis. (Gruber, 2009)

A Figura 1 abaixo ilustra um fragmento de uma ontologia de vinho cedida pela W3C<sup>1</sup> como complemento ao guia que descreve a *Web Ontology Language*. Neste exemplo é possível compreender a natureza hierárquica das classes, todas referenciando o conceito básico *Thing*, e descrevendo as classes relevantes a este domínio de conhecimento.

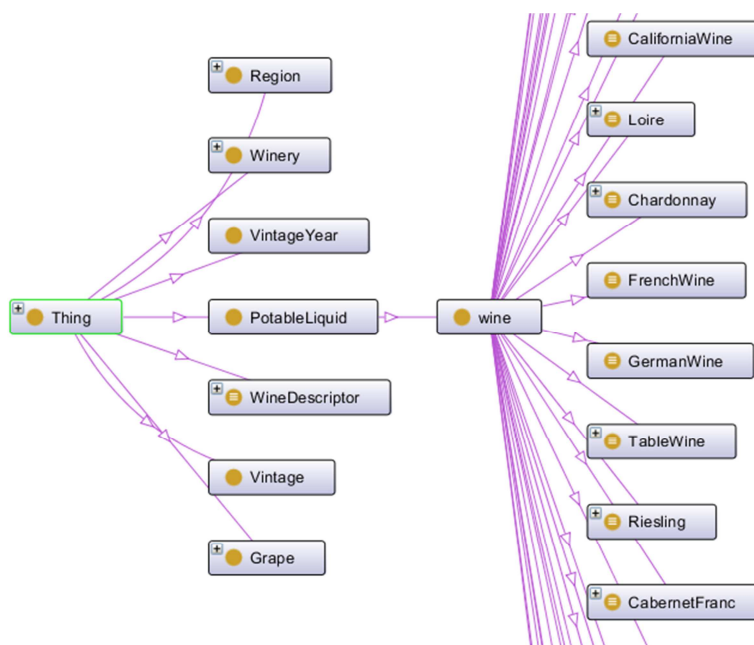


Figura 1 - Ontologia de vinho cedida pela W3C

### 2.2.1 RDF

O *Resource Description Framework* (RDF) é uma família de especificações da W3C projetada inicialmente para modelagem de meta dados para Web. A linguagem define um modelo simples de grafo para capturar a relação entre recursos usando um conceito conhecido como tripla, e tem por característica a flexibilidade de evolução do modelo sem necessidade da alteração de todas as aplicações que consomem as informações do modelo em questão. (W3C, 2004)

Uma tripla é uma expressão formada por sujeito, predicado e objeto, e um conjunto de triplas forma o conceito de grafo RDF. Como exemplo, uma forma de representar a frase “A

<sup>1</sup> O acesso ontologia de exemplo completa pode ser realizado em: <http://www.w3.org/TR/owl-guide/>

marca do smartphone é Apple” em uma tripla RDF seria criar um sujeito denotando “smartphone”, um predicado “tem a marca” e um objeto “Apple”.

Neste trabalho tanto o modelo quanto os dados das ontologias OWL são armazenados de forma distribuída em um banco de dados HBase como um grafo RDF, que só é possível devido ao o fato da OWL ser construída com base no RDF.

## 2.2.2 OWL

Conforme já destacado no capítulo anterior, ontologias fazem parte dos padrões da *World Wide Web Consortium* (W3C) para web semântica, e cuja recomendação para construção de modelos é a *Web Ontology Language* (OWL). (W3C, 2009)

Por herdar características do *Resource Description Framework* (RDF), a OWL é capaz de referenciar conceitos presentes em outras ontologias distribuídas entre sistemas distintos, e é esta característica que destaca o seu objetivo de impulsionar a adoção da web semântica. (Shadbolt, Hall and Berners-Lee, 2006)

A recomendação da W3C apresenta três sublinguagens da OWL com diferentes graus de expressividade e liberdade para construção de modelos (W3C, 2004):

- *OWL Lite*: suporta usuários necessitando primariamente de classificação hierárquica e restrições simples, e dada sua simplicidade provê um caminho de migração simplificado para diversas taxonomias.
- *OWL DL*: suporta usuários que necessitam o máximo poder de expressividade sem perder capacidade computacional, ou seja, todas as relações têm garantia de serem computáveis, e todas as computações tem garantia de terminar em tempo finito.
- *OWL Full*: provê maior liberdade de expressividade que a *OWL DL* em troca das garantias computacionais, e permite a uma ontologia aumentar os conceitos em um vocabulário pré-definido (RDF ou OWL).

Cada uma das sublinguagens é uma extensão do seu predecessor mais simples, de forma que as seguintes relações são válidas, mas não o inverso das mesmas (W3C, 2004):

- Toda ontologia ou conclusão OWL Lite válida é uma ontologia ou conclusão OWL DL válida;
- Toda ontologia ou conclusão OWL DL válida é uma ontologia ou conclusão OWL Full válida;

A OWL 2, evolução da Web Ontology Language, adicionou mais três sublinguagens (ou *Profiles*, como são chamados no documento), cada qual com restrições sintáticas com relação à especificação estrutural da OWL2, e com características um pouco mais restritivas que a OWL DL, com objetivo de trocar poder de expressividade em troca de benefícios computacionais (W3C, 2009):

- *OWL 2 EL*: permite algoritmos em tempo polinomial para todas as tarefas de raciocínio padrão, e é particularmente adequado a aplicações com necessidade de ontologias grandes onde é possível realizar a troca de expressividade por garantias de desempenho.
- *OWL 2 QL*: adequado a aplicações com ontologias pequenas e grandes volumes de dados instanciados, onde é útil o acesso aos dados via consultas relacionais.
- *OWL 2 RL*: semelhante ao OWL 2 QL, mas onde o grande volume de informações precisa de manipulação direta nas informações na forma de triplas RDF.

A OWL 2 possui retro compatibilidade com seu predecessor e, portanto, toda ontologia OWL 1 continua uma ontologia OWL 2 válida, com inferências idênticas em todos os casos práticos. (W3C, 2009)

A Figura 2 apresenta um fragmento de uma ontologia de família em OWL 2 em notação RDF/XML que é utilizada nos exemplos da W3C, e neste fragmento foram marcados os blocos contendo estruturas importantes na construção de uma ontologia.

```

<rdf:RDF xml:base="http://example.com/owl/families/"
  xmlns="http://example.com/owl/families/"
  xmlns:otherOnt="http://example.org/otherOntologies/families/"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Ontology rdf:about="http://example.com/owl/families">
    <owl:imports rdf:resource="http://example.org/otherOntologies/families.owl" />
  </owl:Ontology>

  <owl:ObjectProperty rdf:about="hasWife">
    <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
    <rdfs:domain rdf:resource="Man"/>
    <rdfs:range rdf:resource="Woman"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:about="hasParent">
    <owl:inverseOf rdf:resource="hasChild"/>
    <owl:propertyDisjointWith rdf:resource="hasSpouse"/>
  </owl:ObjectProperty>

  ...

  <owl:Class rdf:about="Woman">
    <rdfs:subClassOf rdf:resource="Person"/>
  </owl:Class>

  <owl:Class rdf:about="Mother">
    <rdfs:subClassOf rdf:resource="Woman"/>
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="Woman"/>
          <owl:Class rdf:about="Parent"/>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>

  ...

  <rdf:Description rdf:about="personAge">
    <owl:equivalentClass>
      <rdfs:Datatype>
        <owl:onDatatype rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
        <owl:withRestrictions rdf:parseType="Collection">
          <rdf:Description>
            <xsd:minInclusive rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</xsd:minInclusive>
          </rdf:Description>
          <rdf:Description>
            <xsd:maxInclusive rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">150</xsd:maxInclusive>
          </rdf:Description>
        </owl:withRestrictions>
      </rdfs:Datatype>
    </owl:equivalentClass>
  </rdf:Description>

  ...

  <Person rdf:about="Mary">
    <rdfs:type rdf:resource="Woman"/>
    <owl:sameAs rdf:resource="otherOnt:MaryBrown"/>
  </Person>

</rdf:RDF>

```

Figura 2 - Fragmento de ontologia de famílias OWL 2

O primeiro bloco possui a instrução de importação dos conceitos de uma ontologia em outro documento, o que permite a reutilização ou extensão destes conceitos.

O segundo bloco é o exemplo de uma propriedade de classe que implica na existência de outra propriedade, no primeiro caso a existência da propriedade *hasWife* (“Tem Esposa”) implica na existência da propriedade *hasSpouse* (“Tem Cônjuge”), e no segundo caso deste

bloco é ilustrado o exemplo onde uma propriedade pode ser derivada de outra apenas invertendo sua direção, ou seja, *hasParent* (“Tem Pai ou Mãe”) é o inverso de *hasChild* (“Tem Filho ou Filha”).

O terceiro bloco exemplifica a construção de novas classes, o uso da herança para construir classes mais especializadas, como o exemplo da construção da classe *Mother* (“Mãe”) como uma subclasse de *Woman* (“Mulher”) e equivalente a interseção das classes *Woman* (“Mãe”) e *Parent* (“Pai ou Mãe”).

O quarto bloco apresenta um exemplo de um tipo de dados chamado *personAge* (“Idade”) com restrições de mínimo e máximo de 0 a 150, permitindo a restrição dos valores deste atributo.

Por fim, o quinto bloco apresenta uma instanciação de pessoa (no caso, *Mary*), que é do tipo *Woman* (“Mulher”) e é a mesma pessoa representada na ontologia *otherOnt* pelo nome *MaryBrown*, fazendo uso da importação da ontologia apresentada no primeiro bloco.

As ontologias deste trabalho foram modeladas em OWL e possuem as informações relativas ao histórico de contexto de forma a permitir o reuso e compartilhamento de informação independente de plataforma.

### 2.2.3 SPARQL

SPARQL é uma linguagem de pesquisa semelhante à *Structured Query Language* (SQL) e um protocolo para acessar RDF projetada pela *W3C RDF Data Access Working Group* e considerada uma das tecnologias-chaves para a web semântica. Conforme apontado pelo diretor da W3C, tentar fazer uso da web semântica sem SPARQL é o mesmo que tentar usar bancos de dados relacionais sem SQL. (W3C, 2008)

Atualmente SPARQL é suportado pelo Jena através de um módulo chamado ARQ, cujas capacidades também incluem busca em texto livre, uma extensão da álgebra do SPARQL e queries em múltiplas fontes de dados. (Apache Software Foundation, 2012)

SPARQL é utilizado neste trabalho para a exposição dos dados distribuídos, cuja seleção é realizada a partir de um serviço utilizando o padrão arquitetura REST, onde a instrução SPARQL é executada e o retorno das informações são providas em formato RDF/XML ou RDF/JSON.

#### 2.2.4 Jena

Jena é um kit de ferramentas para desenvolvedores Java com capacidade de lidar com RDF, RDFS, RDFa, OWL e SPARQL, o kit também possui um motor de inferências capaz de lidar com semânticas RDF e OWL, e tem por objetivo prover uma implementação integrada das recomendações para web semântica da W3C. (Carroll, Dickinson and Dollin, 2004)

Jena foi originalmente construído pela Hewlett-Packard em 2000 como um projeto de código aberto. Em 2010 o projeto foi aceito pela Apache Software Foundation para incubação e graduou para um projeto efetivo em abril de 2012. (Apache Software Foundation, 2012)

A arquitetura do Jena é centralizada em torno da manipulação de triplas e grafos RDF, todos acessados pelo *RDF API*, conforme ilustrado na Figura 3. Esta API tem funcionalidades básicas para adicionar ou remover triplas ou encontrar triplas que se enquadram em um determinado padrão. De forma nativa a arquitetura (na figura ilustrada pelo *Store API*) permite a armazenagem das informações em memória, em base de dados SQL, ou em uma representação própria das triplas, sendo também possível estender a armazenagem de acordo com a necessidade da aplicação. (Apache Software Foundation, 2012)

Uma das principais funcionalidades de aplicações web semânticas é a característica de realizar inferências baseado nas regras semânticas do RDF, RDFS e OWL, e na arquitetura a *Inference API* é responsável por prover as ferramentas de inferências através do motor nativo, ou através do uso de um motor de inferências externo. (Apache Software Foundation, 2012)



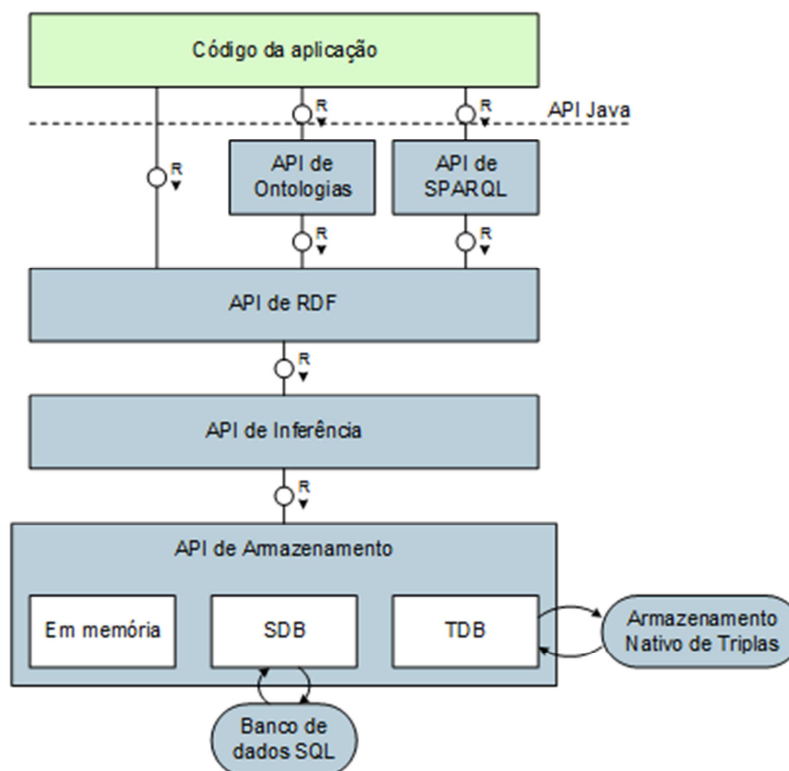


Figura 3 - Arquitetura do Jena

Conforme ilustrado na Figura 3 o SDB é uma parte da API de armazenamento do Jena e projetado especificamente pra suportar a linguagem SPARQL. No SDB uma base de dados relacional é utilizada como forma de armazenamento e todas as instruções SPARQL são traduzidas para SQL para então ser repassadas para o banco de dados relacional executar a instrução.

O SDB tem suporte a três formatos para persistência das ontologias, que diferem na forma como são criadas as tabelas e indexado os nós para os grafos. Essa característica de possuir diferentes formatos facilita a criação de novos formatos otimizados. Como exemplo de formato é possível destacar o formato simplificado, que consiste em apenas uma tabela de quatro colunas: sujeitos, predicados, objetos e grafos que contem a essa tripla.

Em (Khadilkar and Kantarcioglu, 2012) os autores adaptaram o código utilizado no SDB para traduzir as instruções SPARQL para o protocolo utilizado pelo HBase, e o resultado foi utilizado neste trabalho para permitir a persistência distribuída dos grafos RDF.

Jena é utilizado neste trabalho para a manipulação dos dados nas ontologias modeladas para o histórico de contexto, bem como para a exposição dos dados via a interface SPARQL.

### 2.3 AGENTES DE SOFTWARE

Conforme apontado por (Wooldridge, 1999), não existe uma definição universalmente aceita para o termo agente, mas é consenso o fato da autonomia ser o conceito central de um agente, e como ao menos uma definição para o termo é importante, o autor define agente como um sistema de computador situado em algum ambiente, e capaz de realizar ações autônomas neste ambiente com objetivo de atingir seus objetivos de projeto.

A Figura 4 ilustra de forma abstrata a relação entre agente e ambiente, onde ações geradas pelo agente afetam o ambiente, bem como a visão do agente em relação ao ambiente sendo realizada pelos sensores. Na maior parte dos domínios de conhecimento o agente não vai ter controle completo sobre o ambiente, apenas controle parcial no qual o agente é capaz de influenciar o ambiente através de ações. Do ponto de vista do agente isso significa que uma mesma ação realizada duas vezes pode ter resultados distintos, e pode falhar em ter o resultado desejado, e isso significa que o agente deve estar sempre preparado para lidar com as falhas. Formalmente este tipo de ambiente é chamado de estocástico. (Wooldridge, 1999)

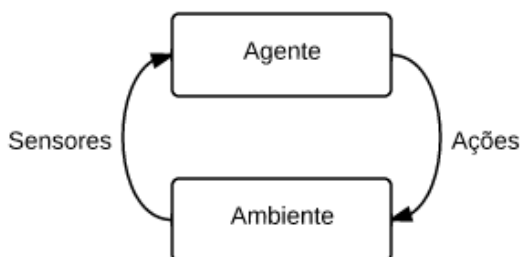


Figura 4 - Relação entre agente e ambiente

A efetividade das ações do agente com relação ao ambiente no qual está inserido é afetada por sua arquitetura a qual deve ser definida levando em consideração as diferentes propriedades do ambiente. A seguinte sugestão de classificação é proposta em (Russell and Norvig, 2009):

- *Completamente observável ou Parcialmente observável.* Um completamente observável é um ambiente onde o agente pode obter informações completas e atualizadas sobre o estado do ambiente. Quanto mais observável é o ambiente, mais simples é a construção do agente que irá operar no mesmo.
- *Determinístico ou Estocástico.* Ambientes determinísticos é o ambiente onde qualquer ação tem um efeito único no ambiente, ou seja, não existe incerteza no resultado da ação em relação ao ambiente.

- *Episódico ou Sequencial.* Em um ambiente episódico, a performance de um agente depende diretamente do número de episódios discretos, sem relação à performance do agente em diferentes cenários. Um exemplo de ambiente episódico é um sistema de ordenação de correspondências, onde não existe relação entre a ordenação atual e futuras ordenações.
- *Estático ou Dinâmico.* Um ambiente estático é um ambiente onde se pode assumir que continuará inalterado a menos que o agente realize alguma ação, e ambientes dinâmicos são ambientes onde outros processos operam de forma além do controle do agente.
- *Discreto ou Contínuo.* Em um ambiente discreto existe um número finito de ações e possibilidades, como por exemplo, um jogo de xadrez. No ambiente contínuo existem inúmeras possibilidades, como por exemplo, dirigir um carro de um ponto A até um ponto B.

A definição de agente pode ser derivada para definir o agente inteligente, que é caracterizado por (Wooldridge, 1999):

- *Reatividade.* Agentes inteligentes são capazes de perceber seu ambiente e responder às mudanças que ocorrem para cumprir seus objetivos de projeto.
- *Pró-atividade.* Agentes inteligentes são capazes de mostrar um comportamento orientado a seu objetivo tomando iniciativa própria para cumprir seus objetivos de projeto.
- *Habilidade social.* Agentes inteligentes são capazes de interagir com outros agentes (e possivelmente humanos) a fim de satisfazer seus objetivos de projeto.

Neste trabalho o conceito de agentes de software é aplicado na construção da API que permite o desenvolvimento de software recebendo informações dos sensores do dispositivo móvel e executando ações baseadas nessas informações.

### 2.3.1 JADE

O *Java Agent Development Framework* (JADE) é uma plataforma com objetivo de facilitar o desenvolvimento de agentes de software em conformidade com a especificação da *Foundation for Intelligent Physical Agents* (FIPA) para sistemas multi-agentes. (Bellifemine, Poggi and Rimassa, 1999)

Atualmente o JADE está na versão 4.2.0 e possui suporte a construção de uma plataforma multi-agente distribuída, mobilidade de agentes entre plataformas e suporte a dispositivos móveis (via J2ME e também na plataforma Android).

Para execução de agentes em uma máquina virtual Java o JADE necessita de uma série de serviços ativos e disponíveis que são providos por uma instância de execução do JADE chamada de container. Um grupo de containers é chamado de plataforma e provê uma camada homogênea que abstrai as camadas inferiores (sistema operacional, rede, tipo de máquina virtual Java) dos desenvolvedores de agentes. (Bellifemine, Poggi and Rimassa, 1999)

Neste trabalho o JADE foi avaliado como opção para criação de agentes sensíveis a contexto, mas substituído pelo projeto e implementação de uma API possível de reuso em diversas plataformas.

## 2.4 COMPUTAÇÃO NA NUVEM

O conceito de *cloud computing* (computação na nuvem) se refere a um grupo de máquinas arranjadas de forma que permita a disponibilização de recursos computacionais sob demanda, via Internet ou LAN. Este nome se deve ao fato do usuário não conseguir distinguir a localização ou organização dos equipamentos que fornecem os recursos computacionais, que depois de utilizados são liberados e devolvidos para a nuvem. (Antonopoulos and Gillam, 2010)

A expansão e utilização em massa da computação na nuvem se deve a união de tecnologias de virtualização (impulsionada pelo aumento na quantidade de núcleos disponíveis nos processadores), do uso do *MapReduce* para dividir e distribuir e paralelizar o processamento e da criação do conceito de *WebService* para padronizar a comunicação entre serviços na web independente de linguagem ou plataforma. (Antonopoulos and Gillam, 2010)

Atualmente as capacidades computacionais de diferentes nuvens são oferecidas em basicamente quatro categorias de serviços (Antonopoulos and Gillam, 2010; Rimal, Choi and Lumb, 2009):

- *Infrastructure as a Service (IaaS)*. Nesta modalidade a entrega dos recursos computacionais é entregue diretamente ao usuário, e possui alta flexibilidade e a possibilidade do pagamento geralmente baseado no tempo de uso. Nesta modalidade o usuário fica responsável por instalar, manter e executar seu

próprio software e pode ainda ser subdividida em nuvens públicas, nuvens privadas ou nuvens híbridas.

- *Hardware as a Service (HaaS)*. Modelo onde o vendedor de hardware permite aos clientes licenciarem o hardware diretamente, e em geral é vantajoso para grandes empresas, uma vez que não precisam mais se preocupar em construir e manter data centers.
- *Platform as a Service (PaaS)*. É o modelo onde o serviço de nuvem é vendido em forma de plataforma para desenvolvedores, sendo a forma mais fácil de desenvolver um software para nuvem. A criação e manutenção da infraestrutura demandam tempo e conhecimento que não necessariamente é de total conhecimento do desenvolvedor, e este modelo foi criado visando solucionar estes problemas. O desenvolvedor deve utilizar as API's disponibilizadas pelo provedor do serviço, que fica com a responsabilidade de manter a infraestrutura e prover ferramentas de escalabilidade para a aplicação em questão. Como exemplo de PaaS é possível listar Google AppEngine, Microsoft Azure, Heroku, Amazon BeanStalk e SAP Netweaver Cloud.
- *Software as a Service (SaaS)*. É a maneira como todo o benefício de uso da nuvem é oferecido diretamente ao usuário final da aplicação, quando o software é mantido pelo próprio desenvolvedor e o pagamento é proporcional ao uso do software. Exemplo de SaaS é o software disponibilizado pela Salesforce.com, NetSuite e SAP ByDesign.

Além das categorias de serviço ainda é possível separar nuvens em diferentes tipos, conforme indicado na categoria de IaaS (Antonopoulos and Gillam, 2010; Rimal, Choi and Lumb, 2009):

- *Nuvens privadas*. Onde dados e processos são gerenciados dentro de uma organização e rede privada. Exemplos de provedores de software para nuvens privadas são Amazon VPC, Eucalyptus, Enomaly e Intalio.
- *Nuvens públicas*. É a forma como a *cloud computing* é mais conhecida, onde o acesso aos recursos é disponibilizado ao público em geral através da Internet, de forma que o poder computacional pode ser compartilhado entre diversas aplicações sem nenhuma relação entre si. Exemplo deste tipo de nuvem é Amazon EC2, Rackspace, SymetriQ e Zimory.

- *Nuvens híbridas*. É o ambiente que consiste tanto de aplicações em nuvens privadas quanto públicas, e é utilizado de acordo com as necessidades específicas de cada organização.

Este trabalho faz uso do serviço de Infraestrutura como serviço (IaaS) da Amazon como forma de armazenamento distribuído e servidor de aplicação. Essa abordagem permite maior controle sobre a escalabilidade que pode ser classificada em duas categorias (Michael *et al.*, 2007):

- *Escalabilidade horizontal (scale out)*. É a abordagem onde novos computadores são adicionados a um cluster permitindo um maior poder de processamento, armazenamento e banda de forma distribuída, e vem se popularizando devido a redução de preços nas ofertas de infraestrutura como serviço. Em sistemas de arquivos distribuídos como o Apache Hadoop, a escalabilidade horizontal é utilizada como forma de aumentar tanto a tolerância a falhas quanto a capacidade de armazenamento dos *DataNodes* (descrito em mais detalhes no capítulo 2.5.3).
- *Escalabilidade vertical (scale up)*. É a forma tradicional de escalabilidade onde é realizada uma atualização no processador, memória ou disco rígido apenas em um servidor para aumento do desempenho, permitindo que o servidor execute mais processos em paralelos ou entregue as informações de forma mais rápida. Em sistemas distribuídos, a escalabilidade vertical é utilizada para aumentar a capacidade do servidor responsável pelo ponto de falha, quando existe, como o caso do *NameNode* no Apache Hadoop.

## 2.5 PLATAFORMAS, LINGUAGENS E ARMAZENAMENTO

A arquitetura deste trabalho tem forte relação com a plataforma Android e tecnologias web, portanto se faz necessário uma revisão das características da plataforma, bem como da linguagem script utilizada como base de desenvolvimento de aplicativos sensíveis a contexto.

### 2.5.1 Android

O sistema operacional Android é uma agregação de três sistemas principais, sendo a base composta pelo sistema operacional baseado em Linux, o *middleware* que permite a

construção de aplicativos utilizando Java e uma série de aplicativos-chave construídos sobre o *middleware* que permitem o funcionamento básico do dispositivo. (Saha, 2008)

Este *middleware* é conhecido como Dalvik, uma máquina virtual semelhante a Java *virtual machine*, mas diferenciada pelo formato dos arquivos compilados e pela forma de inicialização, uma vez que a Dalvik tem toda memória pré-alocada, que é simplesmente copiada para RAM toda vez que a Dalvik necessita iniciar, sendo assim inicializada com muito mais agilidade. (Dorokhova, Amelichev and Krinkin, 2010)

A abrangência de marcas e modelos do Android se deve a Open Handset Alliance, um consórcio de 78 empresas de hardware, software e telecomunicações, que trabalham juntas para definir os padrões deste sistema. (Open Handset Alliance, 2007)

De acordo com a Gartner, a plataforma Android tem 56,1% do mercado global de *smartphones* e até 2016 deve estar presente em 2.3 bilhões de computadores, *tablets* e *smartphones*. (Goasduff and Pettey, 2012; Virki, 2012)

Neste trabalho o uso do sistema operacional Android para desenvolvimento do cliente para sensibilidade de contexto se deve a sua liderança do mercado mundial de dispositivos móveis inteligentes, o código aberto da plataforma e grande suporte da comunidade para desenvolvimento de aplicações e a natureza do trabalho de interagir com outros projetos em desenvolvimento para a plataforma Android.

### 2.5.2 Javascript

Javascript é uma linguagem script baseada em herança prototipal, com tipagem dinâmica e suporte a múltiplos paradigmas, suportando estilos imperativos, funcionais e orientados a objetos. Inicialmente foi projetada como parte do Browser Netscape, foi posteriormente padronizada sob o nome de ECMAScript, atualmente encontrada em todos os principais Browsers modernos na versão 5, inclusive nas principais plataformas de smartphones, como iOS, Blackberry OS, Android e Windows Phones.

No Javascript existem apenas cinco tipos de dados: Booleanos, Números (que compreendem tanto Inteiros e Decimais), Strings, Array's e Objetos (representados utilizando a notação conhecida como JSON), sendo que funções herdam (via protótipo) do tipo primitivo Objeto, sendo um tipo especial chamado "*callable objects*".

Apesar as críticas e de alguns defeitos em seu projeto, sua simplicidade e facilidade de uso o tornaram o padrão para desenvolvimento web no lado cliente, e o fato de estar presente

em todas as plataformas de smartphones o transforma na melhor linguagem para a proposta deste trabalho.

Uma API para desenvolvimento de agentes sensíveis a contexto foi projetada e implementada para ser utilizada via Javascript, permitindo a interpretação de agentes em dispositivos móveis Android, mas permitindo a futura adaptação da API para outras plataformas.

### 2.5.3 Apache Hadoop

O Apache Hadoop é uma plataforma de código aberto construída em Java que permite o armazenamento e processamento de grandes conjuntos de dados de forma distribuída, e foi projetado para ter uma escalabilidade de alguns poucos para até milhares de servidores. É composto por quatro módulos principais sendo que entre estes é possível destacar o *Hadoop Distributed File System* (HDFS) e o *Hadoop MapReduce*.

O HDFS é um sistema de arquivos que permite o armazenamento distribuído de dados, sua arquitetura foi inspirada no *Google File System* e, portanto, herda a replicação de dados e tolerância a falhas implementadas em nível de aplicação, permitindo uma alta disponibilidade dos serviços executados sobre um cluster. A arquitetura do HDFS é *master/slave* e um *cluster* HDFS consiste em um único *NameNode*, que é o servidor *master* que gerencia o espaço de nomes no sistema de arquivos e regula o acesso aos arquivos em si, e de diversos *DataNodes* que gerenciam o armazenamento nos nós em que estão instalados. (Ghemawat, Gobioff and Leung, 2003)

Internamente o HDFS divide um arquivo em um ou mais blocos e distribui entre os *DataNodes*, que por sua vez realizam processos de replicação de dados entre si para garantir a disponibilidade das informações armazenadas. Periodicamente o *NameNode* recebe de cada *DataNode* uma mensagem de *Heartbeat* e relatório de blocos, conforme ilustrado na Figura 5. (Apache Software Foundation, 2012)

O *Heartbeat* é utilizado para detectar a indisponibilidade de um *DataNode* e é utilizada para evitar o envio de operações de entrada e saída para o nó indisponível. O relatório de blocos é uma lista de todos os blocos disponíveis no *DataNode* e permite ao *NameNode* detectar quais blocos precisam ser replicados. (Apache Software Foundation, 2012)



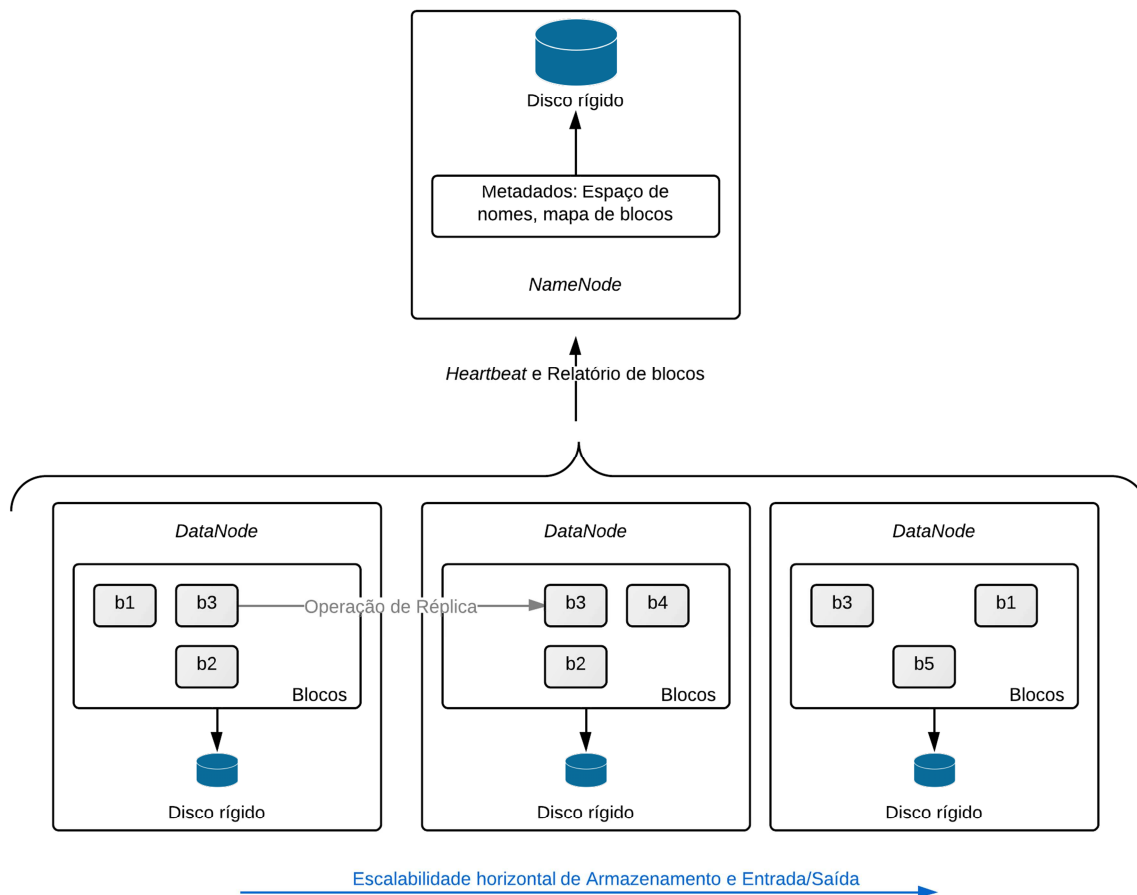


Figura 5 - Arquitetura do HDFS

Dois principais limitações desta forma de sistema de arquivos atingem o *NameNode*, a primeira limitação é que a falha no *NameNode* gera indisponibilidade e a necessidade de uma intervenção manual. A segunda limitação é de escalabilidade, já que apenas os *DataNodes* foram projetados para escalabilidade horizontal, mas para essa limitação já existe a opção chamada de federação HDFS, que permite compartilhar *DataNodes* entre *NameNodes* distintos.

A implementação *MapReduce* do Hadoop visa processar grandes quantidades de informações distribuídas de forma paralela, geralmente espalhada em um *cluster*. Para seu funcionamento é necessário instalar um *JobTracker* sob o nó *master* e uma série de *TaskTrackers* sob os nós *slaves*, e geralmente estão diretamente relacionados a instalação do HDFS garantindo que o nó que armazena a informação é o mesmo que vai processar parte dos dados.

Um processamento *MapReduce* é composto de duas fases principais e uma intermediária opcional: A fase de mapeamento (*Map*) na qual cada nó *slave* realiza uma parte do processamento sobre uma parte dos dados de entrada (geralmente com as informações que estão armazenadas no próprio nó), a fase opcional de combinação (*Combine*) onde é realizada

uma ordenação ou combinação da saída da função de mapeamento (o que permite a redução dos dados transferidos entre o mapeamento e a redução) e por fim a fase de redução (*Reduce*) onde um redutor ordena e agrega as informações de mesma chave e realiza algum processamento final antes de retornar os resultados desejados.

Nos módulos da implementação de *MapReduce* do Hadoop, a responsabilidade do módulo *JobTracker* é agendar um processamento *MapReduce* entre os nós *slaves*, enquanto o *TaskTracker* é responsável por executar as tarefas agendadas para o nó.

Neste trabalho o Hadoop é utilizado pela base de dados HBase como sistema de arquivos distribuídos.

#### 2.5.4 Apache HBase

O Apache HBase é um banco de dados de código aberto baseado no sistema de arquivos HDFS do Apache Hadoop e, portanto, objetiva o armazenamento informação de forma distribuída e em grande escala. Entre suas principais características está o modelo orientado a colunas e versões que é baseado no modelo do *Google Bigtable*. (Chang, Dean and Ghemawat, 2008)

Como o HBase é baseado no HDFS, ele herda a arquitetura *master/slave*, onde o único *master* do HBase coordena a distribuição dos dados entre diversos servidores regionais (*slaves*). Uma tabela HBase é composta de famílias de colunas e linhas, sendo que as informações podem estar distribuídas entre os diversos servidores regionais. De acordo com a definição da *BigTable* em (Chang, Dean and Ghemawat, 2008), essa tabela é definida internamente como um mapa ordenado cuja chave é a tripla: chave de linha, chave de coluna e um identificador de versão, cada valor dessa chave é uma lista de bytes.

Uma linha no HBase é definida por uma chave de linha, enquanto uma família de colunas é uma coleção de colunas que compartilham um mesmo prefixo, delimitado por dois pontos (“:”), portanto a coluna idade de uma família de colunas pessoa seria definida como “pessoa:idade”. A definição da família de colunas deve ser feito em tempo de criação do esquema do modelo, enquanto cada coluna pode ser definida a qualquer momento, inclusive ao se adicionar novas informações.

A Figura 6 é uma tradução da ilustração utilizada em (Chang, Dean and Ghemawat, 2008) para facilitar a visualização do armazenamento de uma informação num banco de dados baseado em *BigTable*. Na ilustração o modelo possui uma chave de linha definida como uma URL invertida, uma família de colunas “conteudo” armazena o conteúdo da página

e uma família de colunas “anchor” armazena o texto das âncoras HTML de outras páginas que referenciam a chave da linha. No exemplo a página da CNN é referenciada por duas páginas onde cada célula das âncoras possui apenas uma versão, e a célula de conteúdo possui duas versões representadas por  $t_5$  e  $t_6$ .

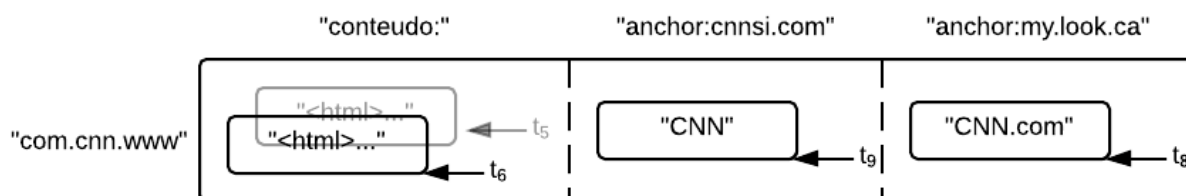


Figura 6 - Exemplo de tabela para armazenamento de páginas da Web, adaptado de (Chang, Dean and Ghemawat, 2008)

Neste trabalho o HBase é utilizado como forma de armazenamento distribuído de triplas RDF em conjunto com o Jena para manipulação das informações.

## 2.6 LINGUAGEM DE MODELAGEM


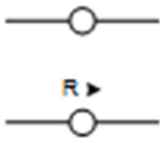
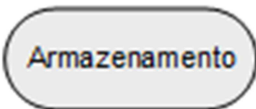
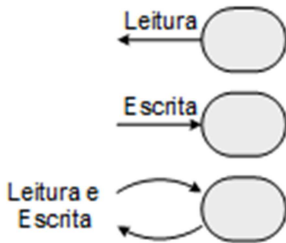
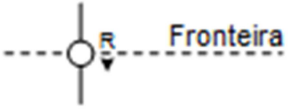
### 2.6.1 Linguagem de modelagem para arquitetura e implementação

Os modelos e diagramas relacionados com a arquitetura e implementação apresentados neste trabalho seguem o padrão de modelagem *Technical Architecture Modeling* (TAM), que é um padrão baseado nas especificações *Meta-Object Facility* (MOF) 2.0 e *Unified Modeling Language* (UML) 2.0 com objetivo de simplificar e reduzir a quantidade de diagramas estruturais e comportamentais necessários para modelagem da arquitetura técnica de um sistema. (SAP, 2007)

Dentre os diagramas e técnicas de modelagem disponíveis no TAM, foram escolhidos os dois principais diagramas para o projeto dos sistemas propostos. Para definição da arquitetura foi escolhido o diagrama de blocos e para ilustração do detalhamento da implementação foi escolhido o diagrama de classes.

O diagrama de blocos tem por objetivo descrever conceitualmente um sistema de informação, permitindo que o entendimento básico de um sistema seja disseminado de forma visual e facilitada. Este diagrama é composto basicamente por componentes ativos (Agentes), componentes passivos (Armazenamentos) e formas de comunicação (Acesso, Canal e Protocolo).

Tabela 1 - Componentes básicos do diagrama de blocos

Ilustração	Descrição
	<p>O componente de Agente corresponde a um elemento ativo e capaz de realizar uma determinada ação, de forma autônoma ou a partir de um estímulo. Podem ser compostos por outros agentes, armazenamentos e acessos internos.</p>
	<p>O componente de Canal é um elemento passivo que deve ser utilizado para ilustrar a comunicação entre agentes, podendo apresentar as operações realizadas no canal e o sentido do fluxo dos dados, e por consequência, a origem da requisição.</p>
	<p>O componente de Armazenamento é um elemento passivo onde um Agente pode realizar uma ação de leitura ou escrita, e é responsável pela retenção de dados de qualquer natureza. Um armazenamento pode estar contido em um Agente, em outro Armazenamento ou em componentes e subsistemas.</p>
	<p>Quando um elemento ativo (Agente) realiza uma ação de leitura ou escrita sobre um elemento passivo (Armazenamento), esta ação deve estar representada com o componente de Acesso.</p>
	<p>O componente de fronteira é representado por uma linha pontilhada e define o protocolo do componente de Canal que cruzar a linha pontilhada. Tem por objetivo facilitar o entendimento de protocolos e padrões de comunicação entre agentes.</p>

### 2.6.2 Padrão simplificado para representação gráfica de ontologias OWL

Os detalhes das ontologias OWL ilustradas neste trabalho seguem um padrão semelhante ao diagrama de classes da UML 2.0, mas adotando a seguinte nomenclatura de relacionamento entre classes:

- *Nome da ontologia.* É o endereço base da ontologia em questão.

- *Prefixos.* Representam os conceitos das ontologias referenciadas pelo modelo da ilustração. Os prefixos devem seguir o padrão de prefixos utilizados no SPARQL. O uso da notação [prefixo:Classe] só é utilizado quando o prefixo é explicitamente definido no espaço reservado para prefixos.
- *Nome da classe.* É o nome da classe representada pelo retângulo
- *Relações* <nome : tipo : referência>. É formada pela tripla ( $n : t : r$ ) onde  $n$  representa o nome do relacionamento,  $t$  representa o tipo de relacionamento (O quando é uma propriedade entre objetos ou D quando o é uma propriedade de relacionada a tipos de dados), e  $r$  que é o tipo da classe destino do relacionamento tipo O ou o tipo do primitivo em um relacionamento tipo D.

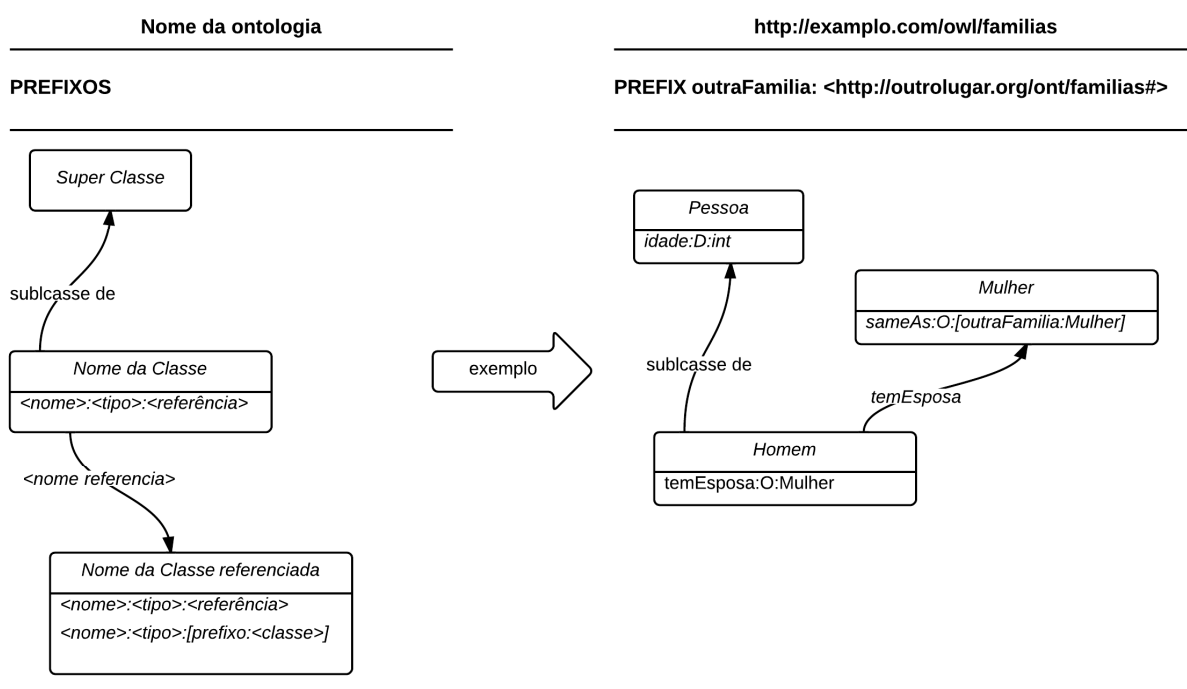


Figura 7 - Definições e exemplo de ontologia no modelo proposto

## 2.7 FECHAMENTO DO CAPÍTULO

Computação pervasiva apresenta características importantes que podem ser exploradas na construção de sistemas para a Internet das coisas, e é possível verificar que existem diversas opções para descoberta de serviços em rede local, bem como localização de

dispositivos, funcionalidades essenciais para o funcionamento de sistemas sem a necessidade da configuração do usuário, construindo a característica de invisibilidade.

A utilização de ontologias para modelagem e construção de um conhecimento reutilizável e independente de sistemas ou plataformas é outra característica importante não só para web semântica, mas como para a Internet das coisas, onde a habilidade de compartilhar o conhecimento entre diferentes dispositivos é essencial. A manipulação e tomada de ação em cima do conhecimento disponível a partir de agentes de software é outra característica importante de um sistema dinâmico e capaz de lidar com diversas situações em um mundo de comportamento não previsível (o ambiente estocástico).

A computação em nuvem disponibiliza grande capacidade computacional sob demanda a um custo muito baixo quando comparada a hospedagem tradicional, o que permite sistemas com alta escalabilidade (vertical e horizontal) e capacidade de liberar recursos quando os mesmos não são mais necessários.

A plataforma *Android* provê um público alvo de grande volume, bem como diversas ferramentas para facilitar o desenvolvimento de aplicativos e junto com a capacidade nativa de interpretar *Javascript*, uma linguagem multiplataforma devido à capacidade de interpretação por todos os navegadores nos principais sistemas operacionais de dispositivos móveis inteligentes, confere o poder tecnológico necessário para construção de sistemas capazes de captar o contexto, distribuir para diversos dispositivos, programar de forma unificada, e permitir a agentes a tomada de ações baseadas nestes contextos.

Por fim, dois projetos de código aberto da fundação Apache, Hadoop e HBase, se destacam tanto pelo uso dessas plataformas por grandes nomes da indústria de desenvolvimento de *software* quanto pela capacidade de escalabilidade, distribuição e alta performance.

### 3 TRABALHOS RELACIONADOS

A seleção de trabalhos relacionados descritos neste capítulo pode ser subdividida em duas categorias, onde na primeira categoria se encaixam os trabalhos que visam resolver o mesmo problema com abordagens similares, e na segunda categoria se encaixam os trabalhos que foram utilizados e modificados para se adequar a necessidade deste trabalho.

A primeira categoria de trabalhos visa validar o tema desta pesquisa bem como avaliar a existência de sistemas semelhantes, foi realizada uma busca em bases de publicações nas áreas relacionadas à ciência da computação. Considerando a natureza multidisciplinar desta proposta, diversos trabalhos aplicados em diferentes situações foram encontrados, dos quais foram selecionados os mais próximos ao tema de plataforma ou *middleware* para internet das coisas e cuja aplicação fosse utilizando recursos de agentes de software e web semântica ou relacionada a sistemas orientados a contexto. Esta categoria compreende os trabalhos descritos nas seções 3.1, 3.2, 3.3, 3.4 e 3.5, e comparados com este trabalho no capítulo 3.6.

A segunda categoria são trabalhos que não são apresentados no comparativo uma vez que não buscam solucionar o problema abordado, mas cuja contribuição foi importante para atingir objetivos específicos de forma mais eficiente e para construção de conhecimento a partir de outras pesquisas existentes. Nesta categoria se encontram os trabalhos descritos nas seções 3.7, 3.8 e 3.9.

#### 3.1 EASYMEETING E CONTEXT BROKER ARCHITECTURE

Conforme descrito em (Chen, Finin and Joshi, 2004), o sistema *EasyMeeting* foi um sistema de computação pervasiva construído na *University of Maryland, Baltimore County* (UMBC) com base em um sistema de computação pervasiva anterior chamado *Vigil* com objetivo de auxiliar apresentadores e plateias durante reuniões em espaços inteligentes, e adicionou capacidades de sensibilidade a contexto utilizando o *Context Broker Architecture* (Cobra). Este sistema foi implementado para demonstrar alguns dos aspectos da tese de doutorado relacionado ao Cobra, e mais especificamente as características de sensibilidade a contexto. (Chen, 2004)

O autor aponta que o Cobra é uma arquitetura de middleware que se distingue das demais arquiteturas por fazer uso de ontologias OWL para suportar modelagem de contextos e compartilhamento de conhecimento em ambientes inteligentes. Conforme indicado pelo

autor, um protótipo da arquitetura foi construído utilizando o padrão FIPA através da plataforma JADE. (Chen, 2004)

No EasyMeeting, o Cobra é responsável por capturar diversas informações de contexto dos sensores em um ambiente inteligente e compartilhar estas informações com o sistema de reuniões do Vigil, e a partir destas informações o Vigil é capaz de tocar áudio de mensagens personalizadas com assim que a audiência entra na sala e auxiliar os apresentadores a exibir a apresentação nos projetores.

Na Figura 8 está ilustrada a arquitetura do *EasyMeeting* e o papel do *Context broker* neste cenário de espaços inteligentes, onde o *Context broker* compartilha seu conhecimento de contexto com o agente *MajorDemo*, que a partir deste conhecimento invoca os serviços apropriados da infraestrutura provida pelo Vigil. (Chen, Finin and Joshi, 2004)

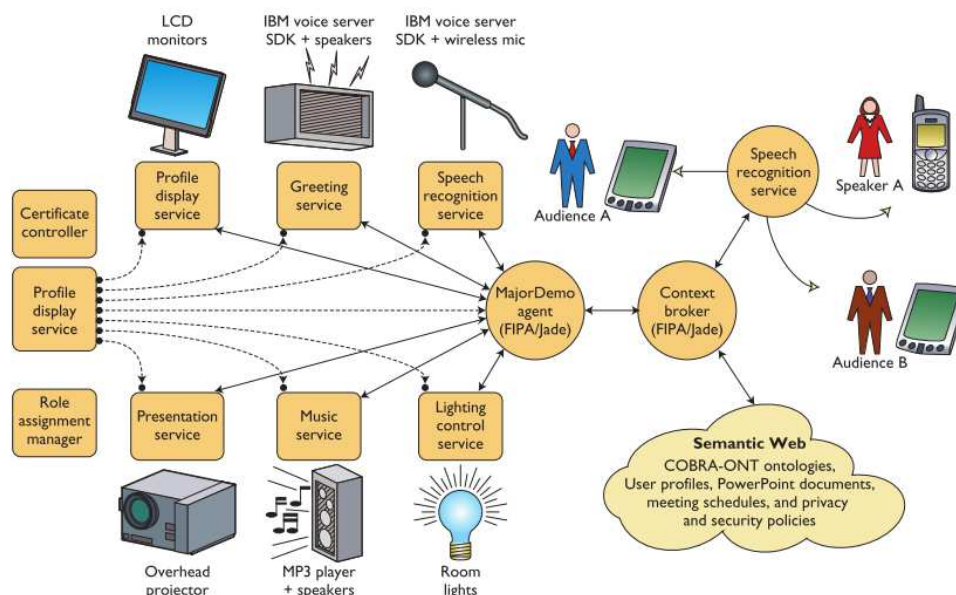


Figura 8 - Arquitetura do *EasyMeeting*, fonte (Chen, Finin and Joshi, 2004)

### 3.2 MIDDLEWARE SEMÂNTICO PARA INTERNET DAS COISAS

Neste trabalho os autores propõem o uso de ontologias como camada de interoperabilidade para aplicativos, implementando um middleware para melhorar a configuração automática de redes heterogêneas. A proposta do trabalho é mapear cada dispositivo na rede a um serviço semântico e associar cada serviço em pelo menos uma descrição semântica em uma ontologia OWL-S (ontologia com objetivo de descrever um *WebService*) de forma automática. (Song, Alvaro and Masuoka, 2012)



Como pode ser observado na Figura 9, o componente de *Discovery Engine* no *middleware* proposto é responsável por localizar os dispositivos em uma rede a partir de descoberta de serviços baseadas em UPNP e Bluetooth (ou outro padrão de descoberta de serviços), e após a descoberta este componente realiza o mapeamento destes dispositivos em forma semântica, disponibilizando estas funcionalidades a uma camada de apresentação de um ambiente de computação baseado em tarefas, escondendo do usuário a complexidade de lidar com diferentes formatos e padrões, apresentando apenas a forma semântica de cada dispositivo. (Song, Alvaro and Masuoka, 2012)

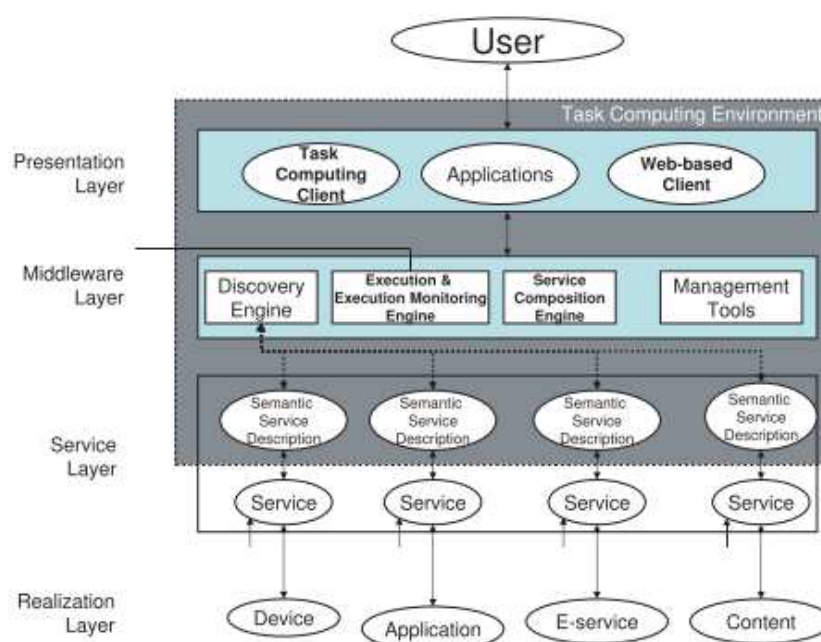


Figura 9 - Arquitetura do sistema e o papel do middleware, fonte (Song, Alvaro and Masuoka, 2012)

### 3.3 MIDDLEWARE SEMÂNTICO INTELIGENTE PARA INTERNET DAS COISAS

Neste artigo os autores propõem um grupo de ferramentas para desenvolvimento de agentes e adaptadores em conjunto com um ambiente de execução chamado de UBIWARE. A intenção dos autores é aplicar conceitos de descoberta automática, composição, orquestração, integração, monitoramento de execução, sensibilidade a contexto e demais conceitos relacionados com a web semântica. (Katasonov, Kaykova and Khriyenko, 2008)

De acordo com os autores, esta proposta de middleware deve ser baseada em um trabalho anterior utilizando RDF para descrever comportamentos e na implementação de comportamentos atômicos reutilizáveis, cujo objetivo é o reuso de atividades simples por

agentes que podem ser escolhidos para executar o comportamento em questão. (Katasonov, Kaykova and Khriyenko, 2008)

Ainda neste artigo os autores descrevem alguns casos de uso industriais para a visão proposta proveem os desafios e necessidades atuais da indústria com relação a sistemas complexos com autogerenciamento.

### 3.4 MIDDLEWARE PARA AGENTES SENSÍVEIS A CONTEXTO EM AMBIENTES DE COMPUTAÇÃO UBÍQUA

Neste trabalho os autores propõem um middleware para promover sensibilidade a contexto entre agentes em ambientes de computação ubíqua integrado ao Gaia, projeto com objetivo de tornar espaços físicos em ambientes inteligentes. (Ranganathan and Campbell, 2003)

No Gaia a infraestrutura relacionada à sensibilidade a contexto é caracterizada por seis módulos principais, ilustrados na Figura 10 (Ranganathan and Campbell, 2003):

- *Context Provider*. Módulo contendo sensores e outras fontes de dados para informações relativas ao contexto, o que permite a outros agentes (representado pelo *Context Consumer*) a requisitar ao módulo informações do contexto ou se registrar para receber notificações nas alterações dos dados do contexto.
- *Context Synthesizer*. Módulo responsável por deduzir informações de contexto baseado nos dados providos pelos *Context Providers*, como exemplo, utilizar informações com relação ao número de pessoas em uma sala e os aplicativos em execução para inferir a atividade da sala.
- *Context Consumer*. São as aplicações sensíveis a contexto, ou agentes que utilizam informações de *Context Synthesizer* e de *Context Provider* para tomada de ações ou adaptação de seu comportamento de acordo com o contexto inferido.
- *Context Provider Lookup Service*. Serviço único no ambiente de computação ubíqua que permite a agentes encontrar o *Context Provider* apropriado para sua respectiva necessidade. *Context Providers* devem ser registrar no *Lookup Service* para que sejam encontrados.

- *Context History Service*. Serviço que permite a agentes realizarem buscas com relação a contextos antigos, todos armazenados por este serviço, que assim como o *Lookup Service*, é único no ambiente de computação ubíqua.
- *Ontology Server*. Servidor que mantém ontologias que descrevem os tipos de informações contextuais, e é único no ambiente de computação ubíqua. Tem interface para adição de novas definições para tipos de contexto e a descrição dos conceitos utilizados pelos agentes no sistema.

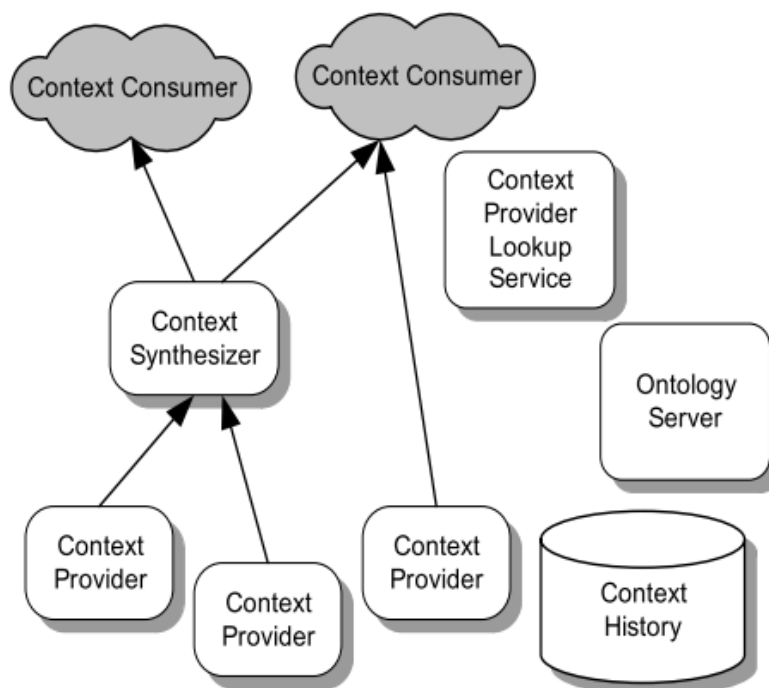


Figura 10 - Infraestrutura de contexto do Gaia, fonte (Ranganathan and Campbell, 2003)

### 3.5 UM MIDDLEWARE PARA AMBIENTES INTELIGENTES E INTERNET DAS COISAS

Neste trabalho os autores analisam a possibilidade de utilizar um middleware voltado à área de robótica, chamado de ROS, e aplicar a sistemas distribuídos, heterogêneos, baseados em sensores e inseridos em ambientes inteligentes. (Roalter, Kranz and Andreas, 2010)

A abordagem proposta é bastante interessante uma vez que é um ambiente inteligente é comparado pelos autores a um robô imóvel, com a diferença que seus sensores são orientados a utilizar tanto dados físicos de luz, temperatura, presença, abertura de portas, janelas e gavetas, quanto dados virtuais de redes sociais, mensagens instantâneas, condições de trânsito

e clima, ou sensores físicos com dados disponibilizados pela web. (Roalter, Kranz and Andreas, 2010)

Os autores também desenvolveram uma forma de acessar o contexto capturado pelo middleware através de *publish/subscribe*, permitindo o uso das informações por aplicações externas, e concluíram o experimento indicando uma percepção bastante positiva com relação ao uso deste middleware específico para construção de sistemas para ambientes inteligentes.

### 3.6 COMPARATIVO ENTRE TRABALHOS RELACIONADOS

Os trabalhos similares apresentados nesta primeira parte deste capítulo foram comparados em relação ao uso de tecnologias relevantes a esta proposta, agrupados pelos tópicos específicos de ontologias, agentes, ambientes inteligentes, dispositivos móveis e tecnologias web. Para comparação de ontologias foi determinado onde a mesma foi empregada, e foram identificados basicamente dois pontos, a modelagem de contexto ou a modelagem de dispositivos. Devido à variedade de cenários e tecnologias para desenvolvimento de agentes, sua comparação foi limitada a existência ou não de agentes responderem a mudanças no contexto.

Na maioria dos trabalhos foi identificado o suporte a ambientes inteligentes, geralmente através de outro projeto maduro, e o uso de dispositivos móveis para determinação de contexto foi menos utilizado do que o esperado.

Por fim foram identificados quais trabalhos faziam uso de tecnologias web, mais especificamente para acesso externo ao contexto de um determinado usuário e com relação ao uso das capacidades de *cloud computing* para escalabilidade.

A Tabela 2 apresenta um quadro comparativo dos trabalhos relacionados a esta proposta compilando algumas das conclusões descritas neste capítulo. Quadros com o valor “Sim” indicam que o critério foi atendido, e quadros com o critério “Não” indicam que o trabalho não apresenta informações relativas ao critério indicado.

Tabela 2 - Comparativo entre trabalhos relacionados e trabalho proposto

Critérios		Proposta	T1	T2	T3	T4	T5
Ontologias	Modela contexto em ontologias	Sim	Sim	Não	Sim	Não	Não
	Modelam dispositivos em ontologias	Sim	Não	Sim	Não	Sim	Não
Agentes	Agentes tem capacidade de responder a mudanças no contexto	Sim	Sim	Não	Sim	Sim	Sim
Ambientes inteligentes	Suporte a ambientes inteligentes	Sim	Sim	Não	Não	Sim	Sim
Dispositivos móveis	Emprego de dispositivos móveis para determinação de contexto	Sim	Sim	Não	Não	Não	Não
Tecnologias web	Uso de cloud computing	Sim	Não	Não	Não	Não	Não
	Exposição de contexto via tecnologia web	Sim	Não	Não	Não	Sim	Sim

LEGENDA.

T1 - EasyMeeting e Context broker architecture (CHEN; FININ; JOSHI, 2004) e (CHEN, 2004)

T2 - Middleware semântico para internet das coisas (SONG; CÁRDENAS; MASUOKA, 2010)

T3 - Middleware semântico inteligente para internet das coisas (KATASONOV; KAYKOVA; KHRIYENKO, 2008)

T4 - Middleware para agentes sensíveis a contexto em ambientes de computação ubíqua (RANGANATHAN; CAMPBELL, 2003)

T5 - Um middleware para ambientes inteligentes e internet das coisas (ROALTER; KRANZ; ANDREAS, 2010)

### 3.7 UMA FORMA DE ARMAZENAMENTO DE TRIPLAS RDF DISTRIBUIDA, ESCALAVEL E EFICIENTE BASEADA EM JENA E HBASE

Em (Khadilkar and Kantarcioglu, 2012) os autores abordam o problema de escalabilidade no armazenamento de dados RDF em apenas um servidor, sendo este problema especialmente evidente ao se utilizar as formas de armazenamento existentes no kit de ferramentas Jena, portanto o foco do trabalho foi construir uma plataforma para armazenamento distribuído da tripla sujeito, predicado e objetos que compõe o RDF.

Para solucionar este problema foi adaptado do conector SDB do Jena (descrito em mais detalhes na seção 2.2.4) para realizar as operações de persistência no banco de dados HBase, conferindo escalabilidade e distribuição das informações.

Os autores apontaram que a motivação para escolha do HBase foram duas características: (i) o fato de ser orientado a colunas, e que este comportamento geralmente tem

melhor performance que armazenamentos baseados em linhas (Abadi *et al.*, 2007), e (ii) que apesar do HBase suportar o uso de *MapReduce* devido a plataforma Hadoop seu uso não é obrigatório, e isso permite que todo suporte RDF do Jena fosse também aplicado aos dados armazenados no HBase, ao invés de ficar limitado a uma implementação de um motor de busca e inferência RDF baseado em *MapReduce*, que nos modelos atuais não possui suporte completo para a especificação do RDF. (Khadilkar and Kantarcioglu, 2012)

Dentre as vantagens desta abordagem é possível destacar a distribuição e tolerância a falhas conferida pelo banco de dados utilizado, e a possibilidade futura de realizar pesquisas e inferências sobre os dados utilizando motores de inferência baseados em *MapReduce*, o que possibilitaria uma forma otimizada de operações para análise das informações.

Neste trabalho foi utilizada uma abordagem de armazenamento utilizando HBase como base de dados para as triplas RDF, utilizando como base a formatação “Simples” proposta em (Khadilkar and Kantarcioglu, 2012), onde os dados de cada grafo RDF são armazenados em três tabelas indexando sujeitos, predicados e objetos.

Nessa forma de armazenamento cada tabela possui apenas uma família de coluna chamada de “triplas”, e para cada nó utilizado para indexar a tabela uma nova coluna é criada na família de coluna “triplas” para cada tripla que contém o nó indexador. Por exemplo, para um dado sujeito  $s_1$  na tabela de sujeitos, uma nova coluna será criada para cada tripla que contenha  $s_1$  em um dado grafo RDF. Esta nova coluna terá o nome composto pelo par *predicado-objeto*  $p_i-o_i$ , onde  $p_i$  e  $o_i$  é respectivamente o predicado e o objeto que fazem parte da tripla  $(s_1, p_i, o_i)$ .

Neste formato simples a nomenclatura das tabelas é composta por três partes seguindo a regra:

1. “esquema” é o nome do esquema a qual a informação pertence, e é determinado ao instanciar uma classe de descrição de armazenamento;
2. “grafo” é o nome do grafo RDF a qual a tabela pertence, e recebe o valor “tbl” quando o valor referencia o grafo raiz, o qual foi carregado na raiz do grafo RDF;
3. “índice” que representa a chave da linha indexada na tabela correspondente do formato “Simples”

A Figura 11 ilustra tanto o modelo de armazenamento de grafos RDF no formato simples quanto um exemplo composto pelo sujeito “agents:Individuo”, o predicado “agents:facebookName” e o objeto “exemplo”.

Esta forma de persistência de triplas RDF não é muito eficiente em termos de utilização de espaço em disco, mas acelera as operações de busca, já que não requer pesquisa em múltiplas tabelas, mas apenas uma busca na tabela indexadora correspondente.

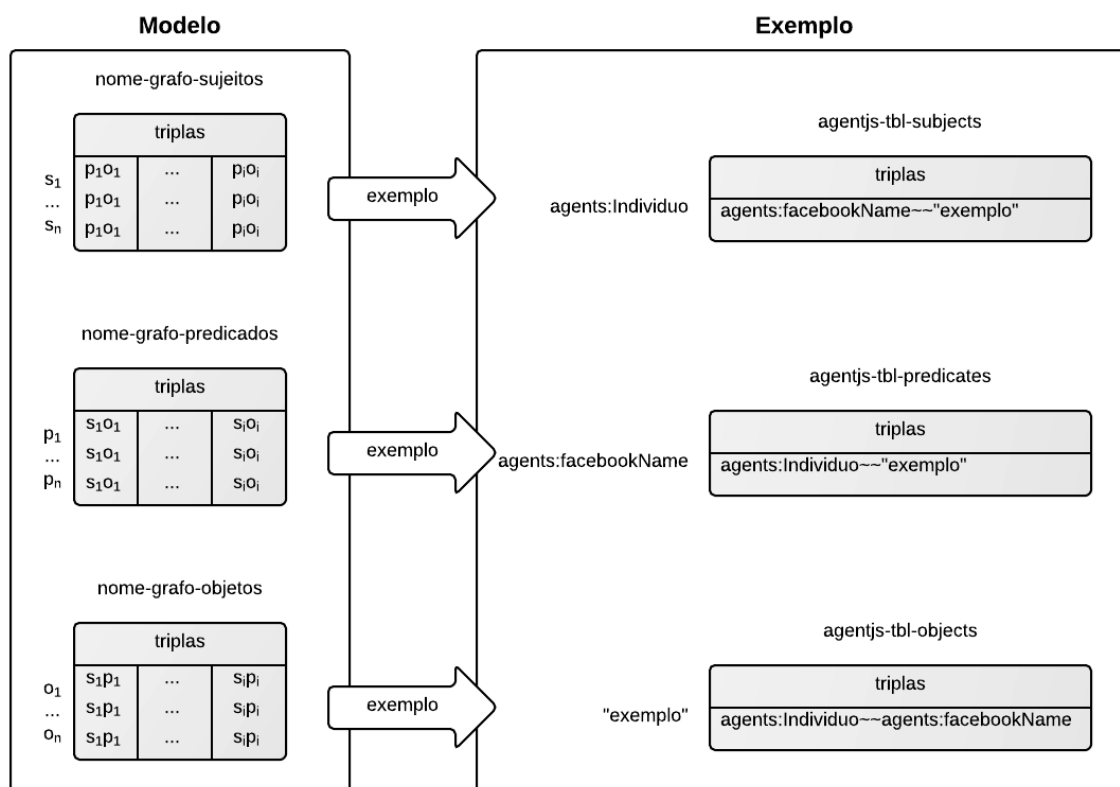


Figura 11 - Modelo e exemplo de armazenamento de triplas RDF no HBase, adaptado de (Khadilkar and Kantarcioglu, 2012)

### 3.8 A ONTOLOGIA SOUPA PARA COMPUTAÇÃO PERVASIVA

Em (Chen, Finin and Joshi, 2005) os autores descrevem uma ontologia chamada de Standard Ontology of Ubiquitous and Pervasive Applications (SOUPA) que faz uso da OWL e de uma estrutura de vocabulários modular para representar agentes, tempo, espaço, eventos, ações e conceitos de crença, desejo e intenção para permitir uma modelagem *Belief-Desire-Intention* (BDI) para agentes inteligentes.

Esta ontologia é composta pelo chamado SOUPA-core, ou seja, conceitos básicos distribuídos em nove documentos OWL, e o SOUPA-extension, que proveem um conjunto de ontologias para cenários de computação pervasiva específicos (como por exemplo, cenários relacionados a reuniões e agendamentos).

Em (Chen, 2004) o autor utiliza a SOUPA como base para construção da ontologia utilizada para intermediar a comunicação em um ambiente de computação pervasiva, e foi

essa mesma abordagem que foi utilizada durante a construção da ontologia para armazenamento do histórico de contextos.

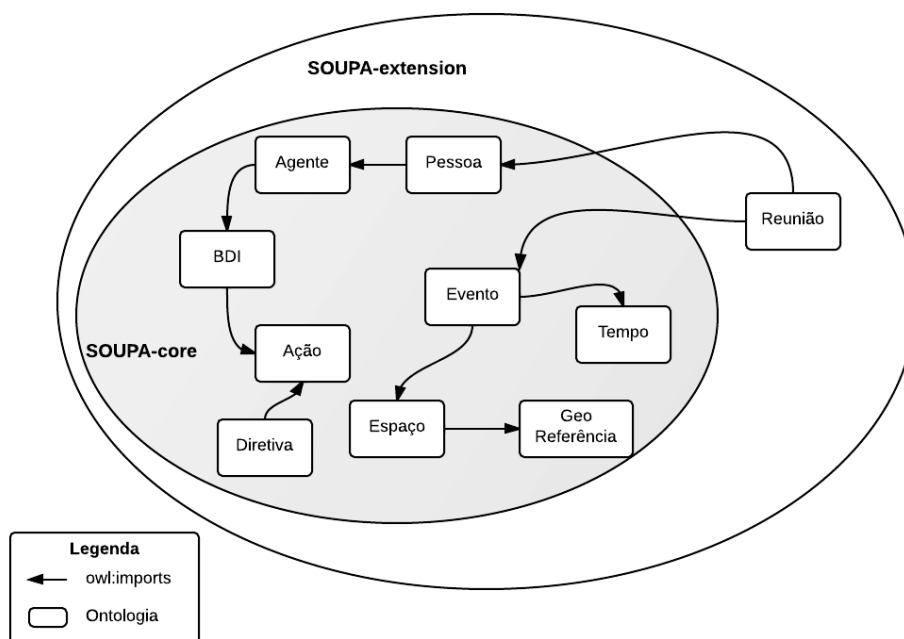


Figura 12 – Relacionamento entre ontologias na SOUPA-core e parte da SOUPA-extension, adaptado de (Chen, Finin and Joshi, 2005)

A Figura 12 ilustra o relacionamento entre ontologias da SOUPA, cuja abordagem modular foi utilizada neste trabalho, onde um novo conjunto de ontologias modulares foi criado com base nos conceitos definidos na SOUPA, mas sem efetivamente importar via OWL os conceitos existentes da SOUPA devido à indisponibilidade do recurso web que descreve a SOUPA, que de acordo com os autores deveria estar disponível em <http://pervasive.semanticweb.org>.

### 3.9 APLICATIVO MICROSOFT ON{X}

Em 2012 a Microsoft lançou um aplicativo para a plataforma *Android* que tem por objetivo permitir aos desenvolvedores a criação de “receitas” em *Javascript* para controlar algumas funcionalidades do dispositivo. (Microsoft, 2012)

O trabalho proposto compartilha algumas dessas características com o aplicativo da Microsoft, mas com objetivos distintos. O *on{x}* visa à automatização de algumas atividades diárias a partir de “receitas” prontas ou criadas pelo usuário, e a proposta deste trabalho é receber agentes tanto criados pelo usuário do sistema como de diferentes dispositivos disponíveis em uma rede local a partir de descoberta automática de agentes.



Outra diferença que vale destaque é que o código fonte deste trabalho é aberto<sup>2</sup>, enquanto o aplicativo criado pela Microsoft mantém seu código fonte fechado.

---

<sup>2</sup> Código fonte disponível em <http://github.com/insidy/agentjs>

## 4 PROPOSTA DE TRABALHO

Esta proposta de trabalho está inserida em um programa composto por três projetos em paralelo, e a contextualização dos diferentes projetos bem como seus objetivos será realizada na primeira seção, antes da descrição em detalhes das características específicas que compõem este trabalho, realizada nas seções subsequentes.

### 4.1 PROGRAMA DESKTOP SEMÂNTICO

O programa de *desktop semântico* está sob a coordenação do Prof. Dr. Sérgio Crespo e está em desenvolvimento a partir de três projetos de mestrado do PIPCA. O objetivo principal deste programa é criar um sistema extensível capaz de identificar padrões de uso e de contexto para realizar recomendações específicas ao usuário de acordo com o contexto em que ele se encontra. É feito o uso do termo “semântico” ao programa porque tem como premissa o uso das tecnologias da web semântica para permitir a extensibilidade e reuso dos componentes, e o “desktop” uma vez que o objetivo primário é a adaptação da tela inicial de um *smartphone* Android baseado nas recomendações. A Figura 13 ilustra a visão inicial do programa, bem como delimita o escopo inicial de cada um dos três trabalhos.

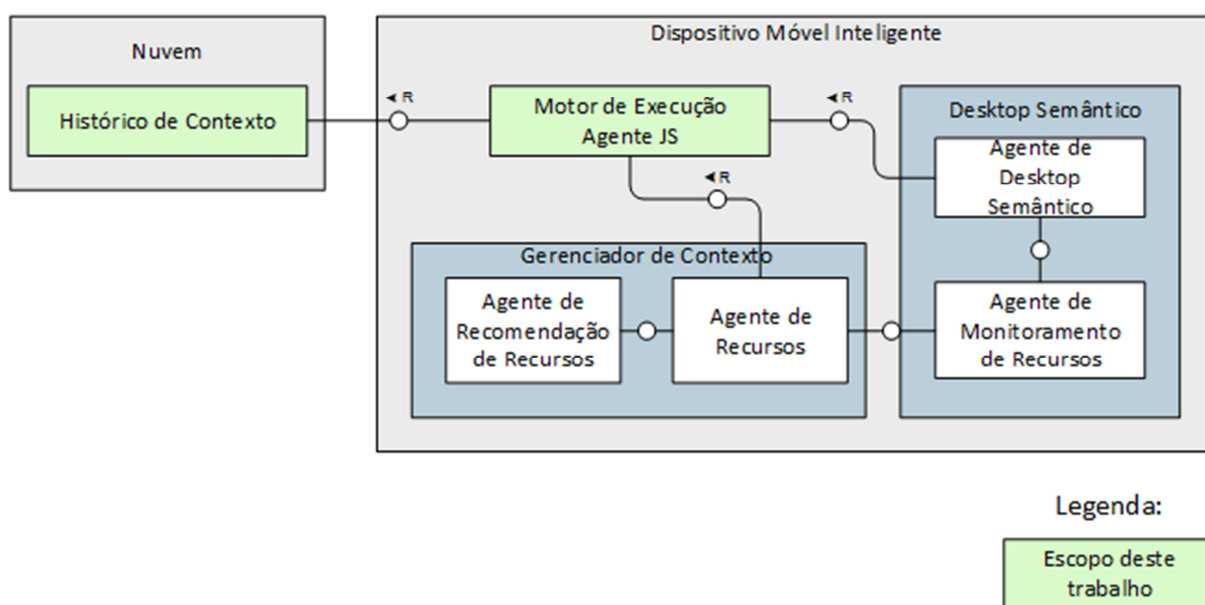


Figura 13 - Visão do programa "desktop semântico"

De cada componente da ilustração é possível dividir em duas plataformas distintas, sendo uma o dispositivo móvel e a outra a nuvem contendo o histórico de contextos. O dispositivo móvel possui os seguintes componentes:

- *Agente de monitoramento de recursos.* Responsável por monitorar as atividades do usuário e avisar o agente de recursos sempre que uma nova aplicação for inicializada. Em contrapartida receberá do Agente de recursos o estado dos recursos do sistema, como por exemplo, localização geográfica por GPS, estado do Bluetooth, acelerômetro e tempo.
- *Agente de recursos.* Responsável por receber as informações de uma aplicação em uso, e estampar dados de localização e tempo para armazenamento na nuvem (identificado como federação de contextos) enquanto existir conectividade.
- *Agente de recomendação de recursos.* Responsável por identificar uma mudança de contexto, seja por informações armazenadas na nuvem ou apenas devido ao deslocamento e conhecimento prévio do agente, e faz sugestões de aplicativos e recursos ao agente de desktop semântico. É o agente responsável por identificar padrões no dispositivo móvel e realizar recomendações baseadas nestes padrões.
- *Agente de desktop semântico.* Responsável por atualizar a tela inicial do smartphone de acordo com as sugestões do agente de recomendação de recursos, tendo assim capacidade de deixar um ícone em evidência caso seja apropriado para determinado contexto.

A segunda plataforma, exposta como o histórico de contextos na nuvem, é primariamente responsável por armazenar o histórico de contextos de forma distribuída e fornecer essas informações a agentes autorizados pelo usuário.

Estas informações sobre os recursos utilizados garantem escalabilidade e poder de processamento para realizar inferências sobre os dados armazenados com o uso da aplicação. Como objetivos secundários e descritos em mais detalhes nas próximas seções, o histórico de contextos é responsável por prover ferramentas e formas de permitir acesso ao contexto e de informar agentes sobre alterações no contexto a fim de permitir o uso do contexto não só dentro do programa, mas também o reuso das informações por outras aplicações ou dispositivos. A Figura 14 ilustra um cenário de uso do desktop semântico, onde ações são

tomadas por diferentes dispositivos quando o usuário da plataforma se aproxima do local de trabalho.

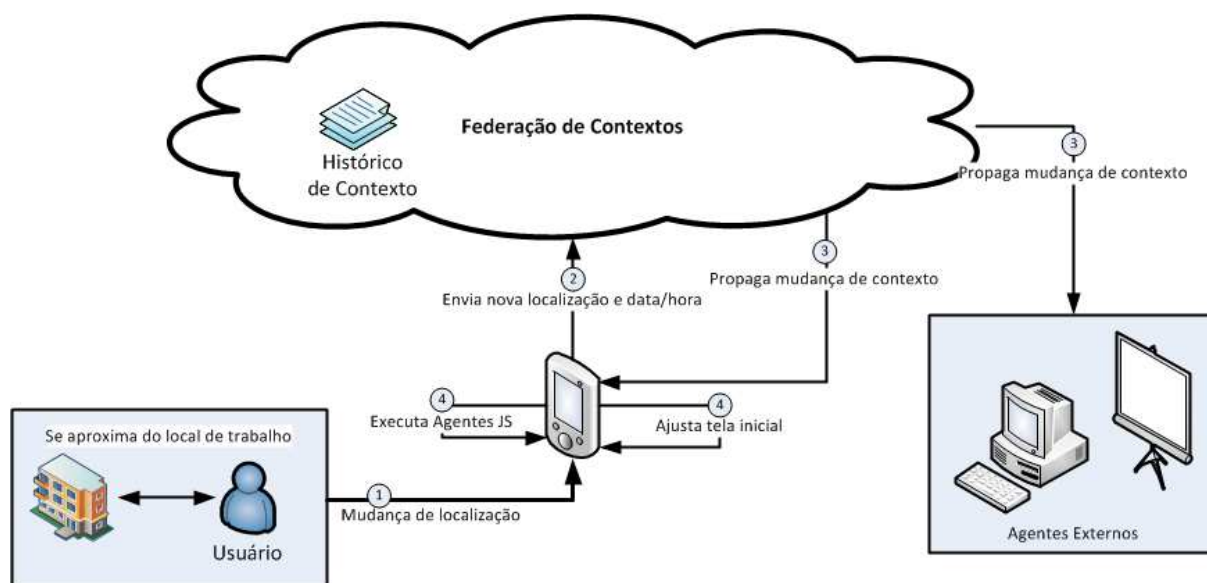


Figura 14 - Ilustração para o cenário de uso online

Este trabalho foi responsável por projetar e implementar o histórico de contextos na nuvem de forma a ter uma escalabilidade horizontal e prover acesso externo aos dados armazenados nas ontologias via uma interface web.

## 4.2 VISÃO GERAL DA PROPOSTA DE TRABALHO

A implementação deste trabalho dividiu a arquitetura em dois módulos principais: (i) o Histórico de contextos na nuvem, responsável pela implementação dos requisitos relacionados com a exposição de contextos para dispositivos externos ao ambiente inteligente utilizando bancos de dados distribuídos no armazenamento das ontologias, e (ii) o Motor de execução de agentes, responsável pela implementação dos requisitos relacionados com a comunicação de ambientes inteligentes e dispositivos móveis, bem como a possibilidade de criação de agentes sensíveis a contexto interpretados pelo dispositivo móvel.

## 4.3 HISTÓRICO DE CONTEXTOS NA NUVEM

O histórico de contexto na nuvem é o componente desta proposta responsável por armazenar mudanças do contexto de um dispositivo móvel em ontologias, o que permite a

agentes externos acessem informações sobre o contexto de um dispositivo móvel independente da plataforma ou linguagem que foram desenvolvidos.

Este capítulo está dividido da seguinte forma: a seção 4.3.1 apresenta o objetivo do componente e os requisitos não funcionais que impactaram diretamente na arquitetura do sistema, a seção 4.3.2 descreve os principais blocos da arquitetura do componente e a divisão de responsabilidades, a seção 4.3.3 apresenta como foi modelado o histórico de contexto em ontologias OWL, a seção 4.3.4 descreve o funcionamento do processo de inicialização do servidor de aplicação e a leitura inicial dos modelos quando os mesmos ainda não existem na base de dados, a seção 4.3.5 descreve a arquitetura e o processo de armazenamento e leitura de grafos RDF no HBase, a seção 4.3.6 descreve os serviços disponibilizados pelo histórico de contexto na nuvem, como a interface do serviço REST para atualização de contexto utilizada pelo componente instalado no dispositivo móvel e a interface do serviço REST para seleção de informações do histórico de contexto baseado em SPARQL.

#### 4.3.1 Requisitos do componente

Antes de apresentar a arquitetura do histórico de contextos é importante definir os requisitos e responsabilidades sob as quais foram tomadas as decisões de projeto relacionadas ao histórico de contexto. A *user story* que motiva a construção do histórico de contexto na nuvem é:

- Como desenvolvedor de dispositivos inteligentes, quero ser capaz de identificar o contexto do meu usuário com objetivo de tomar ações ou gerar notificações relevantes, como por exemplo, notificar o usuário sobre a necessidade de colocar óleo no carro quando o nível do mesmo estiver baixo e o usuário estiver parado em um posto de gasolina.

E a partir desta *user story* é possível derivar e detalhar os seguintes requisitos não funcionais:

- *Escalabilidade*. O histórico de contexto na nuvem deve ser projetado para manter o histórico de contextos de usuários sem limite de tempo, e isso significa um grande volume de dados armazenados ao longo da utilização do sistema. Atualmente a melhor estratégia é projetar o sistema para suportar a distribuição de processamento e armazenamento, e permitir uma escalabilidade horizontal.

- *Disponibilidade.* O sistema deve estar disponível quando o contexto for atualizado ou quando um dispositivo inteligente necessitar informações do contexto de um dispositivo móvel. Para atender essa necessidade se faz necessário o uso de *cloud computing* como infraestrutura de alta disponibilidade.
- *Acesso independente de plataforma.* O histórico de contexto na nuvem deve possibilitar o acesso ao contexto de forma independente de linguagem ou plataforma de acesso, uma vez que visa o uso a partir dispositivos em ambientes heterogêneos. Para atender esse requisito é utilizado o protocolo HTTP com o padrão arquitetural REST que permite o acesso aos dados independente de plataforma ou linguagem.
- *Padronização.* O histórico de contexto na nuvem deve aderir a padrões quando disponíveis, o que implica no uso de ontologias como forma de compartilhamento de conhecimento sobre o contexto de dispositivos móveis.

#### 4.3.2 Arquitetura do componente

O diagrama de blocos ilustrado na Figura 15 apresenta os principais módulos da arquitetura do componente de Histórico de Contextos na Nuvem, que foi construindo utilizando as seguintes ferramentas e plataformas:

- *Apache Tomcat 7.0.29.* Servidor de aplicação Java.
- *Apache Hadoop 1.0.4.* Sistema de arquivos distribuído para o banco de dados.
- *Apache HBase 0.94.4.* Banco de dados distribuído.
- *Jena 2.6.4.* Uma biblioteca de manipulação de RDF.
- *Jersey 1.17.* Uma biblioteca para construção de serviços REST.

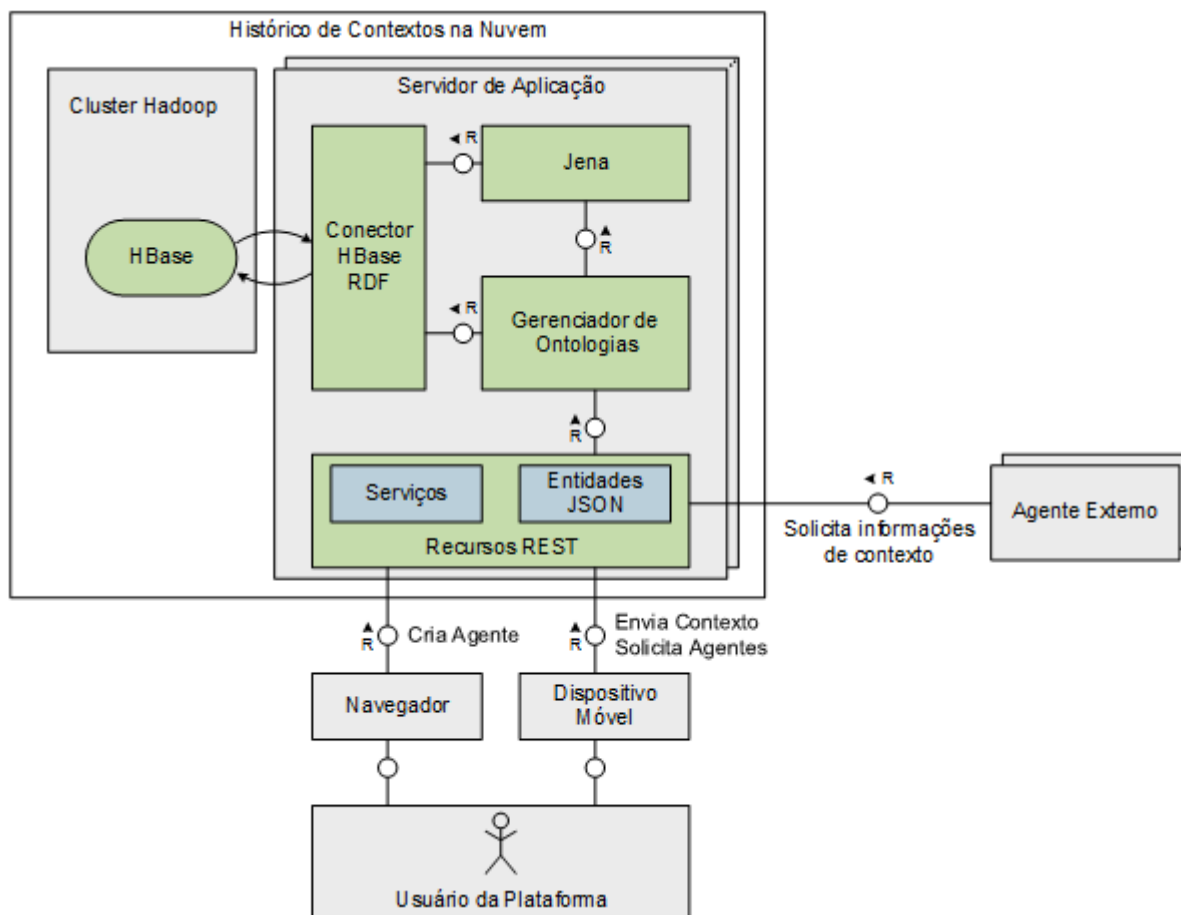


Figura 15 - Arquitetura do Histórico de Contexto na Nuvem

A divisão de responsabilidade entre os principais blocos da arquitetura é definida como a seguinte:

- *Conector HBase RDF.* É responsável por abrir e manter a conexão com o banco de dados HBase, bem como realizar as operações de leitura e escrita conforme solicitado pelo Jena, e foi baseado no trabalho descrito no capítulo 3.7. O diagrama de classes e seu funcionamento são descritos em maiores detalhes no capítulo 4.3.5.
- *Gerenciador de Ontologias.* É o módulo responsável por ler, transformar e armazenar entidades JSON em objetos RDF conforme solicitado pelo módulo de serviços REST. Também é responsável por solicitar a abertura de conexão com a base de dados para o módulo Conector HBase RDF e realizar a carga dos modelos definidos na seção 4.3.3 para o grafo RDF armazenado no banco de dados.
- *Recursos REST.* É composto por dois módulos, o módulo de Entidades JSON que é um conjunto de objetos Java com anotações para conversão para JSON pela

biblioteca Jersey conforme o protocolo para comunicação com os serviços, e pelo módulo de Serviços que agrupa as classes responsáveis por receber as requisições de seleção ou alteração de informações. Todas as requisições realizadas para o Histórico de Contexto na Nuvem é encaminhada a este componente que repassa a solicitação para o Gerenciador de Ontologias realizar a operação solicitada. As operações e o funcionamento deste módulo são detalhados na seção 4.3.6.

### 4.3.3 Modelagem do contexto em ontologias

Conforme descrito no capítulo 3.8, o modelo deste trabalho foi baseado na ontologia SOUPA, e segue uma estrutura modular de documentos OWL, sendo que a raiz de todo espaço de nome das ontologias definidas para este trabalho foi determinado como <http://swe.unisinos.br/ont/<nome da ontologia>>. Os documentos OWL criados para o histórico de contextos são:

- *Ontologia de Tempo*. Contem conceitos equivalentes à ontologia SOUPA-core, é formada por três classes, a superclasse denominada *TemporalThing* que representa um objeto de tempo, e duas subclasses disjuntas: *Instant* que possui a propriedade *atTime*, cujo valor representa os milissegundos passados desde 1º de janeiro de 1900, e a propriedade *timezone* que representa o fuso horário do instante. A segunda subclasse é *Interval*, que contem duas propriedades *from* e *to*, cujo domínio é limitado a indivíduos da classe *Instant*. A Figura 16 ilustra a relação entre as classes utilizando a notação descrita no capítulo 2.6.2.



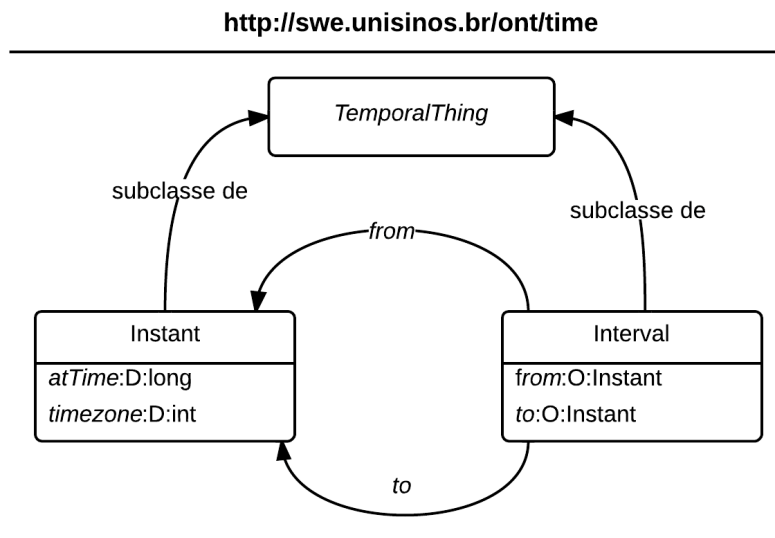


Figura 16 - Ontologia de Tempo

- *Ontologia de Localização.* Assim como a ontologia de tempo também possui conceitos equivalentes à ontologia SOUPA-core e é formada por 24 classes que representam posição geográfica, geopolítica, formas de áreas e localizações compostas. A Figura 17 ilustra as principais classes da ontologia de localização que são: a classe de coordenadas geográficas (*GeoCoordinates*), que representa uma coordenada composta de latitude e longitude com equivalência a classe *Point* e as propriedades *lat* e *long* da especificação *WGS84 lat/long*<sup>3</sup>, a classe ponto único (*SinglePoint*), que é uma subclasse de forma (*Shape*) representando um ponto geográfico sem forma definida e é uma classe disjunta de todas as demais classes de formato. A classe local composto (*CompoundPlace*), que representa um local composto por um ou mais classes do tipo local (*Place*), podendo ser composto de outros locais compostos (*CompoundPlace*) ou de locais atômicos (*AtomicPlaces*), que representa um local indivisível, e que deve possuir uma forma identificando não só o formato do local, mas também sua coordenada geográfica.

<sup>3</sup> Vocabulário WGS84 lat/long disponível em <http://www.w3.org/2003/01/geo/#vocabulary>

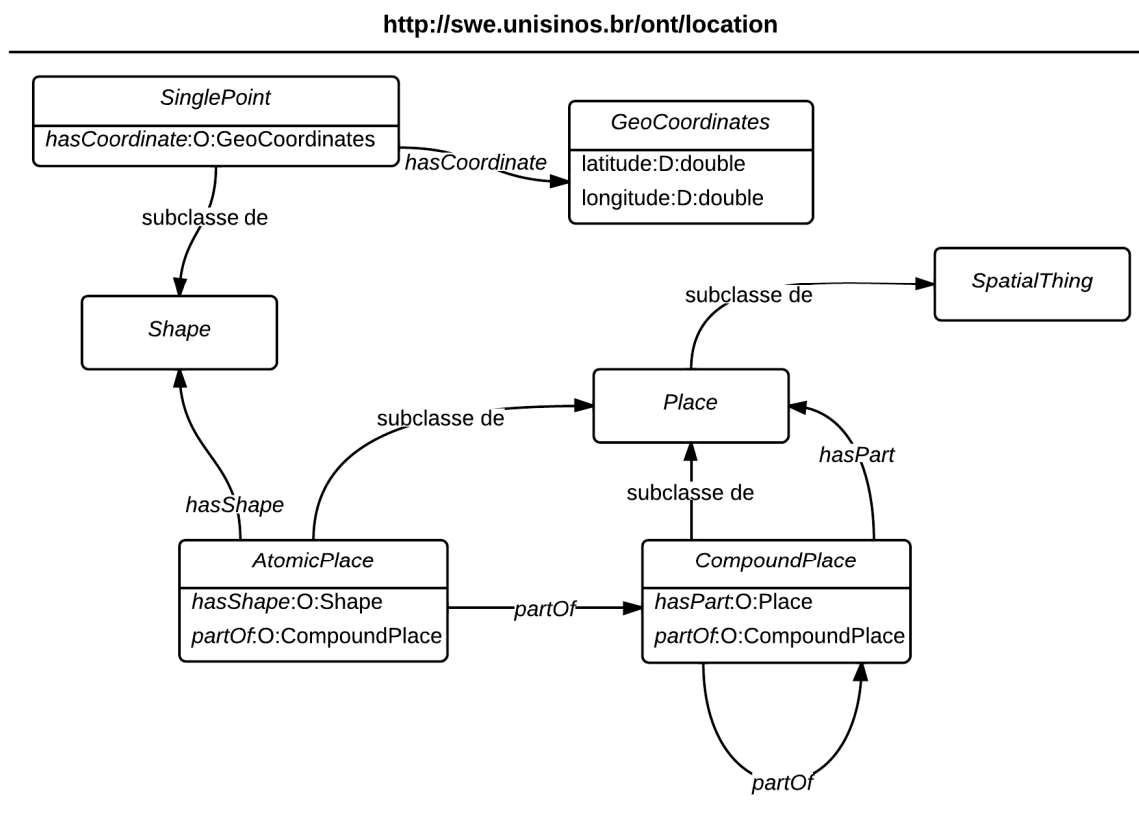


Figura 17 - Ontologia de Localização

- Ontologia de Agentes.* A ontologia de agentes contém o vocabulário responsável por definir agentes (Hardware, Software e Pessoas) e seus respectivos relacionamentos. No vocabulário definido para este trabalho o conceito de pessoa (*Person*) e a propriedade “conhece” (*knows*) são equivalentes as mesmas definições no vocabulário *Friend of a Friend*<sup>4</sup> (FOAF) para permitir o reuso de conceitos existentes. A Figura 18 ilustra o vocabulário, onde é possível notar o relacionamento de dispositivo móvel (*MobileDevice*) com aplicativos (*Application*), e o relacionamento entre a classe pessoa (*Person*) e o dispositivo móvel. É importante destacar que a autenticação neste trabalho se dá através de OAuth 2.0 utilizando o Facebook como provedor para permitir que trabalhos futuros importem informações da rede social para enriquecer a análise do histórico de contexto, e portanto uma classe representando o perfil no facebook (*FacebookProfile*) foi adicionada ao modelo.

<sup>4</sup> Vocabulário FOAF disponível em <http://xmlns.com/foaf/spec/>

http://swe.unisinos.br/ont/agents

PREFIX foaf: <http://xmlns.com/foaf/0.1#>

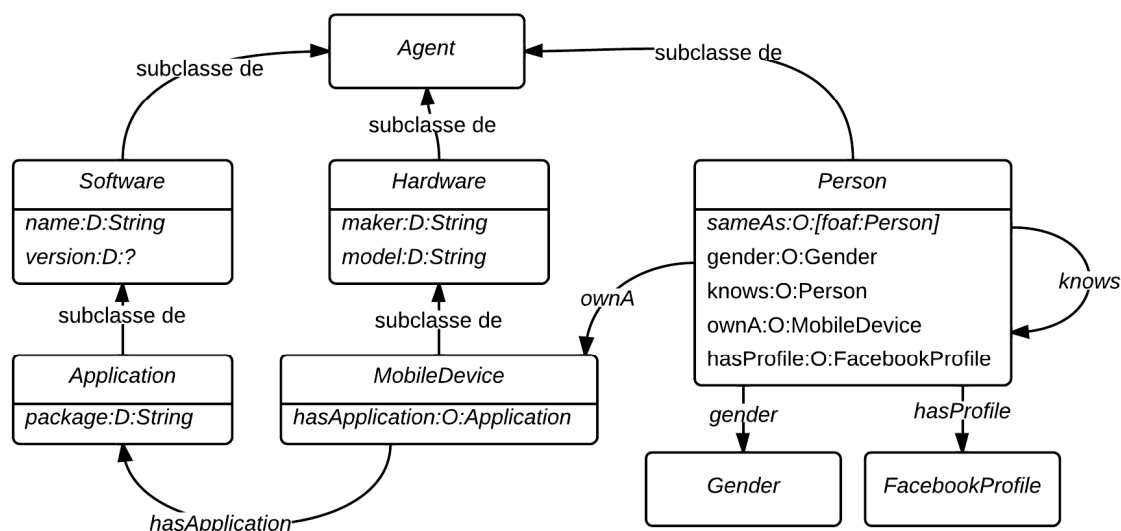


Figura 18 - Ontologia de Agentes

- Ontologia de Contexto.* A ontologia de contexto foi construída a partir do conceito de contexto sugerido por (Brooks, 2003) onde um contexto pode ser determinado a partir da resposta de cinco perguntas: Quando, Onde, O que, Quem e Por que. Devido à característica subjetiva da última pergunta (“Por que”), a ontologia de contexto foi modelada para responder as quatro primeiras perguntas (Quando, Onde, O que e Quem). Essa ontologia importa as definições das ontologias de tempo, localização e agentes, e é composta por três classes: *Activity* que representa uma atividade sendo realizada por alguém, *SoftwareActivity* que representa o uso de um software e *Context*, que procura responder as quatro perguntas utilizando classes definidas nas demais ontologias, portanto é utilizada a classe *SpatialThing* para definir Onde que a atividade está sendo realizada, a classe *TemporalThing* para definir Quando a atividade está sendo realizada, a classe *Person* para definir quem está executando a atividade, e a classe *Activity* para definir Qual atividade está sendo realizada (O Que). A Figura 19 ilustra a relação entre essas classes.

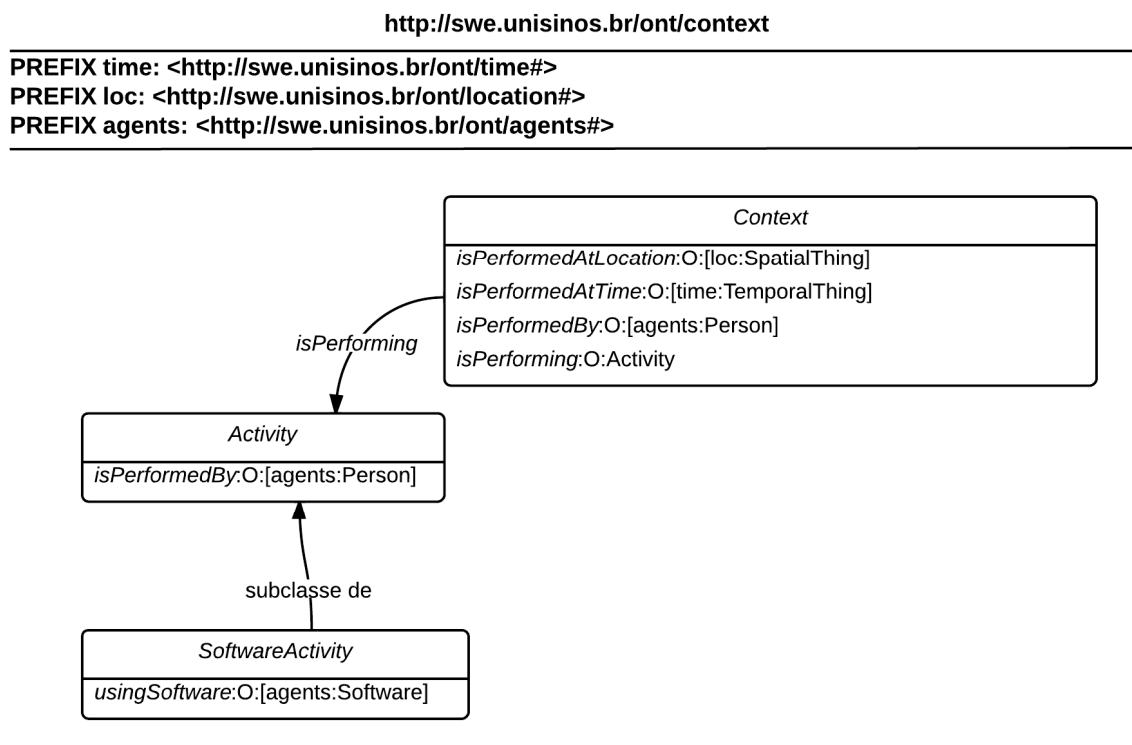


Figura 19 - Ontologia de Contexto

As quatro ontologias descritas neste capítulo permitem o entendimento da forma como as informações sobre o histórico de contexto é armazenada e disponibilizada para seleção via interface SPARQL, e é a especificação básica necessária para que um agente externo obtenha informações do contexto que um determinado usuário da plataforma se encontra em um determinado momento.

#### 4.3.4 Inicialização do servidor e leitura dos modelos

A especificação *Servlet* prevê quatro tipos de Event Listeners, que são interfaces que quando implementadas e indicadas a partir de uma anotação na classe (adotada a partir da especificação *Servlet 3.0*) permitem receber informações sobre eventos que ocorrem no servidor de aplicação.

Neste trabalho foi implementada a interface *ServletContextListener* que é o evento disparado durante a inicialização de uma aplicação web para solicitar a abertura da conexão com o banco de dados, para a carga dos documentos OWL no grafo RDF e para inicialização das classes de vocabulário auxiliares que visam facilitar a manipulação das ontologias pelo Jena.

A Figura 20 é a ilustração do diagrama das classes envolvidas no processo de inicialização, onde a classe *ApplicationLifeCycle* é implementa a interface *ServletContextListener* e realiza as operações de criação do *Singleton OntologyManager*, e do conjunto de classes anotadas na classe *Vocabulary*. As classes *HBaseRdfConnection* e *StoreSimple* representam respectivamente a classe responsável pela conexão com a base de dados distribuída e a classe de formatação das triplas.

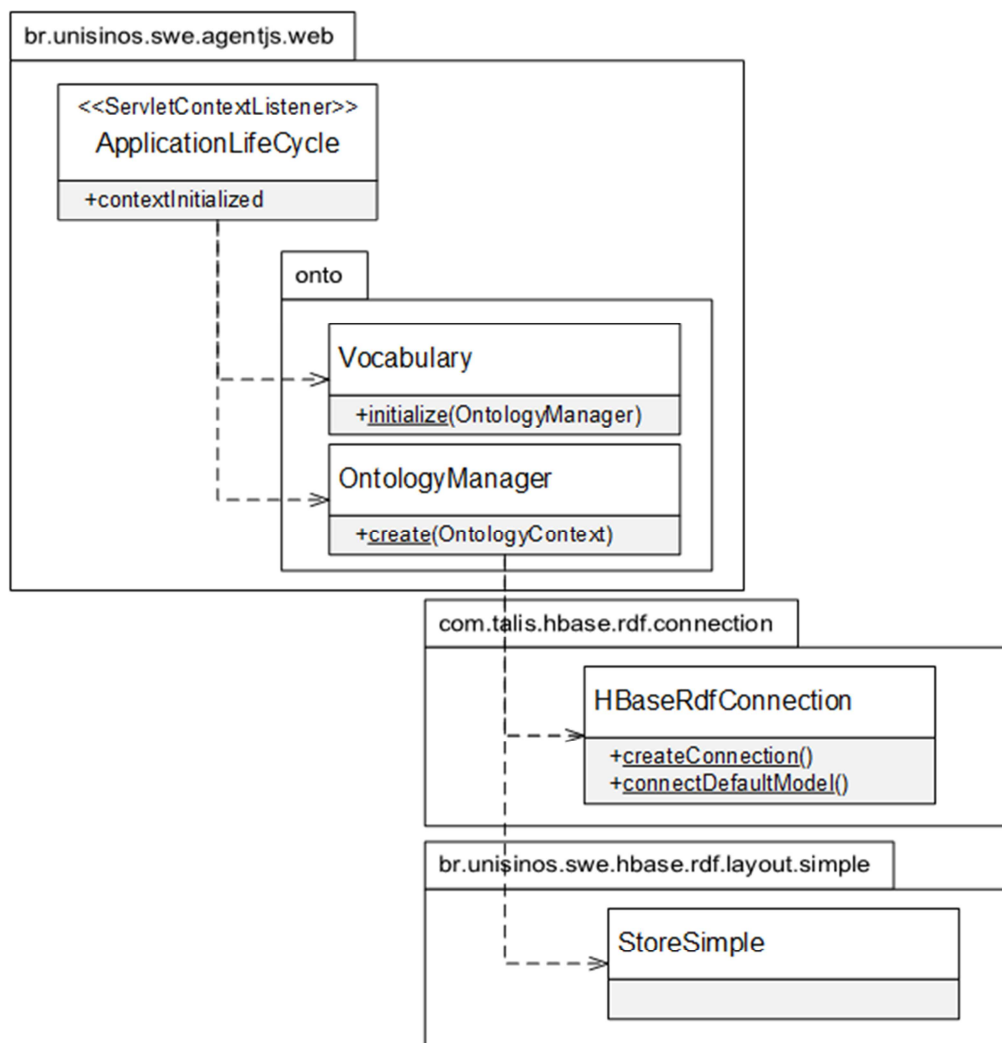


Figura 20 - Diagrama de classe do processo de inicialização

A Figura 21 ilustra o diagrama de sequencia das ações relacionadas com as três classes do procedimento de inicialização, é importante notar a dependência do vocabulário (*Vocabulary*) com o gerenciador de ontologias (*OntologyManager*), uma vez que uma classe no vocabulário representa uma classe do modelo OWL e, portanto, a instância deve ser criada a partir do gerenciador de ontologias. O uso da classe *ServletContext* pelo gerenciador de ontologias se deve a característica da carga dos modelos, que é efetuada a partir da leitura dos

documentos OWL existentes na pasta “WEB-INF/model”, cujo caminho completo é definido pela classe *ServletContext*.

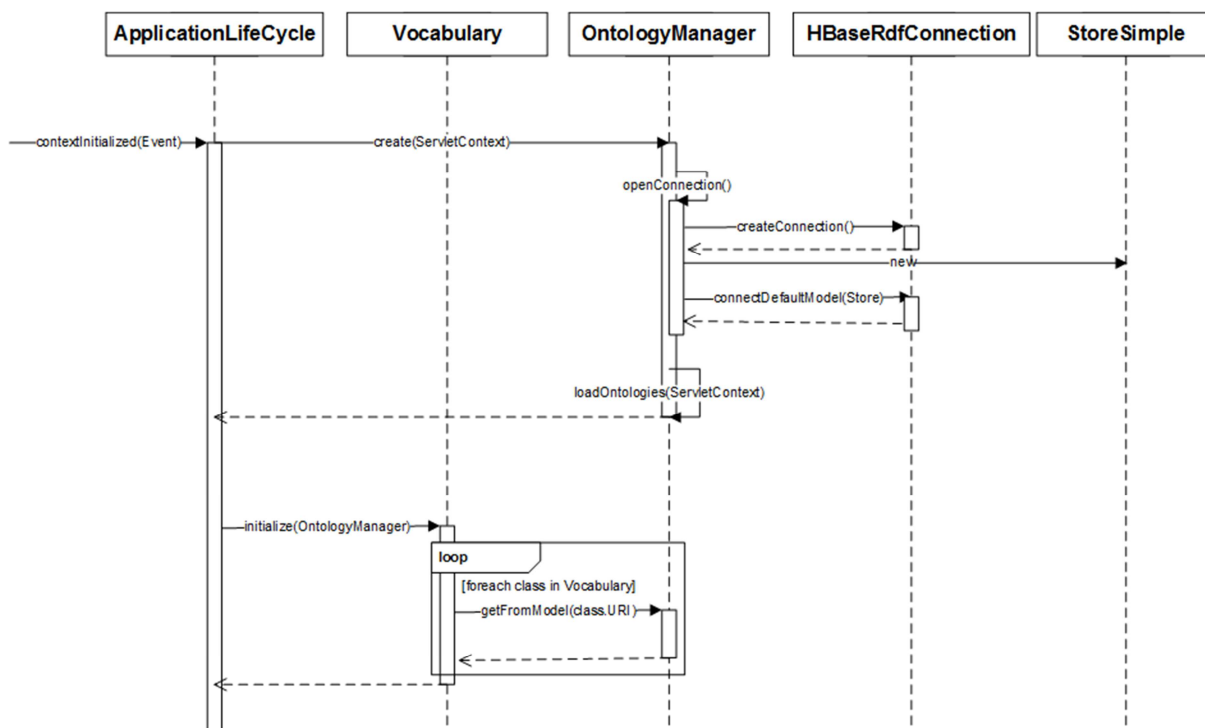


Figura 21 - Diagrama de sequência do processo de inicialização

#### 4.3.5 Armazenamento e leitura de dados no HBase

O armazenamento de grafos RDF no banco de dados distribuído HBase deste trabalho foi realizado utilizando o resultado do trabalho descrito na seção 3.7 com ajustes no formato simplificado para evitar erros relacionados ao separador de colunas utilizado nesse formato.

Neste tipo de formatação são criadas três tabelas de indexação para facilitar a busca por algum item da tripla sujeito-predicado-objeto, onde a linha é o item indexado, a coluna é composta pelo próximo valor da tripla seguido de um separador e um *hash* MD5 do terceiro valor da tripla (para permitir múltiplos objetos associados à mesma chave de coluna, por exemplo, vários objetos para um determinado sujeito-predicado), e o conteúdo da combinação linha/coluna é o valor completo do terceiro valor da tripla.

O motivo da alteração da formatação é devido à impossibilidade do algoritmo no formato simplificado do trabalho anterior em identificar se o separador é efetivamente um separador ou apenas uma informação da segunda parte da coluna. O uso do *hash* MD5 mantém a característica de possibilitar múltiplos objetos para um determinado sujeito-

predicado, e permite encontrar o separador sem possibilidade de erros, a Figura 22 ilustra a adaptação do formato simples.

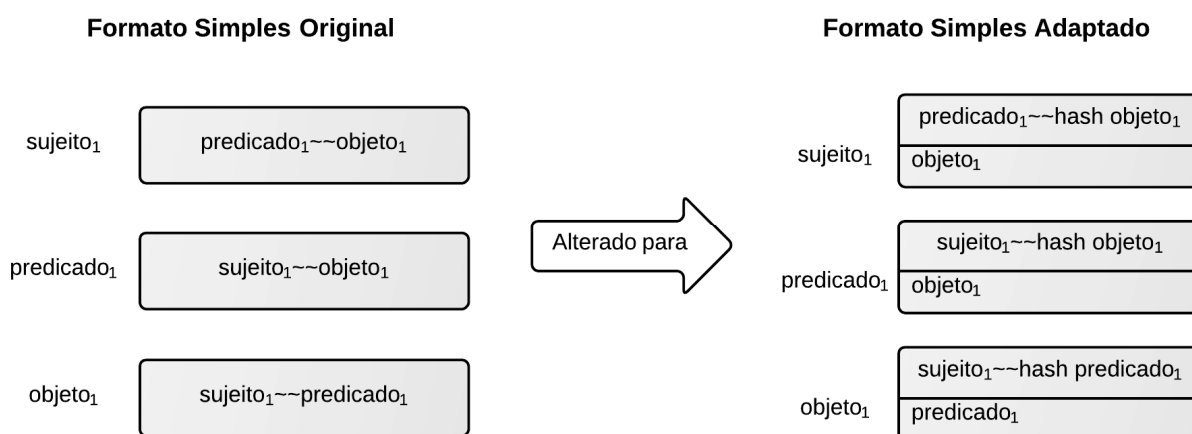


Figura 22 - Adaptação do formato simples de armazenamento de triplas

A Figura 23 ilustra um fragmento de uma tabela de sujeitos contendo um indivíduo da ontologia de agentes (na linha como chave do sujeito) com dois predicados (onde o nome é o predicado concatenado com o separador e o hash do objeto) e os valores dos respectivos objetos.

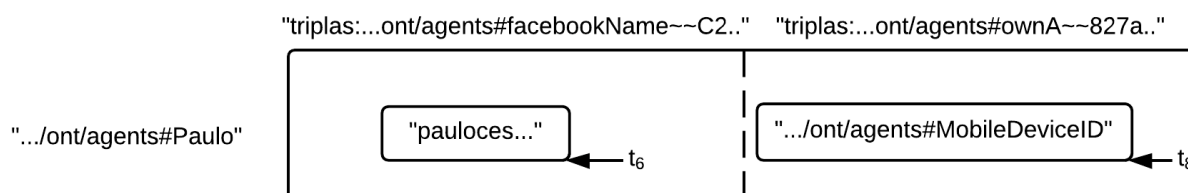


Figura 23 - Exemplo com fragmento de tabela de sujeitos

Conforme detalhado na seção 3.7, o procedimento de armazenamento e recuperação de triplas do banco de dados distribuído é baseado na implementação existente SDB, onde a interface de armazenamento do SDB (chamada *Store*) é implementada para realizar os acessos e leituras a partir do cliente do HBase. As principais classes envolvidas nesse procedimento são:

- Classe *StoreSimple*. Implementação da interface *Store*, é dependente das classes *QueryRunnerSimple* e *TupleLoaderSimple* para realizar operações de leitura e armazenamento.
- Classe *QueryRunnerSimple*. Realiza a operação de busca recebendo do Jena as informações sobre a tripla (sujeito, predicado, objeto) que deve ser recuperada.

A partir das informações recebidas a classe determina se existe algum valor na tripla que permita a leitura direta em uma das três tabelas indexadas e repassa a informação da linha para a classe *HBaseRdfSingleRowIterator* ler a tripla, ou se é necessário uma leitura de toda a tabela, que por padrão é realizada sobre a tabela de sujeitos e repassada para a classe *HBaseRdfSingleTableIterator*.

- *Classe HBaseRdfSingleRowIterator*. É responsável por iterar sobre uma linha da tabela no HBase e retornar todas as triplas disponíveis.
- *Classe HBaseRdfSingleTableIterator*. É responsável por iterar por todas as linhas de uma tabela no HBase, e solicitar as triplas para a classe *HBaseRdfSingleRowIterator*.
- *Classe TupleLoaderSimple*. É classe responsável por construir e popular as tabelas no HBase. Recebe o comando de criação de tabelas durante o procedimento de inicialização do servidor de aplicação e gera as três tabelas necessárias para indexar sujeitos, predicados e objetos. Ao receber o comando de *loadTuple* realiza a operação de criação de registro (*Put*) no HBase, populando as três tabelas de acordo com as informações recebidas.

As classes e suas dependências estão ilustradas no diagrama de classe na Figura 24, as operações ilustradas foram restritas as operações descritas nesta seção, que são essenciais para o funcionamento em conjunto com o Jena.



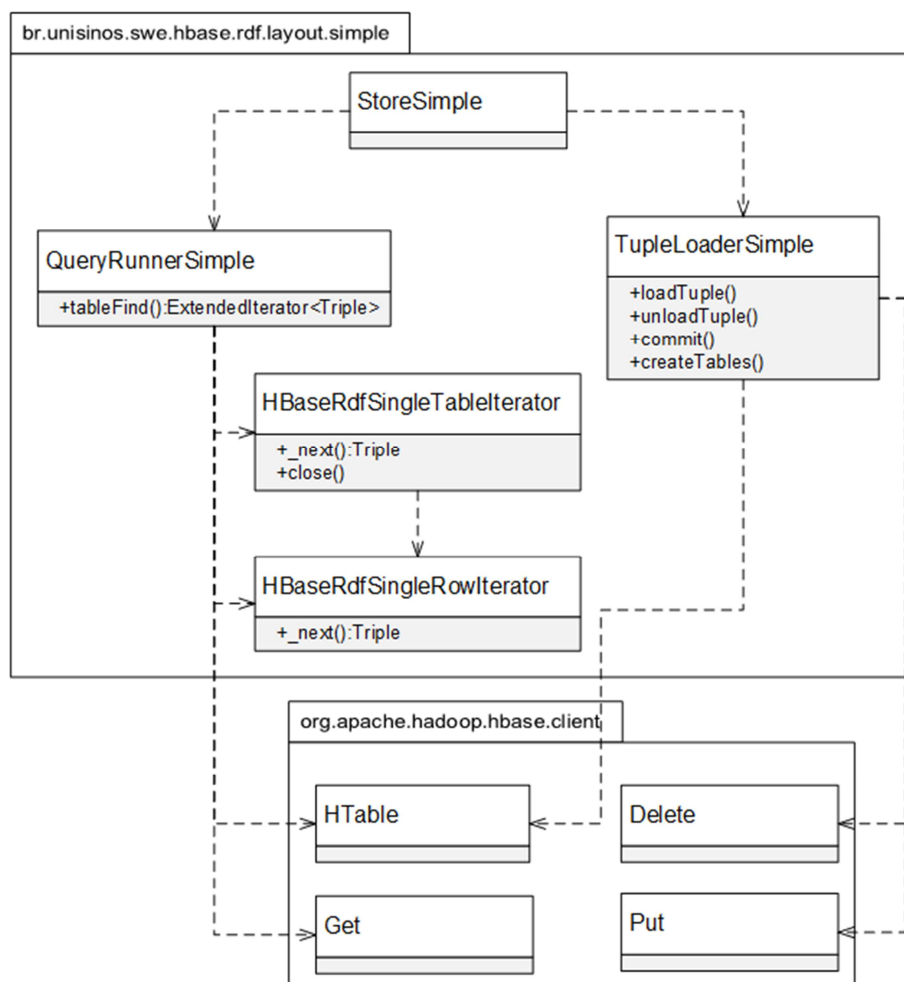


Figura 24 - Diagrama de classes de armazenamento de grafos RDF no HBase

#### 4.3.6 Serviços REST do histórico de contexto

Toda comunicação com o histórico de contexto na nuvem se dá a partir de serviços seguindo o padrão arquitetural REST, projetados para atender três situações:

- *Requisições de agentes externos.* Foi construído um serviço para receber uma consulta SPARQL e executar sobre os dados distribuídos para permitir que agentes externos realizem consultas sobre o contexto de um determinado usuário. Esse serviço tem duas opções de formato de retorno: RDF/XML e RDF/JSON.
- *Requisições da página de agentes.* Foi construída uma interface web para criação de agentes e um serviço com as operações de listar, criar e alterar agentes de um determinado usuário.
- *Requisições do dispositivo móvel.* O terceiro serviço foi construído para receber requisições HTTP de forma periódica, contendo as informações do

contexto em que se encontra o dispositivo móvel, permitindo assim a construção do histórico de contexto.

No serviço de consulta SPARQL toda interpretação da instrução de consulta fica a cargo do mecanismo existente na biblioteca Jena que realiza a leitura dos dados persistidos de acordo com as classes descritas na seção 4.3.5, a conversão do resultado para o formato solicitado no endereço relativo da requisição também faz uso de bibliotecas auxiliares existentes no Jena. Seguindo o padrão arquitetural REST, o procedimento para utilização do serviço é a realização de uma requisição HTTP com os caminhos relativos e corpo de mensagem conforme os descritos na Tabela 3.

Tabela 3 - Interface do serviço de consulta SPARQL

Serviço de consulta SPARQL	
Método	POST
Endereço relativo	/WEB/REST/QUERY para formato RDF/XML /WEB/REST/QUERY/JSON para formato RDF/JSON /WEB/REST/QUERY/TEXT para formato de texto legível
Corpo da requisição	Consulta SPARQL. Exemplo de seleção de perfil com nome identificador no facebook igual a "paulocesar.buttendbender":  <pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX agents: &lt;http://swe.unisinos.br/ont/agents#&gt;  SELECT ?s WHERE {   ?s rdf:type agents:FacebookProfile .   ?s agents:facebookName ?o   FILTER ( ?o = "paulocesar.buttendbender" ) }</pre>
Resposta da requisição	Resposta com os parâmetros resultantes da seleção conforme formato requisitado. Exemplo de resposta em formato RDF/JSON: <pre> {   "head": {</pre>

	<pre> "vars": [ "s" ] } , "results": {   "bindings": [     {       "s": {         "type": "uri" ,         "value": "http://swe.unisinos.br/ont/agents#Paulo"       }     }   ] } } </pre>
--	---

O serviço para a interface web de criação e alteração de agentes é composto por três ações distintas, uma com objetivo de recuperar uma lista de agentes de um determinado usuário, outra com objetivo de criar um agente novo para o usuário, e a terceira ação com objetivo de alterar um agente existente de um usuário. Devido a essa característica esse serviço é composto por três métodos HTTP para cada uma das requisições, conforme descrito na Tabela 4.

Tabela 4 - Interface para o serviço de criação de agentes

Serviço de criação de agentes – Listagem de Agentes	
Método	GET
Endereço relativo	/WEB/REST/AGENT/{user} Sendo {user} o identificador do usuário ao qual se deseja recuperar a lista de agentes
Resposta da requisição	Lista de agentes no formato JSON. Exemplo de resposta: [ { "id" : "agent_uuid", "name" : "nome do agente", "source" : " var x = 1; agent.log(x);" }]

	]
<b>Serviço para interface web – Criação de agente</b>	
Método	POST
Endereço relativo	/WEB/REST/AGENT/{user} Sendo {user} o identificador do usuário que se deseja utilizar para criar o agente.
Corpo da requisição	Objeto no formato JSON com o agente que deve ser criado. Exemplo: <pre>{   "name" : "nome do agente",   "source" : " var x = 1; agent.log(x);" }</pre>
Resposta da requisição	Retorno da requisição com o código 201 (Criado) se o agente foi criado corretamente.
<b>Serviço para interface web – Alteração de agente</b>	
Método	PUT
Endereço relativo	/WEB/REST/AGENT/{user}/{ID} Sendo {user} o identificador do usuário e {ID} o identificador do agente
Corpo da requisição	Objeto no formato JSON com o agente que deve ser criado (o mesmo corpo utilizado no serviço de criação de agente). Exemplo: <pre>{   "name" : "nome do agente",   "source" : " var x = 1; agent.log(x);" }</pre>
Resposta da requisição	Retorno da requisição com o código 200 (OK) se o agente foi alterado corretamente.

Por fim, o principal serviço para o histórico de contexto na nuvem é o serviço responsável por atualização de contexto, e foi construído para receber de um dispositivo móvel uma descrição de contexto baseado nas informações disponibilizadas pelos sinais descritos na seção 4.4. O diagrama de classes na Figura 25 ilustra a classe de contexto que o

dispositivo móvel deve enviar periodicamente para o servidor de aplicação. Essa classe deve ser serializada utilizando a notação JSON, e seus dados serão convertidos pela classe módulo Gerenciador de Ontologias utilizando o vocabulário proposto por este trabalho e armazenados de forma distribuída no banco de dados HBase.

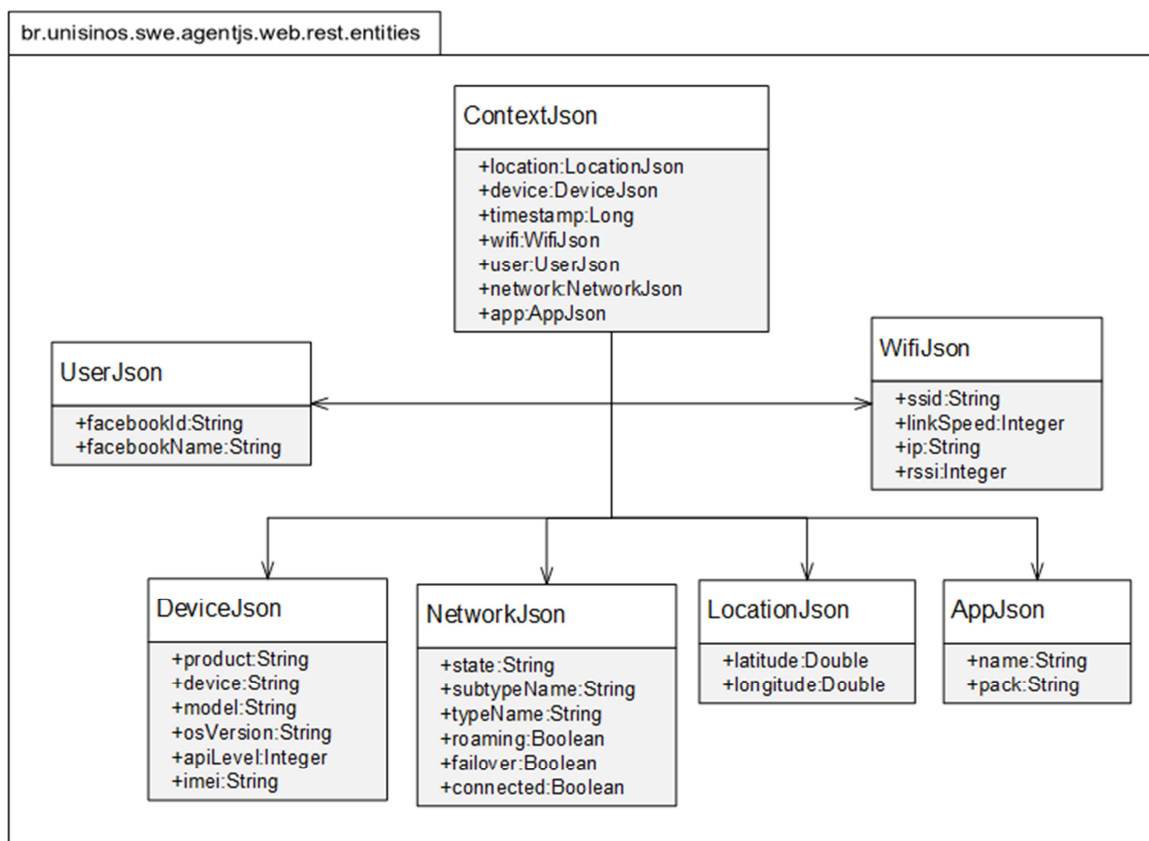


Figura 25 - Diagrama de classe das entidades do serviço de contexto

Uma vez que o contexto é armazenado, a informação se torna disponível para consulta de agentes externos. A Tabela 5 descreve o formato da requisição para o serviço de atualização de contexto.

Os aspectos de segurança e autorização de agentes externos para consulta de informações estão fora do escopo deste trabalho, e está relacionado como um possível trabalho futuro para controle de acesso a informações em ontologias.

Tabela 5 - Interface para o serviço de atualização de contexto

Serviço de atualização de contexto	
Método	POST
Endereço	/WEB/REST/CONTEXT

relativo	
Corpo da requisição	<p>Uma instância da classe <i>ContextJson</i> serializada utilizando a notação JSON.</p> <p>Fragmento de exemplo:</p> <pre>{   "timestamp": 123415135,   "device" :   { "imei" : "1321313", "apiLevel" : "12", ... },   "location" :   { "latitude" : -50.203, "longitude" : -30.123 },   ... }</pre>
Resposta da requisição	Retorno da requisição com o código 200 (OK) se o contexto for alterado com sucesso.

#### 4.4 MOTOR DE EXECUÇÃO DE AGENTES

O motor de execução de agentes é o componente desta proposta que engloba três funcionalidades distintas: (i) enviar periodicamente o contexto do dispositivo móvel para o componente de histórico de contextos na nuvem, (ii) receber agentes criados no aplicativo do histórico de contexto na nuvem e procurar agentes disponíveis em uma rede local via descoberta automática de serviços e (iii) executar as instruções dos agentes quando solicitado pelo usuário do aplicativo. O nome de motor de execução de agentes é devido à característica de implementar para a plataforma *Android* as funcionalidades da API descrita na seção 4.4.3, interpretando o código *Javascript* dos agentes criados utilizando esta mesma API.

Este capítulo está dividido da seguinte forma: a seção 4.4.1 apresenta o objetivo do componente e as *user stories* para a elicitação dos requisitos, a seção 4.4.2 descreve os principais blocos da arquitetura do componente e a divisão de responsabilidades, a seção 4.4.3 apresenta a API em *Javascript* proposta para construção de agentes e a funcionalidade de cada comando disponibilizado, a seção 4.4.4 descreve o funcionamento dos emissores de sinais, e como essa funcionalidade é compartilhada entre agentes e para construção periódica do objeto de contexto, a seção 4.4.5 descreve o funcionamento da execução de agente e como foi construída a interface para isolar a execução de diferentes agentes, a seção 4.4.6 descreve a descoberta automática de agentes via UPnP e a interface do serviço para busca de agentes e

código, por fim a seção 4.4.7 apresenta as telas do aplicativo desenvolvido e o resultado de um protótipo de agente.

#### 4.4.1 Requisitos do sistema

Antes de apresentar a arquitetura do motor de execução de agentes é importante definir os requisitos que nortearam o projeto tanto a API quanto do motor de execução. As *user stories* que motivam a construção do motor de execução são:

- Como usuário da aplicação, quero que minha interface inicie aplicativos relevantes ao contexto em que estou com objetivo de, por exemplo, facilitar o acesso a e-mails no trabalho, anotações em aula ou jogos em casa.
- Como desenvolvedor de dispositivos inteligentes, quero ser capaz de disponibilizar instruções de comando para o meu dispositivo inteligente para dispositivos móveis dos usuários.
- Como usuário da aplicação, quero ser capaz de criar aplicativos simples que utilizem meu contexto com objetivo de tomar ações relevantes, como por exemplo, programar meu dispositivo para ligar as luzes da minha residência assim que eu me aproximo da mesma.

E a partir destas *user stories* é possível derivar e detalhar os seguintes requisitos:

- *Independência de plataforma.* A API deve ser disponibilizada em uma linguagem passível de portabilidade para as principais plataformas de dispositivos móveis.
- *Controle de recursos.* O motor de execução será responsável por iniciar ou terminar a execução de tarefas que tem por característica o alto consumo de bateria, portanto recursos disponibilizados para a API devem ser ativados apenas quando solicitados, e desligados quando não forem mais necessários.
- *Estabilidade.* O motor de execução irá interpretar códigos que podem estar incorretos em termos de sintaxe ou performance, e um código fonte de agente com erro não pode afetar a execução de outro agente. O motor de execução deve garantir a independência de execução dos agentes e controlar o tempo de execução para evitar o consumo excessivo de recursos.
- *Interface para comandos nativos.* Os agentes devem ter a capacidade de invocar comandos nativos da plataforma de acordo com o que for

disponibilizado na API. O motor de execução deve intermediar uma requisição feita pelo agente para o respectivo comando nativo.

- *Descoberta automática de agentes.* O aplicativo deve ter condições de encontrar agentes em uma rede local sem a necessidade de configurações adicionais.

#### 4.4.2 Arquitetura do componente

O diagrama de blocos ilustrado na Figura 26 apresenta os principais módulos da arquitetura do componente de Motor de Execução de Agentes, que foi construindo utilizando as seguintes ferramentas e plataformas:

- *Android SDK 2.3.4.* Kit de desenvolvimento para plataforma Android 2.3.4.
- *Android support library v4 rev 11.* Biblioteca opcional para permitir o uso de componentes retro-compatíveis da plataforma Android.
- *Cling 1.0.5.* Biblioteca para busca, exposição e consumo de serviços UPnP, utilizada para descoberta automática de agentes.
- *Google Guava 14.0-rc1.* Biblioteca de utilitários construída pelo Google, utilizada para facilitar o controle de processamento concorrente de requisições HTTP.
- *Google Places API.* Serviço disponibilizado pela web para busca de locais baseado em palavras chave e localização geográfica. Utilizado para o sinal de proximidade de estabelecimentos.
- *Mozilla Rhino 1.7R4.* Implementação *Javascript* escrito em Java, utilizado para interpretação da API e a realização de chamadas de funções *Javascript* a partir do código Java.



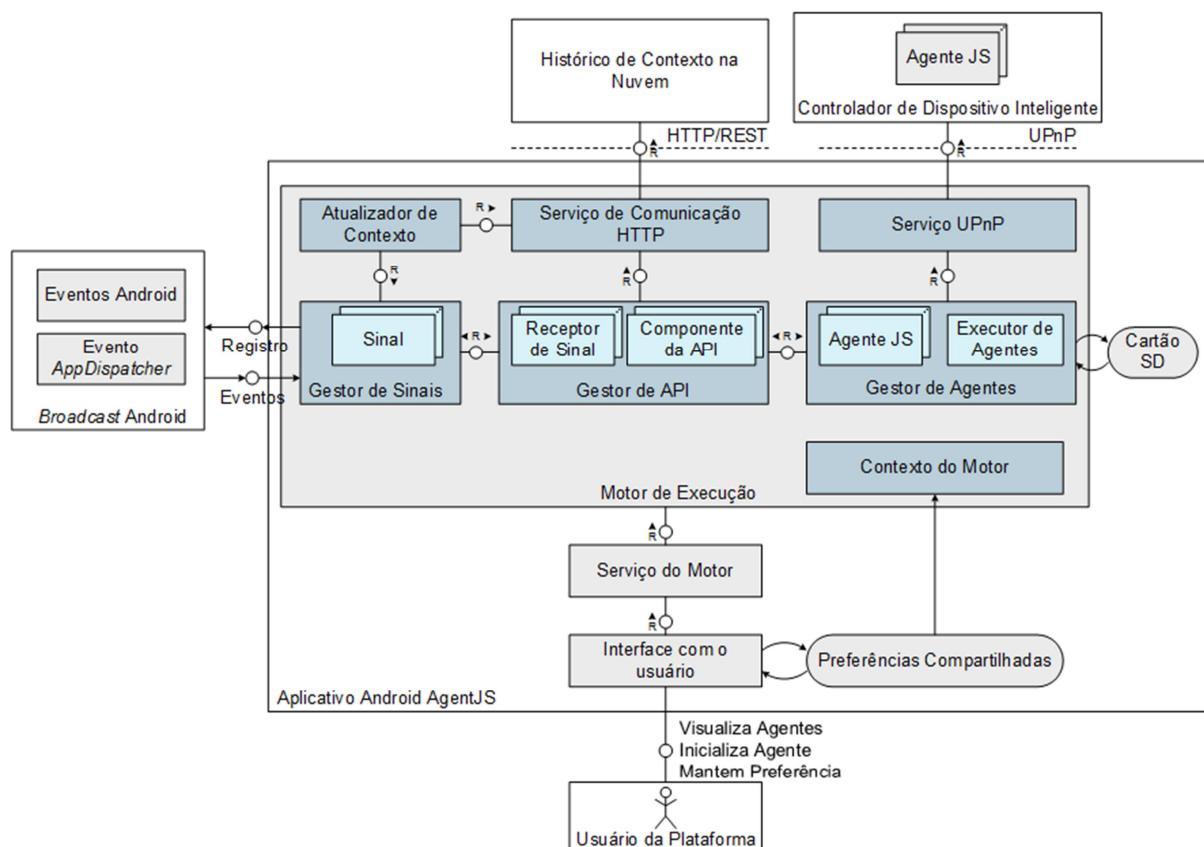


Figura 26 - Arquitetura do Motor de Execução de Agentes

O diagrama de blocos ilustra os principais módulos do motor de execução e as principais formas de comunicação com os demais aplicativos e com dispositivos externos. A divisão de responsabilidade entre os principais blocos da arquitetura é definida como a seguinte (seguindo a ordem de baixo para cima, direita para esquerda):

- *Interface com o usuário.* Módulo responsável pela interface com o usuário e pela inicialização do serviço do motor de execução. Composto pelas telas de listagem de agentes da nuvem, listagem de agentes em rede local e manutenção de preferencias.
- *Serviço do Motor.* O serviço *Android* para permitir que o motor de execução funcione mesmo após fechar a interface do aplicativo. É inicializado pela Interface com o usuário e tem por responsabilidade definir os métodos utilizados pela interface, e por iniciar o módulo de motor de execução de agentes.
- *Motor de Execução.* Principal módulo do aplicativo é responsável pelas tarefas de inicialização e término dos componentes e do *Rhino* para execução ou término de execução de agentes. É composto por sete módulos, cada qual com

responsabilidade sobre uma funcionalidade específica: (i) Contexto do Motor, (ii) Gestor de Agentes, (iii) Gestor de API, (iv) Gestor de Sinais, (v) Atualizador de Contexto, (vi) Serviço de Comunicação HTTP e (vii) Serviço UPnP.

- *Contexto do Motor*. Módulo responsável por manter informações compartilhadas entre todos os demais módulos, como por exemplo, o acesso as preferencias do usuário, acesso a interface de Log e acesso a classe de contexto do aplicativo Android.
- *Gestor de Agentes*. Módulo responsável pela leitura e execução de agentes. Realiza o processo de leitura a partir do cartão SD do dispositivo e requisita ao Serviço UPnP a busca por agentes em uma rede local. É composto pelos módulos de Agente JS e de Executor de Agentes.
- *Agente JS*. Representa um código de agente presente no Gestor de Agentes.
- *Executor de Agentes*. Responsável por criar um escopo e contexto *Javascript* para um Agente JS e iniciar uma nova *Thread* para interpretação do código pelo *Rhino* utilizando a API disponibilizada.
- *Gestor de API*. Módulo responsável por disponibilizar os componentes da API para o interpretador *Rhino* e por criar os receptores de sinais quando solicitado por um Agente JS.
- *Componente da API*. Cada módulo disponibilizado para a API de Agente JS é implementado como um componente da API e disponibilizado para uso do interpretador.
- *Receptor de Sinal*. Criado conforme solicitado por um componente de recepção de sinal, fica aguardando uma resposta em memória até ser chamado disparado por um Sinal. Quando disparado, realiza a chamada da função *Javascript* passada como parâmetro na recepção, encaminhando o objeto de evento criado pelo Sinal correspondente.
- *Gestor de Sinais*. Módulo responsável pela inicialização, término e busca de sinal baseado na interface solicitada.
- *Sinal*. Cada sinal disponibilizado pela API é gerenciado de forma centralizada, garantindo que um mesmo sinal seja compartilhado entre diferentes agentes. Um módulo de sinal é responsável por apenas consumir recursos quando

solicitado, e ficar em estado de repouso enquanto não existir um receptor associado ao mesmo.

- *Broadcast Android*. Módulo representando a funcionalidade de mensagens entre aplicativos do *Android*. É amplamente utilizado para consumo dos serviços de localização, posicionamento, rede e bateria da plataforma *Android*. Também é utilizado para receber informações dos demais aplicativos que compõem o projeto Desktop Semântico. É ilustrado contendo os eventos da plataforma *Android* e o evento *AppDispatcher*, emitido pelo aplicativo de Desktop Semântico.
- *Serviço UPnP*. Responsável por iniciar o serviço do *Cling* para busca dispositivos oferecendo o serviço de Agentes JS. Ao encontrar um dispositivo com o serviço, envia a mensagem solicitando os detalhes dos agentes e repassa para o módulo Gestor de Agentes.
- *Serviço de Comunicação HTTP*. Módulo responsável por gerenciar a abertura de conexões HTTP para evitar o excesso de conexões simultâneas. Cria uma fila de requisições e executa as mesmas de forma simultânea baseado em limite de threads. Em caso de perda de conectividade guarda as requisições para reiniciar o processo de execução assim que a conexão é reestabelecida.
- *Atualizador de Contexto*. Módulo responsável por montar periodicamente o objeto de contexto conforme os sinais do dispositivo móvel e enviar para o histórico de contextos na nuvem, conforme formato e serviço definido na seção 4.3.6.
- *Controlador de Dispositivo Inteligente*. Um controlador para dispositivos inteligentes que suportem o protocolo de descoberta de agentes sugerido, implementado neste trabalho apenas como exemplo de funcionamento. É ilustrado contendo o código de Agentes JS.
- *Histórico de Contexto na Nuvem*. Componente de histórico de contextos descrito na seção 4.3.

#### 4.4.3 A API Agente JS

A API Agente JS é um conjunto de instruções definidas para permitir a construção de Agentes em *Javascript* de forma sensível a mudanças no contexto de um dispositivo móvel. Essa API é baseada em um conjunto de Componentes de API, cada qual responsável por

expor as funcionalidades e eventos relacionados a um aspecto do dispositivo móvel (como por exemplo, um componente responsável pela exposição da situação ou mudança de rede *wifi*, e outro responsável pelas mudanças na localização).

Todas as instruções da API seguem o padrão ilustrado na Figura 27, composto pelo espaço de nomes da API (*agent*), o componente de API responsável por tratar a instrução (*componente*), a instrução desejada (*instrução*) e os parâmetros esperados pela instrução (*parâmetros...*).

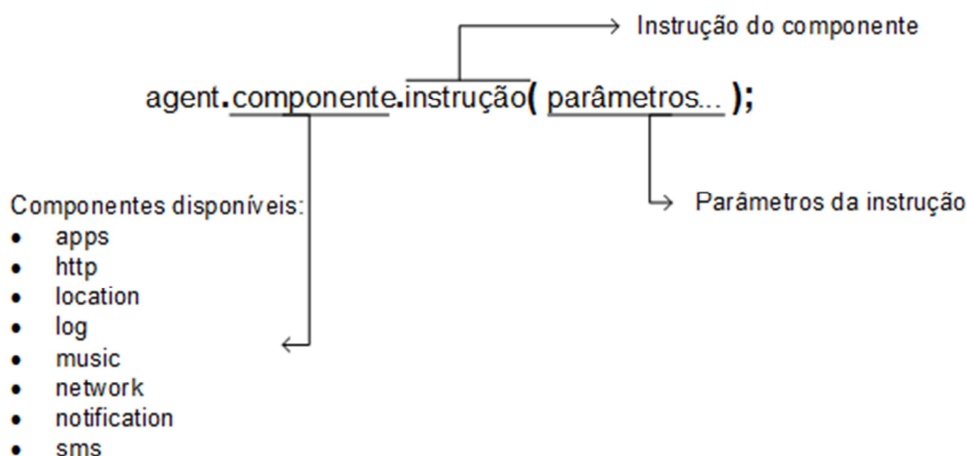


Figura 27 - Estrutura das instruções da API Agente JS

O diagrama de classes ilustrado na Figura 28 apresenta a estrutura da implementação dos componentes de API omitindo os métodos específicos que serão detalhados mais adiante nesta seção. Essa estrutura foi projetada de forma a facilitar a disponibilização de novos receptores de sinais a partir de uma implementação única das instruções *on* e *off*. É importante ressaltar que a classe *AgentAPI* representa o espaço de nomes da API, e portanto é uma composição de componentes de API retornando cada componente disponível, apenas os componentes ali listados aparecem como disponíveis para uso. A interface *IAgentAPIComponent* tem por responsabilidade representar o método utilizado pelo módulo de execução de agentes para criação de instâncias da classe auxiliar utilizada para a chamada de funções *Javascript* a partir de outros pontos do aplicativo que não o próprio executor. A classe abstrata *AbstractAgentAPIComponent* implementa os métodos de registro e desligamento dos receptores de sinais nos componentes da API, e o método abstrato *isOwnSignal* deve ser implementado pelo componente da API de forma a só permitir que a instrução *on* funcione para o componente quando o sinal for pertinente ao componente.

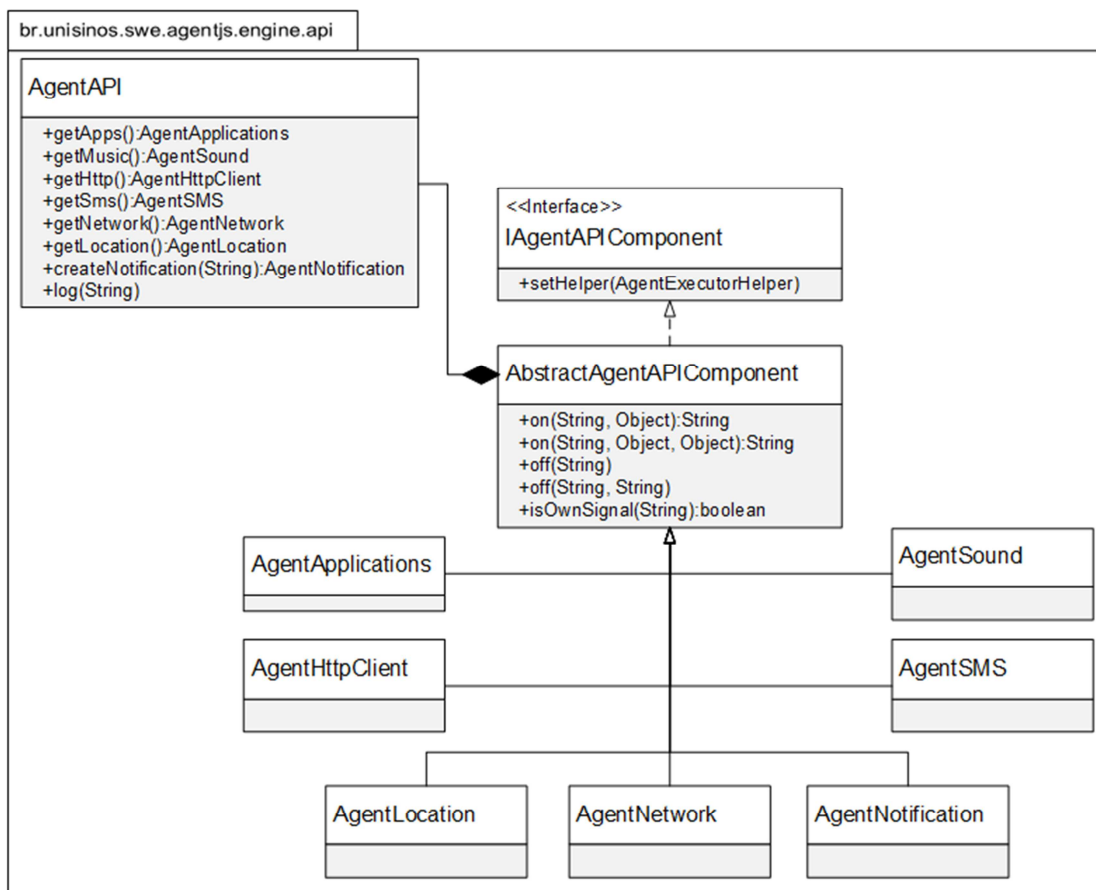


Figura 28 – Diagrama de classe dos componentes da API

A versão atualmente definida e implementada da API tem suporte as seguintes funcionalidades para o desenvolvimento de Agentes JS:

- a) Habilidade de iniciar aplicativos baseado em nome ou na URL do conteúdo;
- b) Habilidade de listar aplicativos instalados;
- c) Habilidade de realizar requisições HTTP e receber a resposta;
- d) Habilidade de identificar mudanças na posição geográfica do dispositivo móvel;
- e) Habilidade de identificar a proximidade do dispositivo móvel a uma região geográfica;
- f) Habilidade de identificar a proximidade do dispositivo móvel a um estabelecimento baseado em palavras-chave, realizando a busca de estabelecimentos a partir da API do *Google Places*.
- g) Habilidade de identificar mudanças de conectividade *wifi* (IEEE 802.11);
- h) Habilidade de criar notificações no dispositivo;
- i) Habilidade de ler e enviar mensagens SMS;

- j) Habilidade de controlar o aplicativo de música para iniciar, parar ou trocar as músicas em reprodução;
- k) Habilidade de emitir sons pelo dispositivo móvel.

A Tabela 6 descreve o funcionamento das instruções compartilhadas por todos os componentes de API. Os sinais e o detalhe dos objetos retornados para as funções de retorno são detalhados nas tabelas dos respectivos componentes.

Tabela 6 - Instruções compartilhadas entre componentes

Componente de API: Componente Base (classe <i>AbstractAgentAPIComponent</i> )	
Instruções	
<b>Instrução</b>	<p><code>on(String signal, Object callback):String listenerId</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para criação de receptores.</p> <p>Recebe como parâmetro o nome do sinal e a função <i>Javascript</i> que será chamada assim que o sinal for disparado.</p> <p>O retorno é o código identificador do receptor criado.</p> <p>É a função que deve ser chamada para os sinais detalhados em cada componente de API.</p>
<b>Instrução</b>	<p><code>on(String signal, Object options, Object, callback): String listenerId</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução padrão para criação de receptores com opções.</p> <p>Recebe como parâmetro o nome do sinal, o objeto <i>Javascript</i> com as opções ou filtros e a função <i>Javascript</i> que será chamada assim que o sinal for disparado.</p> <p>O retorno é o código identificador do receptor criado.</p> <p>É a função que deve ser chamada para os sinais com filtros ou opções, detalhados em cada componente de API.</p>
<b>Instrução</b>	<p><code>off(String signal)</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução padrão para remoção de receptores.</p> <p>Remove todos os receptores deste agente para o sinal indicado.</p>
<b>Instrução</b>	<code>off(String signal, String listenerId)</code>

	<p><b>Funcionalidade:</b></p> <p>Instrução padrão para remoção de receptor baseado no identificador retornado pela instrução <i>on</i>.</p> <p>Remove todos os receptores deste agente para o identificador indicado.</p>
--	---

A classe *AgentSound* é responsável pela implementação do componente da API exposto como *music*, cujo objetivo é permitir que um agente utilize o sistema de mídia do dispositivo para emitir sons. A Tabela 7 descreve os sinais e instruções disponíveis para o componente.

Tabela 7 - Instruções do componente de API *music*

Componente de API: <i>music</i> (classe <i>AgentSound</i> )	
Instruções	
<b>Instrução</b>	<p><code>playFromUrl(String url)</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para emitir um som a partir da URL informada. A procura do arquivo de mídia será feita no cartão SD do dispositivo se o caminho for relativo (<code>/musica/exemplo.mp3</code>).</p>
<b>Instrução</b>	<p><code>next()</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para trocar para a próxima música da lista se o aplicativo de mídia estiver em uso.</p>
<b>Instrução</b>	<p><code>pause()</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução que solicita a parada da reprodução de música, se o aplicativo de mídia estiver em uso.</p>
<b>Instrução</b>	<p><code>previous()</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para trocar para a música anterior da lista se o aplicativo de mídia estiver em uso.</p>
<b>Instrução</b>	<p><code>play()</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para continuar a reprodução de música se o aplicativo de mídia estiver em uso.</p>

<b>Instrução</b>	<code>stop()</code>
	<b>Funcionalidade:</b> Instrução para terminar a reprodução de música se o aplicativo de mídia estiver em uso.

A classe *AgentSMS* é responsável pela implementação do componente da API exposto como *sms*, cujo objetivo é permitir que um agente envie mensagens via SMS ou receba um sinal quando um SMS é recebido pelo dispositivo. A Tabela 8 descreve as instruções disponíveis para o componente.

Tabela 8 - Instruções do componente de API *sms*

Componente de API: <i>sms</i> (classe <i>AgentSMS</i> )	
Sinais	
<b>Sinal</b>	<b>Código:</b> <code>sms:income</code>
	<b>Evento:</b> É disparado toda vez que um SMS é recebido pelo dispositivo móvel
	<b>Retorno:</b> A função de retorno recebe objeto do tipo <i>SmsSignalInfo</i>
	<b>Filtros:</b> <code>origin</code> : Filtra por número de origem do sms
	<b>Exemplo:</b> <pre> 1 // Filtro por sms recebidos do número 5549 2 agent.sms.on("sms:income", { 'origin' : '5549' }, function(smsInfo) { 3     agent.log("sms recebido de 5549"); 4     agent.log(smsInfo.getMessage()); 5 }); 6 7 // Todos os sms recebidos 8 agent.sms.on("sms:income", function(smsInfo) { 9     agent.log(smsInfo.getOriginAddress() + ":" + smsInfo.getMessage()); 10 }); </pre>
Instruções	
<b>Instrução</b>	<code>send(Object message)</code>
	<b>Funcionalidade:</b> Instrução para envio de SMS pelo agente.
	<b>Formato do parâmetro <i>message</i>:</b> Objeto JSON com o seguinte formato: {



	<pre>"destination" : "número de destino", "message" : "mensagem" }</pre>
	<p><b>Exemplo:</b></p> <pre>1 // mensagem para o numero 5549 2 var mensagem = { 3   "destination" : "5549", 4   "message" : "conteudo do sms" 5 }; 6 agent.sms.send(mensagem);</pre>

A classe *AgentNotification* é responsável pela implementação do componente da API exposto como *notification*, cujo objetivo é permitir que um agente crie notificações no dispositivo móvel. A Tabela 9 descreve as instruções disponíveis para o componente.

Tabela 9 - Instruções do componente de API *notification*

Componente de API: <i>notification</i> (classe <i>AgentNotification</i> )	
Instruções	
<b>Instrução</b>	send(Object notification)
	<p><b>Funcionalidade:</b></p> <p>Instrução para envio de notificações pelo agente.</p>
	<p><b>Formato do parâmetro <i>notification</i>:</b></p> <p>Objeto JSON com o seguinte formato:</p> <pre>{   "title" : "número de destino",   "content" : "mensagem",   "vibrate": true,   "sound": "/musica/som.mp3" }</pre> <p>Os atributos <i>title</i> e <i>content</i> são obrigatórios.</p> <p>O atributo <i>vibrate</i> define se o dispositivo deve vibrar com a notificação.</p> <p>O atributo <i>sound</i> altera o toque emitido pela notificação.</p>
	<p><b>Exemplo:</b></p> <pre>1 // Nova notificação 2 var notif = { 3   "title" : "Titulo de notificação", 4   "content" : "conteudo da notificação" 5 }; 6 agent.notification.send(notif);</pre>

A classe *AgentNetwork* é responsável pela implementação do componente da API exposto como *network*, cujo objetivo é permitir que um agente observe eventos de rede *wifi* e de telefonia. A Tabela 10 descreve os sinais disponíveis para o componente.

Tabela 10 - Instruções do componente de API *network*

Componente de API: <i>network</i> (classe <i>AgentNetwork</i> )	
Sinais	
<b>Sinal</b>	<b>Código:</b> <code>wifi:on</code>
	<b>Evento:</b> É disparado toda vez que o <i>wifi</i> (IEEE 802.11) do dispositivo móvel é ligado.
	<b>Retorno:</b> A função de retorno não recebe nenhum objeto, uma vez que não existe informação de conectividade <i>wifi</i> .
<b>Sinal</b>	<b>Código:</b> <code>wifi:off</code>
	<b>Evento:</b> É disparado toda vez que o <i>wifi</i> do dispositivo móvel é desligado.
	<b>Retorno:</b> A função de retorno não recebe nenhum objeto, uma vez que não existe informação de conectividade <i>wifi</i> .
<b>Sinal</b>	<b>Código:</b> <code>wifi:scan</code>
	<b>Evento:</b> É disparado pelo adaptador <i>wifi</i> com a lista dos <i>hotspots</i> disponíveis após cada operação de busca.
	<b>Retorno:</b> A função de retorno recebe um <i>array</i> de objetos do tipo <i>ScanResultInfo</i> .
<b>Sinal</b>	<b>Código:</b> <code>wifi:connected</code>
	<b>Evento:</b> É disparado toda vez que o adaptador <i>wifi</i> conecta a um <i>hotspot</i> .

	<p><b>Retorno:</b></p> <p>A função de retorno recebe dois objetos de retorno: (i) objeto do tipo <i>NetworkSignalBasicInfo</i>, e (ii) objeto do tipo <i>WifiSignalBasicInfo</i>.</p> <p>O primeiro objeto possui informações gerais sobre a conectividade, e o segundo objeto informações específicas sobre a rede <i>wifi</i>.</p>
<b>Sinal</b>	<p><b>Código:</b></p> <p><code>wifi:disconnected</code></p>
	<p><b>Evento:</b></p> <p>É disparado toda vez que o adaptador <i>wifi</i> desconecta a um <i>hotspot</i>.</p>
	<p><b>Retorno:</b></p> <p>A função de retorno recebe um objeto do tipo <i>NetworkSignalBasicInfo</i> com informações sobre o estado de conectividade do dispositivo móvel (como por exemplo, se está utilizando o 3G).</p>

A classe *AgentLocation* é responsável pela implementação do componente da API exposto como *location*, cujo objetivo é permitir que um agente observe eventos de deslocamento geográfico, proximidade de pontos geográficos e de proximidade de estabelecimentos. A Tabela 11 descreve os sinais e instruções disponíveis para o componente.

Tabela 11 - Instruções do componente de API *location*

Componente de API: <i>location</i> (classe <i>AgentLocation</i> )	
Sinais	
<b>Sinais</b>	<p><b>Códigos:</b></p> <p><code>query:enter</code> ou <code>query:exit</code></p>
	<p><b>Evento:</b></p> <p>É disparado quando o dispositivo se aproxima ou se afasta de algum local encontrado pela API do <i>Google Places</i> conforme as características definidas pelo filtro.</p>
	<p><b>Filtros:</b></p> <p><code>name</code>: Nome do estabelecimento</p> <p><code>keyword</code>: Palavra-chave relacionada ao estabelecimento</p> <p><code>type</code>: Tipo do estabelecimento conforme padrão da API do <i>Google Places</i></p> <p><code>distance</code>: A distância para que o alerta de proximidade seja disparado.</p>

	<p><b>Retorno:</b></p> <p>Uma instância da classe <i>PlaceSignalInfo</i> contendo as informações sobre o estabelecimento mais próximo informado pela API do <i>Google Places</i>.</p> <p><b>Exemplo:</b></p> <pre> 1 // Alerta de proximidade quando um Pizzaria estiver a menos de 300 metros. 2 var myQuery = { 3   'keyword' : 'Pizzaria', 4   'distance': '300' 5 }; 6 7 var myFuncnt = function(place) { 8   agent.log(place.name + " @ " + place.address + " - distance: " + place.distance); 9 }; 10 11 agent.location.on('query:enter', myQuery, myFuncnt); </pre>
<p><b>Sinais</b></p>	<p><b>Códigos:</b></p> <p>region:enter OU region:exit</p> <p><b>Evento:</b></p> <p>São sinais análogo ao query:enter ou query:exit, com a diferença de realizar o disparo baseado em coordenadas geográficas ao invés de uma busca por lugares, e portanto não tem dependência com nenhuma forma de conectividade do dispositivo.</p> <p><b>Filtros:</b></p> <p>longitude: Longitude decimal do ponto desejado</p> <p>latitude: Latitude decimal do ponto desejado</p> <p>radius: Raio em metros da região para disparo do sinal</p> <p><b>Retorno:</b></p> <p>Não existe objeto para a função de retorno.</p> <p><b>Exemplo:</b></p> <pre> 1 // Notificação ao chegar a 100 metros da entrada da Unisinos 2 var entradaUnisinos = { 3   'longitude': '-51.147578', 4   'latitude': '-29.779965', 5   'radius': '100' // em metros 6 }; 7 8 var myCallbackId = null; 9 10 var onEnter = function() { 11   agent.log("Unisinos a 100 metros"); 12   agent.location.off('region:enter', myCallbackId); // remove o receptor pelo id 13 14 }; 15 16 var myCallbackId = agent.location.on('region:enter', entradaUnisinos, onEnter); </pre>
<p><b>Sinal</b></p>	<p><b>Código:</b></p> <p>location:changed</p>

	<p><b>Evento:</b></p> <p>É disparado toda vez que o dispositivo se move pelo menos 30 metros, ou tenha se passado pelo menos 2 segundos desde o ultimo disparo.</p>
	<p><b>Retorno:</b></p> <p>Uma instância da classe <i>LocationSignalInfo</i> contendo as informações sobre a posição do dispositivo naquele momento.</p>
<b>Instruções</b>	
<b>Instrução</b>	<code>lastLocation():LocationSignalInfo</code>
	<p><b>Funcionalidade:</b></p> <p>Instrução para recuperar a ultima posição conhecida pelo dispositivo. Esta posição pode estar desatualizada, mas em compensação não consome excessivamente a bateria do dispositivo.</p>
	<p><b>Exemplo:</b></p> <pre> 1 // Envio de SMS informando onde o dispositivo está ao receber um SMS do número 2 // 99009900 com a mensagem "onde?" 3 agent.sms.on("sms:income", { 'origin' : '99009900' }, function(smsInfo) { 4     if(smsInfo.getMessage() == "onde?") { 5         var lugar = agent.location.lastLocation(); // pega o ultimo lugar conhecido 6         var mensagem = "Estou em http://maps.google.com/?q=" + lugar.lat + "," + lugar.lon; 7 8         agent.sms.send( 9             { 10                "destination" : "99009900", 11                "message" : mensagem 12            }); 13     } 14 }); </pre>

A classe *AgentHttpClient* é responsável pela implementação do componente da API exposto como *http*, cujo objetivo é permitir que um agente realize requisições HTTP a partir do dispositivo móvel. A Tabela 12 descreve as instruções disponíveis para uso no componente.

Tabela 12 - Instruções do componente de API *http*

<b>Componente de API: <i>http</i> (classe <i>AgentHttpClient</i>)</b>	
<b>Instruções</b>	
<b>Instrução</b>	<code>get(Object httpRequest, Object success, Object error)</code>
	<p><b>Funcionalidade:</b></p> <p>Instrução para execução de uma requisição HTTP get.</p> <p>O parâmetro <code>httpRequest</code> deve ser um objeto JSON contendo o atributo <code>url</code>, cujo valor é o destino da requisição.</p>

	<p>O parâmetro <code>success</code> deve ser uma função que receberá como entrada uma String com o conteúdo do corpo da resposta em caso de sucesso, e o parâmetro <code>error</code> deve ser uma função que será chamada em caso de erro na requisição.</p> <p><b>Formato do parâmetro <i>httpRequest</i>:</b></p> <p>Objeto JSON com o seguinte formato:</p> <pre>{   "url" : "url para a requisição" }</pre> <p><b>Exemplo:</b></p> <pre> 1 2  var weatherAPI = "http://free.worldweatheronline.com/feed/weather.ashx?q=5%c3%a3o+Leopoldo,Brazil&amp;form 3  var httpRequest = { "url" : weatherAPI }; 4 5  var onSuccess = function(sResponse) { 6    var weather = JSON.parse(sResponse); 7 8    agent.log("Temperatura fora: " + weather.data.current_condition.temp_C); 9  }; 10 11 var onError = function() { 12   agent.log("API offLine"); 13 }; 14 15 agent.http.get(httpRequest, onSuccess, onError);</pre>
<b>Instrução</b>	<p><code>ajax(Object httpRequest, Object success, Object error)</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para execução de uma requisição HTTP com qualquer método desejado.</p> <p>O parâmetro <code>httpRequest</code> deve ser um objeto JSON contendo três atributos: <i>(i)</i> <code>url</code> com o endereço de destino da requisição, <i>(ii)</i> <code>method</code> com o método HTTP desejado e <i>(iii)</i> <code>data</code> com uma String com os dados que devem ser enviados no corpo da requisição. Se <code>data</code> for nulo, nada será enviado no corpo requisição.</p> <p>O parâmetro <code>success</code> deve ser uma função que receberá como entrada uma String com o conteúdo do corpo da resposta em caso de sucesso, e o parâmetro <code>error</code> deve ser uma função que será chamada em caso de erro na requisição.</p> <p><b>Formato do parâmetro <i>httpRequest</i>:</b></p> <p>Objeto JSON com o seguinte formato:</p> <pre>{   "url" : "url para a requisição",   "method" : "método da requisição",   "data" : "dados para o corpo da requisição" }</pre>

A classe *AgentApplications* é responsável pela implementação do componente da API exposto como *apps*, cujo objetivo é permitir que um agente observe eventos de inicialização de aplicativos a partir do Desktop Semântico, liste os aplicativos disponíveis ou inicialize novos aplicativos quando necessário. A Tabela 13 descreve os sinais e instruções disponíveis para o componente.

Tabela 13 - Instruções do componente de API *apps*

Componente de API: <i>apps</i> (classe <i>AgentApplications</i> )	
Sinais	
<b>Sinal</b>	<p><b>Código:</b> home:app:started</p> <p><b>Evento:</b> É disparado quando o usuário abre um aplicativo a partir do Desktop Semântico.</p> <p><b>Retorno:</b> Uma instância da classe <i>AppDispatcherSignalInfo</i> contendo as informações sobre o aplicativo escolhido, como nome, pacote e número de vezes que foi inicializado.</p> <p><b>Exemplo:</b></p> <pre> 1 // Abre a agenda de contatos 2 var onAppStarted = function(appDispatcherInfo) { 3     agent.log( 4         "Aplicativo " + 5         appDispatcherInfo.title + 6         " inicializado " + 7         appDispatcherInfo.executionCount + 8         " vezes" ); 9 }; 10 11 agent.apps.on('home:app:started', onAppStarted); </pre>
Instruções	
<b>Instrução</b>	<p>launchAppForUrl(String url, String mime)</p> <p><b>Funcionalidade:</b> Instrução para abrir um aplicativo para a URL especificada. Se o parâmetro <i>mime</i> for nulo, seu valor será inferido a partir da URL.</p>
<b>Instrução</b>	<p>launchAppByPackage(String packageName)</p> <p><b>Funcionalidade:</b> Instrução para abrir um aplicativo a partir do nome do pacote <i>Android</i> que identifica o aplicativo.</p>

	<p><b>Exemplo:</b></p> <pre>1 // Abre a agenda de contatos 2 agent.apps.launchAppByPackage('com.android.contacts'); 3</pre>
<b>Instrução</b>	<p><code>getInstalledApps():ApplicationInfo[]</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para receber uma lista de todos os aplicativos instalados no dispositivo móvel. O retorno é um <i>array</i> de objetos da classe <i>ApplicationInfo</i>.</p>
<b>Instrução</b>	<p><code>findAppByName(String appName):ApplicationInfo</code></p> <p><b>Funcionalidade:</b></p> <p>Instrução para receber os detalhes da instalação de um aplicativo a partir do nome. Retorna um objeto da classe <i>ApplicationInfo</i> se encontrar o aplicativo instalado no dispositivo, ou nulo se não encontrar.</p>

#### 4.4.4 Emissor de sinal

Esta seção visa descrever o funcionamento das classes relacionadas com a emissão de sinal. No escopo deste trabalho um sinal é uma mensagem assíncrona enviada a todos os agentes registrados assim que um evento acontece. Estes eventos estão em geral relacionados a atividades relativas ao contexto do dispositivo móvel, como a localização ou o estado de conectividade. O digrama de classe ilustrado na Figura 29 apresenta as classes e métodos envolvidos na emissão de sinal.

A classe *SignalsManager* é a classe responsável pela gestão dos sinais e é nela que cada sinal é registrado uma vez, o que permite a exposição e compartilhamento do sinal entre os demais módulos do motor de execução. A *SignalsManager* implementa a interface *IEngineComponent* por ser parte dos componentes do motor de execução (e portanto acessível a partir do motor de execução), e a interface *ISignalManager* que é utilizada por outras classes para acesso as funcionalidades de busca de sinais e de acesso a um sinal específico.

A classe *SignalListener* é a implementação padrão da interface de receptor de sinal, e é instanciada pelo componente de API toda vez que um agente solicita a recepção de um sinal. Essa classe implementa os métodos necessários para enviar o disparo de um sinal para o agente e para permitir que os sinais acessem os filtros definidos pelo agente durante o registro para recepção de sinal.



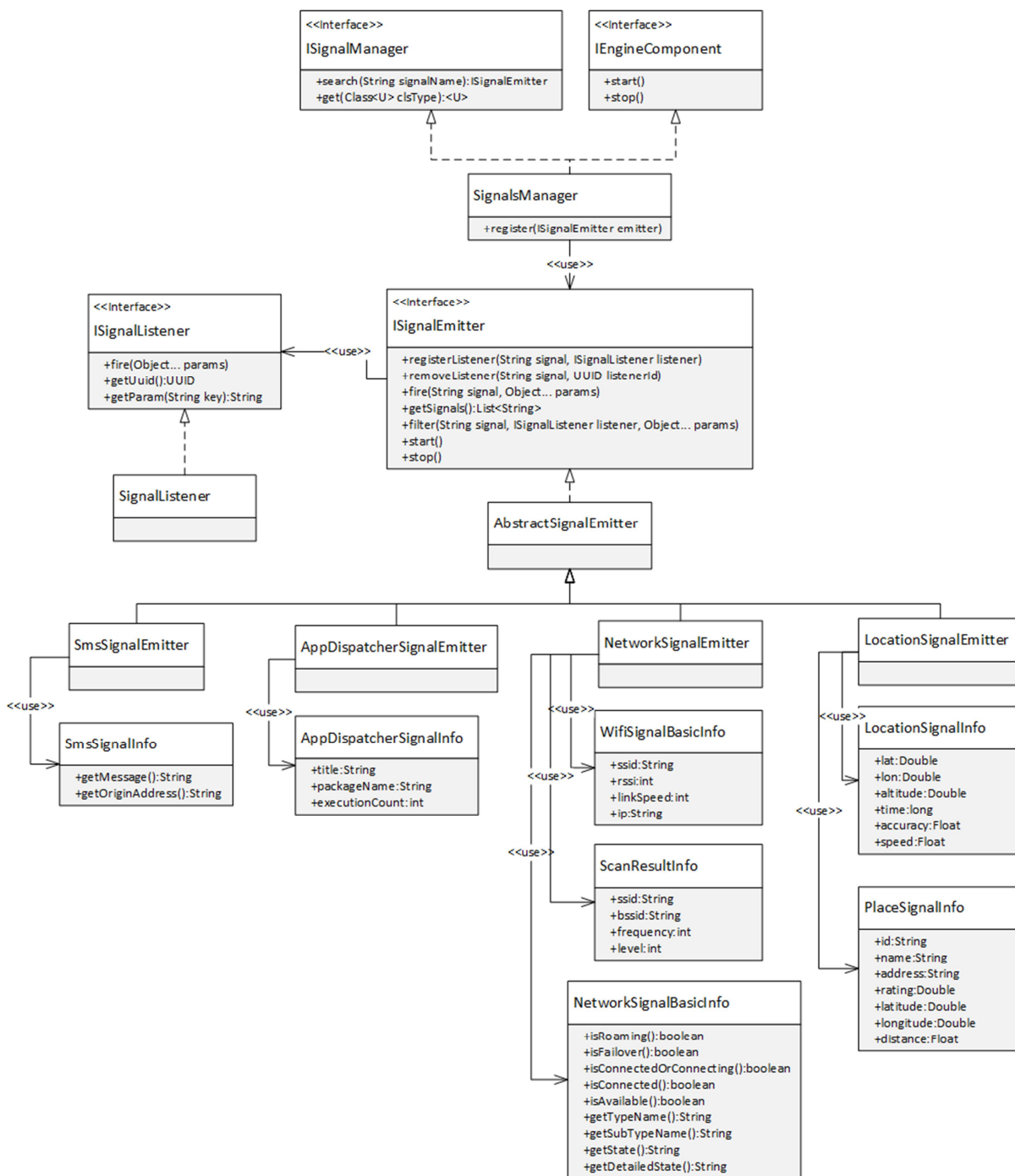


Figura 29 - Diagrama das classes de emissão de sinal

A classe *AbstractSignalEmitter* é a implementação base de todos os sinais, e implementa uma forma comum de lidar com o registro ou remoção de receptores (instâncias de *SignalListener*) e com a chamada do disparo para a classe *SignalListener* registrada para o sinal em questão. Essa abordagem facilita implementação de novos sinais, permitindo que a classe do sinal se preocupe apenas com as funcionalidades específicas do sinal que será disponibilizado e não com o código necessário para o envio de informações para os receptores.

A classe *SmsSignalEmitter* é a implementação do emissor de sinais relacionados a SMS. Ela solicita para plataforma *Android* o recebimento todos os broadcasts relacionados com a ação *android.provider.Telephony.SMS\_RECEIVED*, o que permite a leitura de todas as mensagens SMS recebidas pelo dispositivo móvel. O retorno do sinal de SMS (*sms:income*) é uma instância de *SmsSignalInfo* que é uma classe de encapsulamento e exposição de atributos da classe *SmsMessage* da plataforma *Android*, expondo o acesso a mensagem e ao endereço de origem da mensagem.

Por questão de segurança e padronização, todas as instâncias de classe que os agentes têm acesso residem em pacotes do motor de execução. Isso permite implementações em outras plataformas seguindo uma exposição com os mesmos nomes de atributos e o controle sobre o que será disponibilizado para o agente.

A classe *NetworkSignalEmitter* é a implementação do emissor de sinais relacionados a conectividade. Ela utiliza os serviços de *wifi* e conectividade da plataforma *Android* para identificar mudanças no estado da rede, e reportar como um sinal para os agentes registrados. Essa classe é responsável pela implementação de cinco sinais: (*i, wifi:on*) sinal emitido após operação de ligar o adaptador *wifi* do dispositivo, (*ii, wifi:off*) sinal emitido após operação de desligar o adaptador *wifi*, (*iii, wifi:scan*) sinal de resultado da operação de busca de *hotspots* pelo adaptador *wifi*, cujo retorno é um *array* de *ScanResultInfo* (classe de encapsulamento da classe *ScanResult* da plataforma *Android*), (*iv, wifi:connected*) sinal emitido após a operação de estabelecimento de conexão com um *hotspot wifi*, cujo retorno é uma instância de *NetworkSignalBasicInfo* (classe de encapsulamento de *NetworkInfo* da plataforma *Android*) e uma instância de *WifiSignalBasicInfo* (classe de encapsulamento de *WifiInfo* da plataforma *Android*), e (*v, wifi:disconnected*) sinal emitido após a operação de término da conexão com um *hotspot wifi*, que retorna uma instância de *NetworkSignalBasicInfo*.

A classe *LocationSignalEmitter* é a implementação do emissor de sinais relacionados a localização. Ela utiliza a classe *LocationManager* da plataforma *Android* para criar alertas de proximidade ou alertas de alteração de localização quando solicitado por um agente JS, e o serviço de busca de estabelecimentos do *Google Places* para o funcionamento do sinal de proximidade de estabelecimentos.

Os dois sinais relacionados a busca de estabelecimentos (sinal *query:enter* e sinal *query:exit*) são implementados seguindo a seguinte lógica: é realizada uma requisição HTTP para o serviço de busca de locais assim que um receptor é solicitado por um agente JS, a resposta desta requisição vai definir onde os alertas de proximidade devem ser criados e o tamanho do raio de resultados. Quando o dispositivo móvel se aproximar de qualquer um dos

lugares sugeridos, o disparo do evento ocorre, repassando as informações do local (uma instância da classe *PlaceSignalInfo*) para o agente que criou o receptor.

Se o dispositivo móvel sair do raio de resultados (ou seja, se o ultimo resultado ficar a 2 km da posição inicial e o dispositivo se mover além de 2 km dessa posição inicial) uma nova requisição HTTP para o serviço de busca de locais é realizada, a lista de alertas de proximidade criados anteriormente é excluída e novos alertas de proximidade são definidos de acordo com o novo resultado da busca.

Os demais sinais relacionados a localização fazem uso de funcionalidades existentes no *LocationManager*, como no caso do sinal de mudança de localização (*location:changed*) que solicita que informações sobre mudanças relacionadas a localização sejam repassadas para a classe *LocationSignalEmitter* a cada dois segundos ou após o deslocamento superior a trinta metros, e essa informação é então encaminhada para todos os receptores em uma instância da classe *LocationSignalInfo*.

Os sinais de proximidade de uma região geográfica (sinal *region:enter* e sinal *region:exit*) utilizam os alertas de proximidade do *LocationManager* para disparar o sinal conforme solicitado por um agente JS.

A classe *AppDispatcherSignalEmitter* é a implementação do emissor de sinais relacionados ao Desktop Semântico. Ela solicita para plataforma *Android* o recebimento todos os broadcasts relacionados com a ação *com.unisinos.AppDispatcher*, o que permite o recebimento de informações sobre o aplicativo que o usuário selecionou para execução a partir de três *extras* disponibilizadas no *broadcast*: (i) o parâmetro *App.Title* que contem o nome do aplicativo escolhido, (ii) *App.PackageName* que contem o pacote do aplicativo escolhido, e (iii) *App.ExecutionCount* que contem a contagem do número de vezes que o aplicativo foi escolhido para execução. O retorno do sinal de inicialização de aplicativo (*home:app:started*) é uma instância de *AppDispatcherSignalInfo* que contem um atributo para cada *extra* do *broadcast*, respectivamente atributo *title*, atributo *packageName* e atributo *executionCount*.

#### 4.4.5 Controle de execução de agentes

Neste trabalho, cada agente JS é interpretado em uma *thread* própria, o que permite limitar falhas e erros no código do agente ao seu próprio escopo, e portanto sem afetar a estabilidade dos demais agentes. Uma das responsabilidades do módulo gestor de agentes (classe *AgentScriptManager*) é tratar o comando de execução do agente JS, instanciando um

executor de agentes (classe *AgentExecutor*) em uma *thread* separada da *thread* principal do motor de execução. O executor de agentes por sua vez realiza quatro passos em sequência: (i) cria um novo contexto para o agente JS, (ii) realiza uma cópia o escopo padrão do motor de execução, (iii) adiciona uma nova instância de componentes de API no escopo criado e (iv) realiza a interpretação do código do agente.

Quando se utiliza o *Rhino* para interpretação de *Javascript* em um ambiente concorrente, é importante entender dois conceitos: contextos e escopos.

Um contexto no *Rhino* é utilizado para armazenar informações específicas de uma *thread* sobre o ambiente de execução, e só um contexto deve estar associado a cada *thread*, o que no caso deste trabalho se traduz em um contexto *Rhino* por execução de agente.

Um escopo no *Rhino* é um conjunto de objetos *Javascript* que representam os objetos e funções disponíveis para utilização do script que vai ser interpretado. A inicialização de um escopo *Rhino* é custosa em termos de processamento e alocação de memória, e portanto o escopo que representa instruções comuns (como a interpretação de objetos JSON) é criado apenas uma vez, e compartilhado entre os contextos dos diversos agentes.

Neste trabalho foi utilizado um conceito de *Sandbox* de contexto devido ao potencial risco de acesso às classes nativas da plataforma *Android* que existe ao se interpretar código *Javascript* sem necessariamente saber a origem combinado com a característica nativa do *Rhino* de expor toda e qualquer classe Java para o *Javascript*.

O *Sandbox* de contexto é um ambiente controlado e limitado, onde o acesso a uma classe Java a partir do *Javascript* só é permitida se o código Java permitir o acesso de forma explícita, e portanto o acesso a qualquer classe que não está na lista de classes permitidas é negado. Outra responsabilidade do *Sandbox* é garantir que o tempo de resposta de um agente seja inferior a sessenta segundos. Agentes cuja instrução síncrona demore mais de sessenta segundos serão terminados e os recursos da *thread* correspondente liberados.

No diagrama de classes da Figura 30 estão ilustradas as classes relacionadas a execução de agentes JS, e parte dos métodos relevantes a execução. É possível verificar a existência de duas classes representando scripts de agente JS. Uma classe base (*AgentScript*) que é instanciada para agentes existentes no cartão SD do dispositivo (copiados do histórico de contexto na nuvem), e a classe *AgentNetworkScript*, que representa agentes encontrados via descoberta automática (UPnP). Essa especialização da *AgentNetworkScript* existe para permitir que scripts encontrados em uma rede local mantenham a rastreabilidade para o dispositivo que está disponibilizando o agente. Isso permitiria em trabalhos futuros criar uma

certificação de dispositivos, e distinguir agentes certificados de agentes com potencial malicioso. Ainda é possível destacar as seguintes classes:

- *Classe AgentExecutor*. Classe que realiza as operações relacionadas a execução do agente JS em uma *thread* própria.
- *Classe AgentExecutorHelper*. Classe auxiliar que permite a realização de chamadas das funções de retorno pelo emissor de sinal, no contexto e escopo específicos do agente JS.
- *Classe AgentScriptManager*. Classe responsável por solicitar a leitura de agentes da rede local, da nuvem, e do cartão SD. Também cria instâncias de *AgentExecutor* para execução de agentes JS.
- *Classe EngineScriptSandbox*. Classe de Sandbox do contexto *Rhino* explicada nesta seção.
- *Classe EngineLogger*. Classe para criação de logs tanto por agentes quando pelo motor de execução.
- *Classe EngineContext*. Classe *singleton* com referência para o *EngineLogger*, o contexto do aplicativo *Android*, o módulo gestor de sinais e as preferências do aplicativo.
- *Classe Engine*. Classe central do motor de execução, é instanciada e inicializada pelo serviço *EngineService* e responsável por criar o *EngineContext* e os principais módulos do motor (Gestor de sinais, gestor de agentes, e atualizador de contexto).

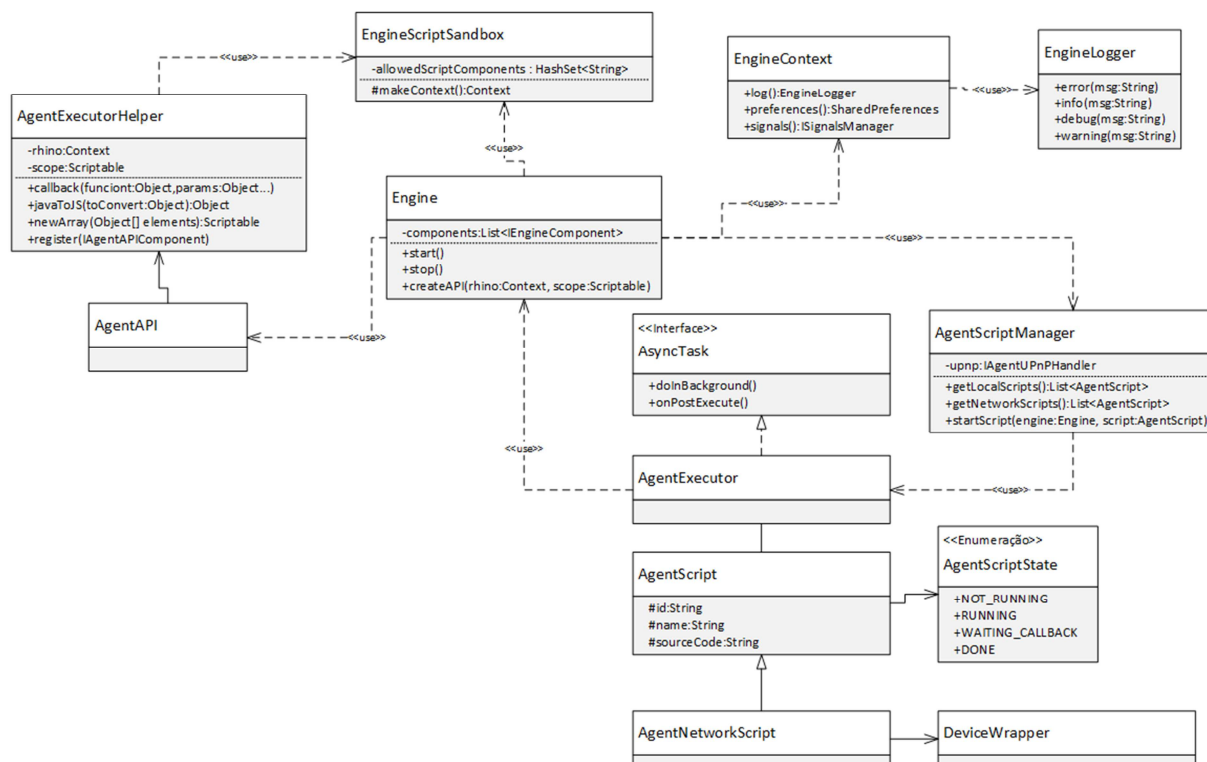


Figura 30 - Classes relacionadas a execução de agentes

#### 4.4.6 Descoberta de agentes em via UPnP

A funcionalidade de descoberta de agentes via UPnP está relacionada as características espaços inteligentes e invisibilidade de sistemas pervasivos. O objetivo deste módulo é permitir que dispositivos inteligentes tenham capacidade de expor agentes JS em uma rede local de forma a permitir que dispositivos móveis conectados a rede encontrem e utilizem estes agentes sem necessidade de nenhuma configuração adicional por parte do usuário do dispositivo.

Para permitir essa descoberta este trabalho define um formato de serviço UPnP (ilustrado na Figura 31) que um dispositivo inteligente deve expor de forma a disponibilizar agentes JS em uma rede local.

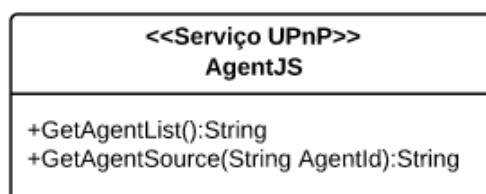


Figura 31 - Serviço UPnP para descoberta de agentes

Este serviço UPnP deve ser declarado com o tipo *AgentJS*, e implementar duas ações: (i) a ação *GetAgentList*, que deve retornar como resposta uma *String Agents* contendo um *array* de agentes JS serializados utilizando a notação JSON, que devem possuir atributos de identificação (*id*) e de nome (*name*), e (ii) a ação *GetAgentSource*, que deve receber o parâmetro de entrada *AgentId* do tipo *String*, e deve retornar como resposta uma *String Agent* contendo uma instância de agente JS serializado utilizando a notação JSON, com os atributos de identificação (*id*), nome (*name*) e código fonte (*sourceCode*).

O diagrama de classe da Figura 32 ilustra as principais classes e interfaces envolvidas na descoberta de agentes via UPnP. A interface *IAgentUPnPHandler* define os métodos que serão utilizados pela classe *AgentScriptManager* para solicitar a descoberta de agentes. A interface *IAgentUPnPListener* define os métodos que devem ser implementados pela classe que vai receber a resposta da descoberta de agentes. A classe *UPnPHandler* é a responsável por criar um serviço *Android* da biblioteca *Cling* e procurar pelo tipo de serviço UPnP "AgentJS". Todo dispositivo encontrado na rede local que possuir o serviço AgentJS será encapsulado pela classe *DeviceWrapper*, que expõe para os demais módulos do sistema o nome único do dispositivo (UDN) e o nome legível do dispositivo.

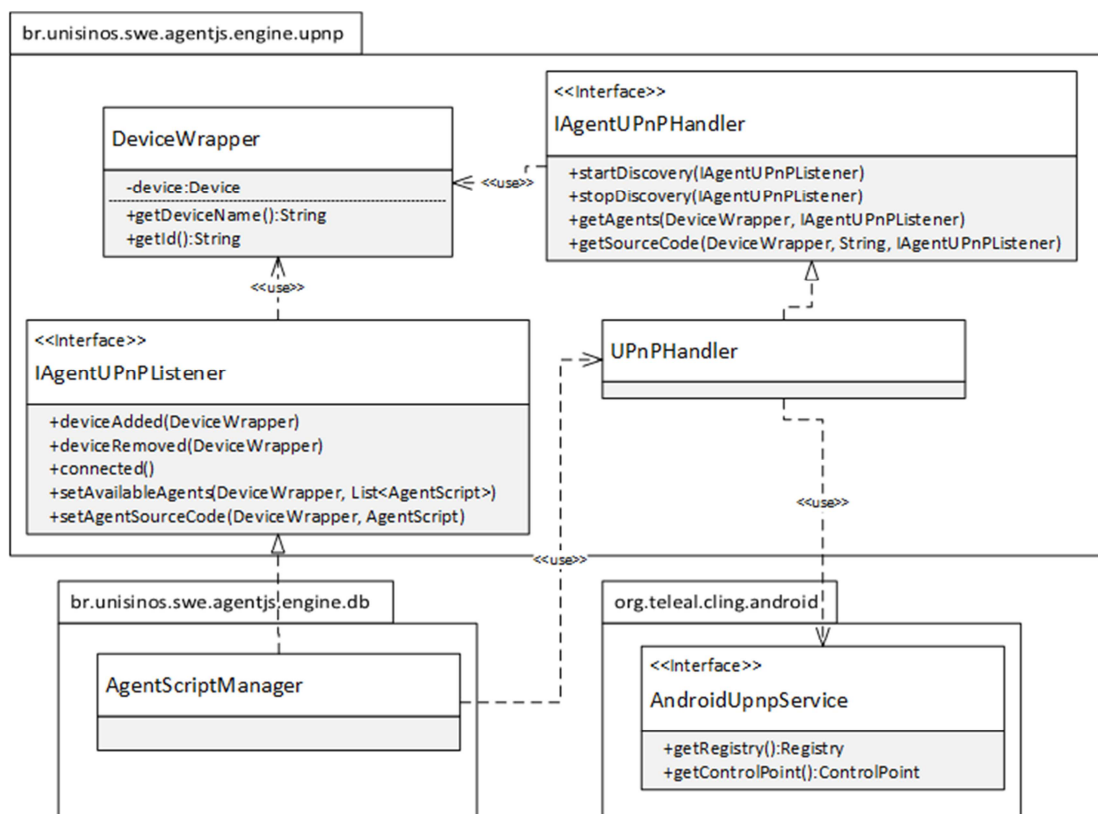


Figura 32 - Diagrama de classe do módulo de serviços UPnP

#### 4.4.7 Atualizador de contexto

O envio periódico do contexto para a nuvem é formado por apenas uma classe (*ContextUploader*) e uma interface (*IContextUploader*). Na inicialização desta classe é realizado um agendamento de execução periódica a partir do serviço de alarme da plataforma Android. Toda vez que o alarme dispara a execução, o contexto é elaborado a partir dos dados disponibilizados pelos sinais descritos na seção 4.4.4 e serializados conforme o formado na seção 4.3.6. A Figura 33 ilustra o diagrama de classes deste módulo e sua relação com os Sinais utilizados.

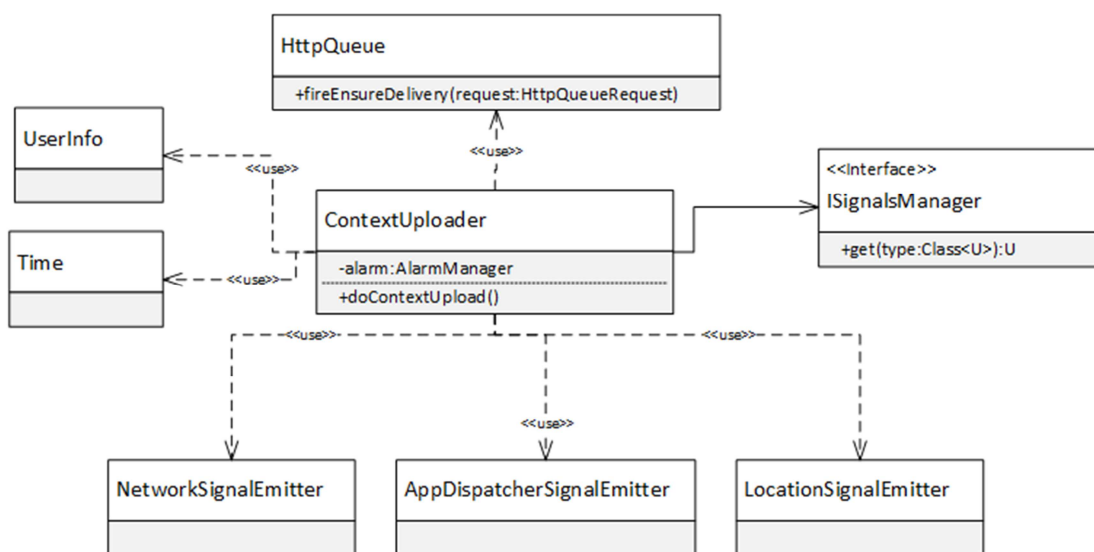


Figura 33 - Diagrama das classes envolvidas na atualização de contexto



## 5 VALIDAÇÃO E TESTES DA PLATAFORMA

As atividades de verificação e validação têm por objetivo garantir que a plataforma proposta atende aos requisitos determinados no capítulo anterior e é plenamente funcional quando inserido em um ambiente inteligente.

A próxima seção detalha o plano de como será realizada a validação da plataforma proposta por este trabalho.

### 5.1 PLANEJAMENTO DA VALIDAÇÃO E LIMITAÇÃO DA METODOLOGIA

A validação da plataforma proposta neste trabalho visa garantir que as funcionalidades (de API da plataforma e das interfaces do histórico de contexto) estão de acordo com a especificação e que a plataforma atende aos objetivos deste trabalho.

Devido à característica de ser uma plataforma para desenvolvimento de software, a validação apresentada neste capítulo tem como principal objetivo garantir que o funcionamento dos componentes e define como um trabalho futuro prioritário realizar a validação da utilidade percebida pela plataforma por desenvolvedores de software para computação pervasiva.

Esta validação foi realizada em duas etapas principais para permitir a verificação dos diferentes aspectos funcionais da plataforma: (i) ambiente virtual, onde todos os dispositivos, atividades e servidores envolvidos na arquitetura são simulados em um ambiente virtual controlado, e (ii) ambiente físico, composto por um dispositivo móvel, um dispositivo inteligente e um *cluster* de servidores na nuvem da *Amazon* para validar um cenário com dispositivos reais aplicando conceitos de sensibilidade a conceito.

### 5.2 TESTE DE FUNCIONALIDADE EM AMBIENTE VIRTUAL

O ambiente virtual para os testes de funcionalidade que também foi utilizado durante toda a etapa de desenvolvimento deste trabalho é um computador que possui as seguintes características:

- Processador Intel Core i7 2GHz;
- Memória 16GB DDR3;
- Disco rígido de 500GB 5400 RPM;

- Sistema operacional OS X 10.8.2.

Neste ambiente controlado foram instalados e configurados os seguintes servidores e dispositivos:

- Sistema de arquivos distribuído HDFS, utilizando Apache Hadoop 1.0.4, configurado de forma pseudo-distribuída (o que significa que um mesmo computador roda os processos de *NameNode* e *DataNode*);
- Banco de dados distribuído Apache HBase 0.94.4;
- Servidor de aplicação Apache Tomcat 7.0.29, inicializado a partir do ambiente de desenvolvimento Eclipse Juno utilizando o componente de *Web Tools Platform* (WTP), responsável por simular o histórico de contexto na nuvem;
- Emulador de dispositivo Android 4.0.3, com 512MB de RAM e 256MB de espaço no cartão SD virtual;
- Emulador de dispositivo inteligente virtual construído em Java, utilizando a biblioteca *Cling* para expor um dispositivo e os agentes conforme a definição de serviço UPnP.

Antes do início da validação é importante ressaltar as limitações conhecida do emulador de dispositivos *Android*. Atualmente o emulador não consegue receber *broadcasts* UDP e nem simular as conexões *wifi*. Isso significa que o dispositivo móvel emulado não pode ser utilizado para testes dos sinais de *wifi*, e que a descoberta automática de agentes só consegue listar agentes quando os mesmos já estiverem disponíveis na rede local. Agentes disponibilizados após o início da plataforma no dispositivo móvel emulado não serão descobertos, e portanto este comportamento já é esperado durante a validação em ambiente simulado.

A validação no ambiente controlado seguirá os seguintes passos:

1. Validar a persistência dos dados e modelo da ontologia no banco de dados distribuído;
2. Validar a leitura dos dados e modelo da ontologia do banco de dados distribuído;
3. Validar a leitura e criação de agentes na nuvem;
4. Validar o atualizador de contexto;
5. Validar a funcionalidade de interpretação de agentes via histórico de contexto;
6. Validar a funcionalidade de interpretação de agentes via descoberta automática;

## 7. Validar componentes da API simulando movimentação e recebimento de SMS;

O primeiro passo da validação é garantir que a biblioteca Jena carregou os modelos e um indivíduo de exemplo para o banco de dados distribuído conforme o modelo proposto. Assim que o servidor de aplicação é inicializado o mesmo realiza o procedimento de inicialização conforme descrito na seção 4.3.4, e para verificar se os dados foram carregados com sucesso é realizada uma busca diretamente no banco de dados via terminal. Essa busca é composta por dois comandos: (i) pela listagem das tabelas usando o comando *list*, que deve ter como resultado as três tabelas do modelo de armazenamento proposto, e (ii) pelo comando *get <tabela de sujeitos>, <identificador do individuo>*, que deve ter como resposta todos os predicados e objetos para o indivíduo de exemplo existente na ontologia. Conforme pode ser observado na Figura 34 o resultado desta validação foi conforme esperado, onde as três tabelas para indexação de sujeitos, objetos e predicados foram criadas corretamente, e o segundo comando onde a linha identificada pelo sujeito *http://swe.unisinos.br/ont/agents#Paulo* resultou em todos os predicados e objetos ligados a este indivíduo, sendo o identificador da coluna composto pela família de colunas *triples:*, seguido pelo predicado (por exemplo, *http://swe.unisinos.br/ont/agents#facebookID*), pelo separador *~~*, e por fim pela *hash* do objeto. O conteúdo da combinação linha e coluna é o valor completo do objeto da tripla RDF, conforme o esperado.

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.4, r1428173, Thu Jan 3 06:29:56 UTC 2013

hbase(main):001:0> list
TABLE
2013-02-23 16:20:58.887 java[3921:1703] Unable to load realm info from SCDynamicStore
agentjs-tbl-objects
agentjs-tbl-predicates
agentjs-tbl-subjects
3 row(s) in 0.6620 seconds

hbase(main):002:0> get 'agentjs-tbl-subjects', 'http://swe.unisinos.br/ont/agents#Paulo'
COLUMN                                CELL
triples:http://swe.unisinos.br/ont     timestamp=1361644143701, value="100000579436195"^^http://www.w3.org/2001/XMLSchema#long
/agents#facebookID~~a1288cbb4b6220
fad1f0a12c89cb6451
triples:http://swe.unisinos.br/ont     timestamp=1361644143913, value="paulocesar.butenbender"^^http://www.w3.org/2001/XMLSchema#string
/agents#facebookName~~c21062978400
ef270ef8f5d982c563f8
triples:http://swe.unisinos.br/ont     timestamp=1361644143706, value=http://swe.unisinos.br/ont/agents#Paulo
/agents#hasProfile~~e0c4f05b091e50
0f445977c3c096114f
triples:http://www.w3.org/1999/02/     timestamp=1361644143642, value=http://swe.unisinos.br/ont/agents#FacebookProfile
22-rdf-syntax-ns#type~~6a2245c72c1
f248e461e3e45f9c2cb62
triples:http://www.w3.org/1999/02/     timestamp=1361644143636, value=http://www.w3.org/2002/07/owl#NamedIndividual
22-rdf-syntax-ns#type~~bb70ba4af84
eba77e76cf90374b5e869
triples:http://www.w3.org/1999/02/     timestamp=1361644143647, value=http://swe.unisinos.br/ont/agents#Person
22-rdf-syntax-ns#type~~cc0c3870e72
9aebac81088b44692725
6 row(s) in 0.1440 seconds
```

Figura 34 - Resultado das operações de busca no banco de dados distribuído

O segundo passo da validação consiste em verificar se a interface de busca de dados RDF do Jena é capaz de ler e retornar os dados conforme o que foi persistido no banco de dados. Para isso é utilizado a extensão do navegador *Google Chrome* chamada *Postman* e serviço de consulta SPARQL descrito no capítulo 4.3.6. A requisição solicita a busca de um sujeito que possua a classe *FacebookProfile*, e cujo atributo *facebookName* seja equivalente a “paulocesar.butenbender”, e retorne o identificador do sujeito e a sua propriedade *facebookID*. Essa consulta foi realizada com sucesso conforme ilustrado pela Figura 35.

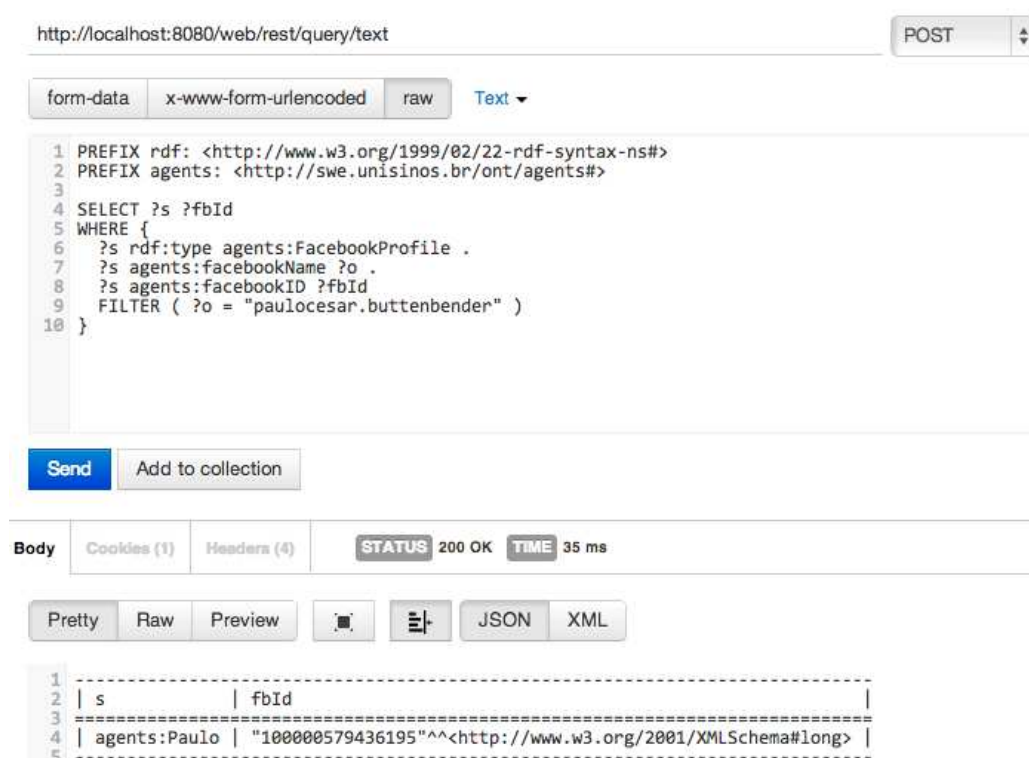


Figura 35 - Validação do serviço de consulta SPARQL

O terceiro passo da validação consiste em verificar se os serviços de criação e leitura de agentes estão funcionando conforme o especificado a partir do uso das telas de criação e listagem de agentes. A Figura 36 ilustra a tela de modificação de agentes, onde um agente existente de nome “Sms Localizador” é apresentado para o usuário com o identificador “paulocesar.butenbender”. Logo abaixo da listagem é possível verificar os campos para alteração de nome e código do agente. Uma vez que todas as funcionalidades responderam conforme o esperado, é possível determinar que os três serviços relacionado com as três funcionalidades (listagem de agentes, criação de agente, alteração de agente) estão respondendo e se comportando conforme o esperado.

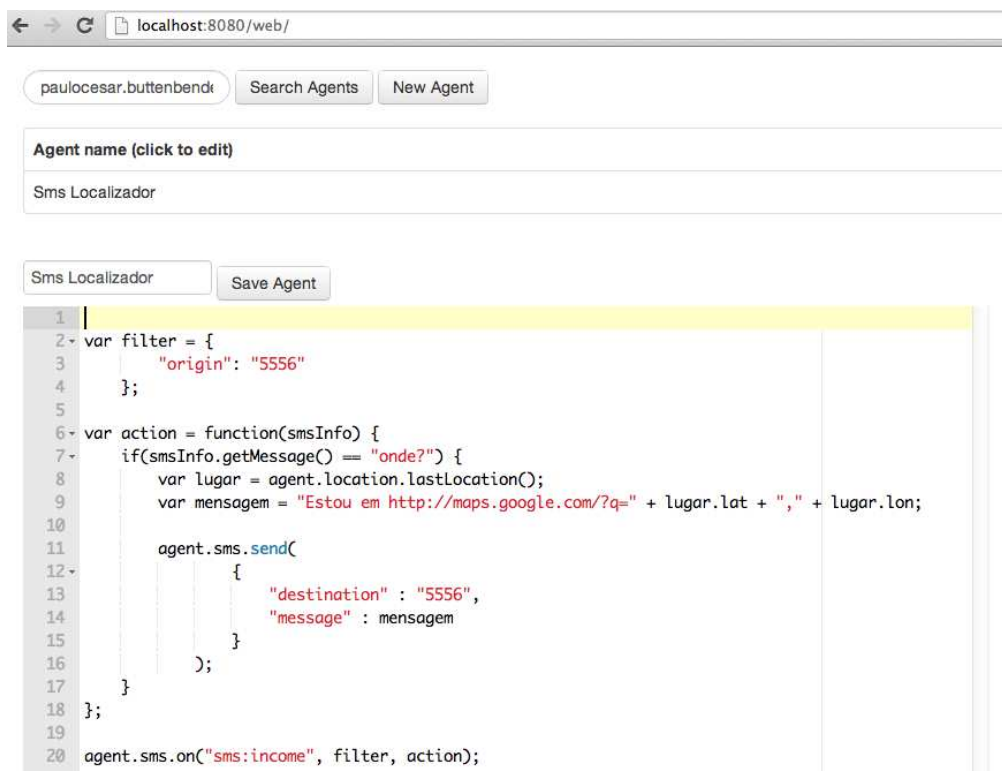


Figura 36 - Tela para manutenção de agentes

O quarto passo da validação consiste em verificar se a atualização periódica de contexto está sendo transmitida do dispositivo móvel para o histórico de contexto, e se o histórico de contexto está persistindo os dados corretamente.

Para esta validação foi necessário instalar e iniciar o motor de execução no emulador de dispositivo móvel, configurar o endereço do histórico de contextos para o servidor de aplicação local (conforme ilustrado na primeira imagem da Figura 37) e verificar nos logs da aplicação se o contexto está sendo emitido conforme o esperado (ilustrado na Figura 38).

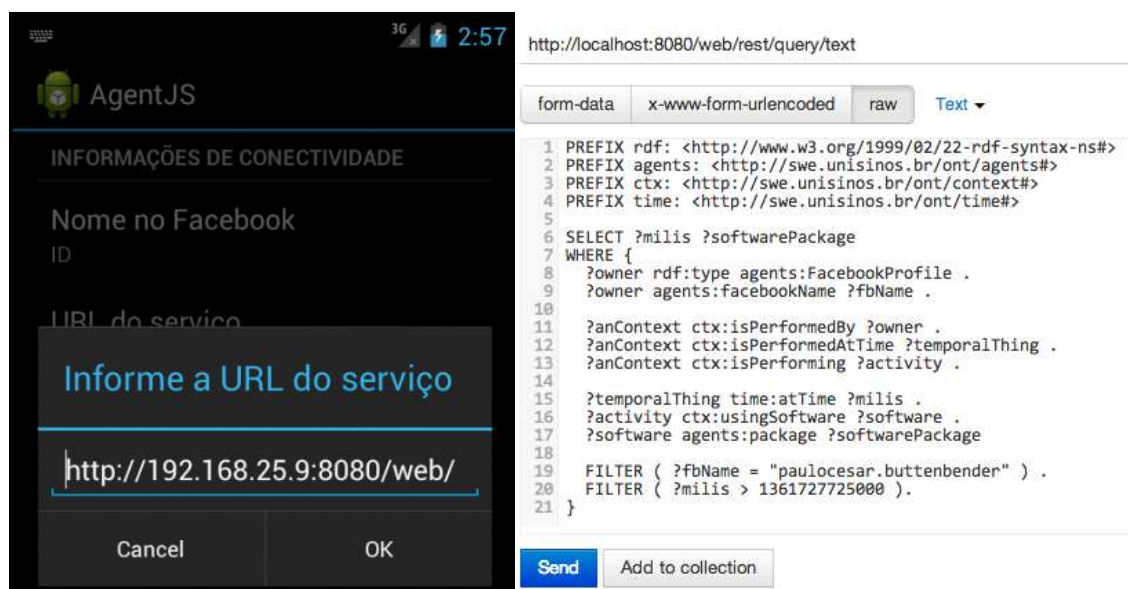


Figura 37 - (a) Preferência do histórico de contextos, (b) Busca SPARQL para listar aplicativos executados após um intervalo pré-definido

Após confirmar que o dispositivo está emitindo informações de contexto para o endereço correto, é necessário validar se o contexto está sendo persistido corretamente. Para isso foram elaboradas instruções de busca SPARQL (a qual uma está ilustrada no segundo item da Figura 37) com objetivo de verificar se o contexto foi corretamente persistido, no caso da ilustração a consulta visa listar os aplicativos em execução em um intervalo pré-definido de tempo.

Neste intervalo de tempo foram realizadas as seguintes atividades: (i) executado o aplicativo do motor de execução, (ii) retornado para a tela inicial do *Android* e aguardado 12 minutos, (iii) retornado para o aplicativo de motor de execução.

Esta atividade busca validar dois aspectos da aplicação: (i) a periodicidade do envio de informações de contexto para o histórico de contexto na nuvem, e (ii) a capacidade do motor de execução de continuar em funcionamento durante a execução de outras atividades no dispositivo móvel. A Figura 39 ilustra o resultado a busca SPARQL, que apresenta o funcionamento da atualização de contexto conforme o esperado.

	PID	Application	Tag	Text
05.392	2256	br.unisinos.swe.agentjs	AgentJS	Uploading context to cloud: http://192.168.25.9:8080/web/rest/context

Figura 38 - Informação no emulador do contexto sendo emitido para o servidor de aplicação local

Body		Cookies (1)	Headers (4)	STATUS 200 OK	TIME 77 ms
Pretty Raw Preview   JSON XML					
1					softwarePackage
2	millis				
3	-----				
4	"1361727785000"^^<http://www.w3.org/2001/XMLSchema#long>				"br.unisinos.swe.agentjs"^^<http://www.w3.org/2001/XMLSchema#string>
5	"1361727845000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
6	"1361727905000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
7	"1361727965000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
8	"1361728025000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
9	"1361728085000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
10	"1361728145000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
11	"1361728205000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
12	"1361728265000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
13	"1361728325000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
14	"1361728385000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
15	"1361728445000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
16	"1361728505000"^^<http://www.w3.org/2001/XMLSchema#long>				"com.android.launcher"^^<http://www.w3.org/2001/XMLSchema#string>
17	"1361728565000"^^<http://www.w3.org/2001/XMLSchema#long>				"br.unisinos.swe.agentjs"^^<http://www.w3.org/2001/XMLSchema#string>
18	"1361728625000"^^<http://www.w3.org/2001/XMLSchema#long>				"br.unisinos.swe.agentjs"^^<http://www.w3.org/2001/XMLSchema#string>
19	"1361728685000"^^<http://www.w3.org/2001/XMLSchema#long>				"br.unisinos.swe.agentjs"^^<http://www.w3.org/2001/XMLSchema#string>
20	-----				

Figura 39 - Resultado da seleção de aplicativos em execução em um determinado intervalo

O quinto passo da validação consiste em verificar se o dispositivo móvel está encontrando e interpretando agentes criados no histórico de contexto. Para este teste foi criado um agente chamado de “Notificador” cujo código emite uma notificação no dispositivo para simplesmente validar os dois aspectos: (i) o dispositivo móvel é capaz de fazer o download dos agentes criados no servidor de aplicação, (ii) o dispositivo móvel é capaz de interpretar o código dos agentes criados no servidor de aplicação.

Na Figura 41 estão ilustrados os agentes existentes no servidor de aplicação e o código do agente “Notificador”.

```

1 var notif = {
2   "title" : "notificador!",
3   "content" : "Conteudo"
4 };
5 agent.notification.send(notif);
  
```

Figura 40 - Agentes existentes no servidor de aplicação e instrução do agente “Notificador”

A Figura 41 ilustra tanto a listagem de agentes locais (composta por agentes cujo download é realizado do servidor da aplicação e agentes que existem no cartão SD do dispositivo) e o resultado da interpretação do agente “Notificador”.

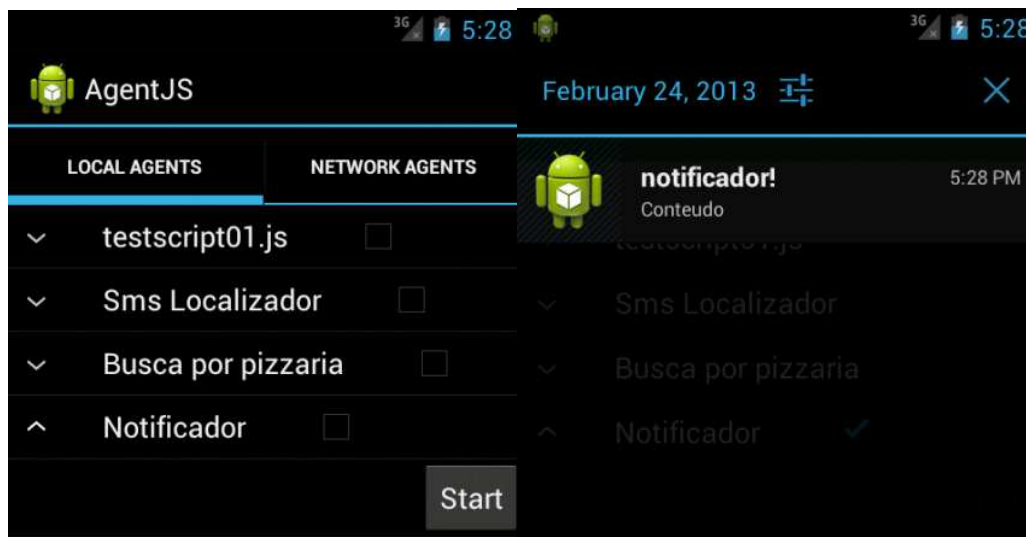


Figura 41 - (a) Lista de agentes no cartão SD e no servidor de aplicação, (b) Resultado da interpretação do agente Notificador

O sexto passo da validação consiste em verificar se o dispositivo móvel está encontrando agentes em uma rede local de forma automática conforme a especificação de serviço UPnP para descoberta de agentes descrita na seção 4.4.6. Para este passo de validação foi construído um simulador de dispositivo UPnP com o serviço de disponibilização de agentes utilizando a biblioteca *Cling* para enviar ao dispositivo móvel um agente de nome “Notificador UPnP” com objetivo de emitir um alerta de sucesso assim que executado. Para validação foi inicializado o simulador de dispositivo e em seguida inicializado o motor de execução no dispositivo móvel, foi verificado na aba de agentes na rede se o agente “Notificador UPnP” estava presente conforme esperado, e executado o notificador para verificar se o código do mesmo seria executado conforme o esperado. Todas as funcionalidades para busca de agentes, código e execução do agente em rede se comportaram conforme o esperado e a listagem e resultado estão ilustrados na Figura 42.



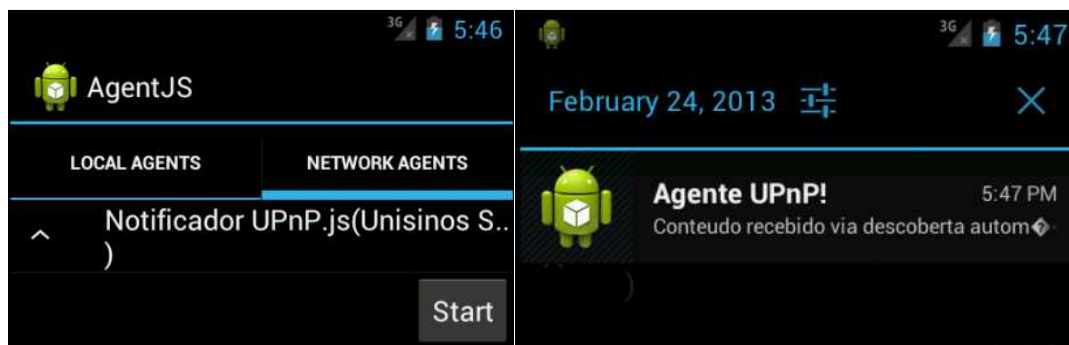


Figura 42 - (a) Listagem de agentes descobertos via UPnP, (b) Resultado da interpretação do agente descoberto via UPnP

O sétimo passo da validação consiste em verificar se os componentes de API estão se comportando conforme especificado na seção 4.4.3, e por consequência validar se os sinais descritos na seção 4.4.4 estão funcionando de forma adequada. Para esta validação foram construídos quatro agentes para testar os componentes de API: (i) agente para teste do componente de API responsável pelos sinais de localização, utiliza a rota gerada pelo Google Earth ilustrada na Figura 43 para simular o movimento do dispositivo móvel pela cidade de São Leopoldo.



Figura 43 - Rota simulada pelo emulador em branco, marcadores com todas as pizzarias próximas a rota

O agente será construído para emitir uma notificação toda vez que o dispositivo móvel ficar a menos de 150 metros de uma pizzaria, e, portanto é possível determinar que apenas alertas para as pizzarias 1 a 4 devem ser emitidos durante o teste. O resultado deste teste foi

conforme o esperado, confirmando o funcionamento dos sensores de proximidade e da busca de locais baseado em palavras chave.



Figura 44 - (a) Código do agente de Busca por pizzaria, (b) Última notificação emitida, correspondente a Pizzaria 4

(ii) agente para teste do componente de API de SMS e música, criado com objetivo de testar o sinal de SMS e a capacidade de iniciar o aplicativo de música para um endereço relativo no dispositivo móvel (ilustrado pela Figura 45).

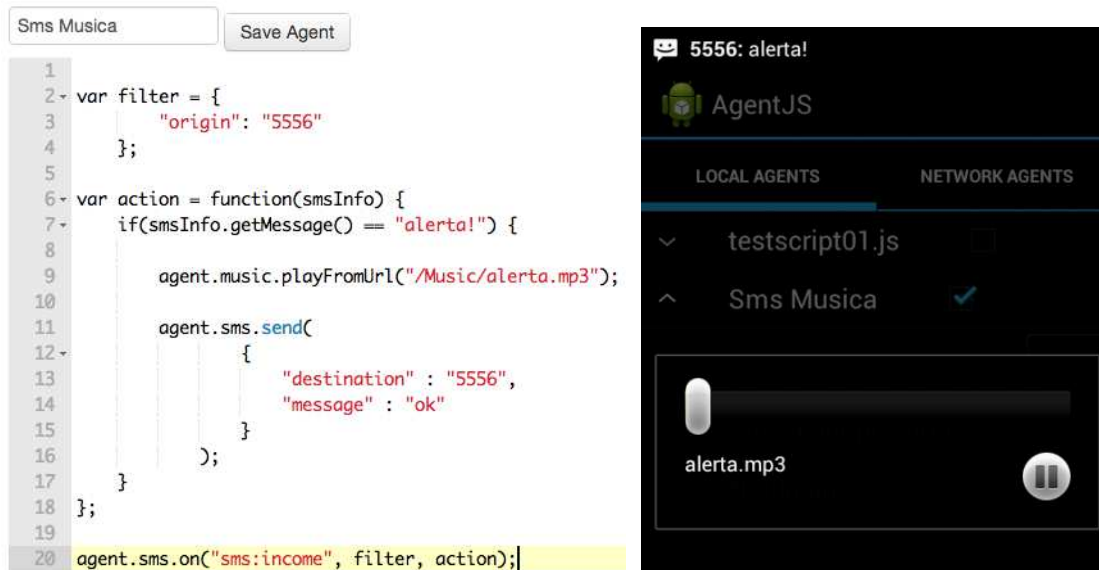


Figura 45 - (a) Código do agente de Sms Música, (b) Música iniciada logo após o recebimento do SMS de alerta

(iii) agente para testes do componente HTTP, cujo objetivo é realizar uma requisição HTTP GET, realizar a conversão da resposta de um objeto JSON para um objeto Javascript e colocar no log do dispositivo os dados retornados pelo Webservice. O código fonte do agente está ilustrado na Figura 46 e a resposta no log está ilustrada na Figura 47.

Teste Http      Save Agent

```

1 var url = "http://www.isitbirthday.com/paulo.json";
2
3 var httpRequest = {
4   "url" : url
5 };
6
7 var onSuccess = function(sResponse) {
8   var bDay = JSON.parse(sResponse);
9   // resposta esperada:
10  // {"birthday": {"day": 23, "month": 9}, "name": "paulo"}
11  agent.log( bDay.name +
12            " aniversario em " +
13            bDay.birthday.day +
14            "/" +
15            bDay.birthday.month );
16 };
17
18 var onError = function() {
19   agent.log("offline");
20 };
21
22 agent.http.get(httpRequest, onSuccess, onError);

```

Figura 46 - Código do agente de validação do componente de HTTP

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit

PID	Application	Tag	Text
16121	br.unisinos.swe.agentjs	AgentJS	trying to access component:
16121	br.unisinos.swe.agentjs	AgentJS	trying to access method: cl
16121	br.unisinos.swe.agentjs	AgentJS	trying to access component:
16121	br.unisinos.swe.agentjs	AgentJS	trying to access method: cl
16121	br.unisinos.swe.agentjs	AgentJS	trying to access method: cl
16121	br.unisinos.swe.agentjs	AgentJS	paulo aniversario em 23/9

Figura 47 - Resultado da requisição no log do dispositivo móvel

(iv) agente para teste do componente de manipulação de aplicativos, construído para testar a capacidade da API de inicializar um aplicativo, no exemplo foi inicializado o aplicativo de calendário conforme ilustrado na Figura 48.

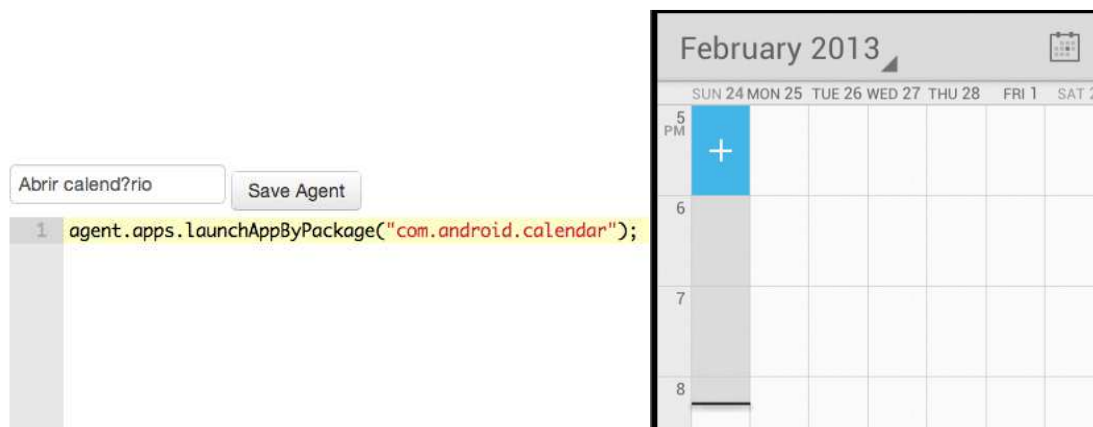


Figura 48 – (a) Código do teste de inicialização de aplicativos por agentes, (b) aplicativo aberto após a interpretação do agente

Nesta seção foram realizadas validações em um ambiente virtual controlado com objetivo de apresentar as telas, funcionalidades e garantir a conformidade entre as funcionalidades especificadas e as funcionalidades desenvolvidas. A próxima seção apresenta os dispositivos utilizados para a validação em um ambiente real e as considerações sobre os resultados obtidos.

### 5.3 TESTE DE FUNCIONALIDADE EM AMBIENTE REAL

Para o ambiente real utilizado nos testes de funcionalidade foram utilizados os seguintes dispositivos:

- Um Motorola Dext, processador ARM de 528MHz e 256MiB de RAM, atualizado para a versão 2.3.4 da plataforma Android, utilizado como dispositivo móvel e com o motor de execução instalado;
- Um *Raspberry PI* modelo B, processador ARM de 700MHz e 512MiB de RAM, sistema operacional *Debian Wheezy* e máquina virtual *Java SE Embedded 7* da Oracle. Foi utilizado como dispositivo inteligente para disponibilizar agentes para o serviço UPnP de descoberta automática de agentes;
- Um *cluster* formado por três servidores *m1.large* na infraestrutura da *Amazon Web Services*, sendo um dos servidores utilizado tanto como servidor de aplicação como controlador do banco de dados distribuídos, e dois servidores como armazenamento dos dados distribuídos; Uma instância *m1.large* corresponde a um servidor com 7.5MiB de RAM e 4 *EC2 Compute Unit* (uma

EC2 Compute Unit tem uma capacidade de processamento similar a um processador Intel Xeon 2007, de 1 GHz);

- Um roteador Asus RT-N10+ como infraestrutura de rede e para testes dos sinais de *wifi*;

O dispositivo móvel foi formatado e nele foi instalado o motor de execução. Em seguida foi realizado com sucesso o seguinte cenário de teste:

1. Criar um agente para iniciar o aplicativo de motor de execução toda vez que o dispositivo móvel conectar em uma rede *wifi*;
2. Entrar no aplicativo motor de execução, configurar o servidor para apontar para a nuvem;
3. Atualizar a lista de agentes e inicializar o agente criado no primeiro passo;
4. Sair do aplicativo do motor de execução;
5. Criar um agente no dispositivo inteligente para emitir uma notificação de sucesso quando executado;
6. Conectar com o dispositivo móvel em uma rede *wifi*;
7. Após a abertura automática do motor de execução, acessar a aba de agentes na rede (ilustrado na Figura 49);
8. Inicializar o agente descoberto via UPnP.

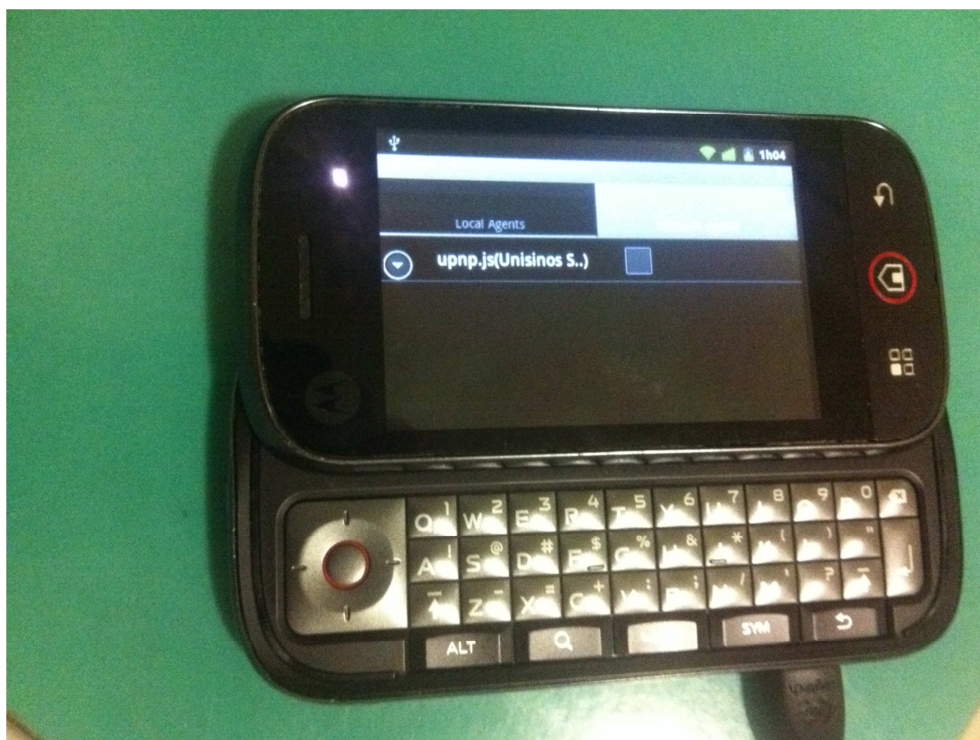


Figura 49 - Motorola Dext listando os agentes encontrados na rede

Foi utilizado o *Raspberry Pi* como dispositivo inteligente no cenário de teste desta seção. Como principal motivo é possível destacar seu baixo custo (35 dólares) e a possibilidade da instalação da máquina virtual Java, o que reduziu o esforço para a implementação dos serviços UPnP fazendo uso da biblioteca *Cling*. Neste dispositivo foi construída uma aplicação Java para ler uma pasta local a cada 60 segundos, e expor cada arquivo com extensão *.js* como um agente na rede local. A Figura 50 ilustra o dispositivo utilizado.



Figura 50 - Raspberry Pi

Neste mesmo cenário de teste, todas as instâncias dos servidores *m1.large* da *Amazon EC2* foram alocadas em São Paulo para reduzir o efeito da latência da rede durante os testes. A Figura 51 apresenta a página de monitoramento das instâncias, com os três servidores online.

The screenshot shows the AWS Management Console interface for EC2 instances in the São Paulo region. The left sidebar contains navigation options like 'EC2 Dashboard', 'Events', 'INSTANCES', 'Instances', 'Spot Requests', 'Reserved Instances', 'IMAGES', and 'AMIs'. The main area displays a table of running instances:

Name	Instance	AMI ID	Root Device	Type	State
HBase_Slave2	i-e84716f4	ami-89805b94	instance store	m1.large	running
HBase_Slave1	i-eb4716f7	ami-89805b94	instance store	m1.large	running
HBase_Master	i-ef4716f3	ami-89805b94	instance store	m1.large	running

Figura 51 - Instâncias Amazon EC2 alocadas em São Paulo

O alto poder de processamento das instâncias e a curta distância geográfica afetam diretamente o tempo de resposta de uma consulta SPARQL, que conforme ilustrado na Figura 52 buscou todos os agentes do usuário “paulocesar.butenbender” em 60 *milissegundos*.

The screenshot shows a SPARQL query execution interface. The URL is `http://ec2-54-232-38-16.sa-east-1.compute.amazonaws.com:8080/web/rest/query/text`. The query is displayed in a text area, and the results are shown in a table format.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX agents: <http://swe.unisinos.br/ont/agents#>
3
4 SELECT ?uuid ?name ?code
5 WHERE {
6   ?owner rdf:type agents:FacebookProfile .
7   ?owner agents:facebookName ?fbName .
8   ?owner agents:ownAgent ?anAgent .
9
10  ?anAgent rdf:type agents:AgentJS .
11  ?anAgent agents:uuid ?uuid .
12  ?anAgent agents:name ?name .
13  ?anAgent agents:sourceCode ?code
14  FILTER ( ?fbName = "paulocesar.butenbender" )
15 }

```

The interface shows a status of 200 OK and a response time of 60 ms. The results are displayed in a table with columns for uuid, name, and code.

uuid	name	code
"b07de00e-5382-4ad7-a46c-2852574adf64"	<http://www.w3.org/2001/XMLSchema#string>	"Notificador"

Figura 52 - Consulta SPARQL no servidor de aplicação

É importante destacar que o desempenho dos servidores e dos dispositivos no ambiente real foi muito superior quando comparado ao ambiente virtual, mesmo considerando a latência e os tempos de conexão via da rede, e que todas as funcionalidades testadas no ambiente virtual também foram testadas com sucesso no ambiente real.

#### 5.4 TESTE DE DESEMPENHO

O cluster de servidores criados para o teste em ambiente real foi colocado sob stress de carga para medir o desempenho com o crescimento do volume de informações.

Para este teste de carga foram criados contextos de forma progressiva (iniciando em 10.000 contextos e progredindo até 35.000 contextos), e para cada passo da progressão foi realizada a média do tempo de resposta para quatro consultas SPARQL no histórico de contextos armazenado.

A primeira consulta analisada (apresentado na Tabela 14) é uma simples contagem de registros de tempo (que por consequência resultam na contagem de registros de histórico de contexto), que mostra a progressão do tempo de resposta com o aumento da quantidade de registros no banco de dados distribuído.

Tabela 14 - Consulta C1

(C1) Consulta SPARQL de contagem de histórico de contexto
<pre>PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX agents: &lt;http://swe.unisinos.br/ont/agents#&gt; PREFIX time: &lt;http://swe.unisinos.br/ont/time#&gt;  SELECT (COUNT(?milis) as ?cnt) WHERE {   ?temporalThing time:atTime ?milis . }</pre>

A segunda consulta analisada (apresentada na Tabela 15) é uma seleção do contexto de um usuário específico para um período de tempo pré-determinado, que mostra a variação do tempo de resposta com o aumento da quantidade de registros de histórico de contexto.

Tabela 15 - Consulta C2

(C2) Consulta SPARQL otimizada para seleção de histórico de contexto em um período pré-determinado
--



```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agents: <http://swe.unisinos.br/ont/agents#>
PREFIX ctx: <http://swe.unisinos.br/ont/context#>
PREFIX time: <http://swe.unisinos.br/ont/time#>

SELECT ?milis ?softwarePackage
WHERE {
  ?temporalThing time:atTime ?milis .
  FILTER ( ?milis > 1361919814174 ) .
  FILTER ( ?milis < 1361919859174 ) .

  ?anContext ctx:isPerformedAtTime ?temporalThing .
  ?anContext ctx:isPerformedBy ?owner .
  ?anContext ctx:isPerforming ?activity .

  ?owner rdf:type agents:FacebookProfile .
  ?owner agents:facebookName ?fbName .

  ?activity ctx:usingSoftware ?software .
  ?software agents:package ?softwarePackage
  FILTER ( ?fbName = "paulocesar.buttenbender" ) .
}

```

A terceira consulta analisada (apresentada na Tabela 16) é uma seleção do contexto idêntica à segunda consulta, mas com mudança nos critérios de filtro, que ilustra problemas no desempenho apenas com a ordenação errônea das cláusulas de filtro SPARQL.

Tabela 16 - Consulta C3

(C3) Consulta SPARQL sem otimização para seleção de histórico de contexto, idêntica a consulta C2.

```

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX agents: <http://swe.unisinos.br/ont/agents#>
PREFIX ctx: <http://swe.unisinos.br/ont/context#>
PREFIX time: <http://swe.unisinos.br/ont/time#>

SELECT ?milis ?softwarePackage
WHERE {
  ?owner rdf:type agents:FacebookProfile .
  ?owner agents:facebookName ?fbName .
}

```

```

?anContext ctx:isPerformedBy ?owner .
?anContext ctx:isPerformedAtTime ?temporalThing .
?anContext ctx:isPerforming ?activity .

?temporalThing time:atTime ?milis .
?activity ctx:usingSoftware ?software .
?software agents:package ?softwarePackage

FILTER ( ?fbName = "paulocesar.buttendbender" ) .
FILTER ( ?milis > 1361919814174 ) .
FILTER ( ?milis < 1361919859174 ) .
}

```

A quarta e última consulta analisada (apresentada na Tabela 17) é uma seleção dos agentes de um usuário da plataforma, que ilustra o comportamento de uma consulta SPARQL não relacionada ao histórico de contexto com o aumento no volume de registro de histórico de contexto.

Tabela 17 - Consulta C4

(C4) Consulta SPARQL de agentes.
<pre> PREFIX rdf: &lt;http://www.w3.org/1999/02/22-rdf-syntax-ns#&gt; PREFIX agents: &lt;http://swe.unisinos.br/ont/agents#&gt; PREFIX ctx: &lt;http://swe.unisinos.br/ont/context#&gt;  SELECT ?uuid ?name ?code WHERE {   ?owner rdf:type agents:FacebookProfile .   ?owner agents:facebookName ?fbName .   ?owner agents:ownAgent ?anAgent .    ?anAgent rdf:type agents:AgentJS .   ?anAgent agents:uuid ?uuid .   ?anAgent agents:name ?name .   ?anAgent agents:sourceCode ?code   FILTER ( ?fbName = "paulocesar.buttendbender" ) } </pre>

A Figura 53 apresenta os resultados obtidos pelo teste de carga nas consultas C1, C2 e C4, onde é possível verificar que a seleção C4 não é afetada pelo aumento de registros de contexto.

Já as consultas C1 e C2 apresentam o crescimento linear no tempo de resposta em relação à quantidade de contextos armazenados no banco de dados distribuído. Considerando uma granularidade de um contexto a cada 15 minutos, o tempo de resposta para uma busca no histórico de contexto acumulado durante um ano ficaria um pouco abaixo de 1 segundo. Como forma de aperfeiçoar as consultas existe a possibilidade de utilizar *MapReduce* como motor de inferência e buscas, aproveitando a distribuição dos dados para também distribuir o processamento das consultas. Este uso do *MapReduce* aplicado ao processamento de dados RDF distribuídos está listado como um dos possíveis trabalhos futuros consequentes desta proposta de plataforma.

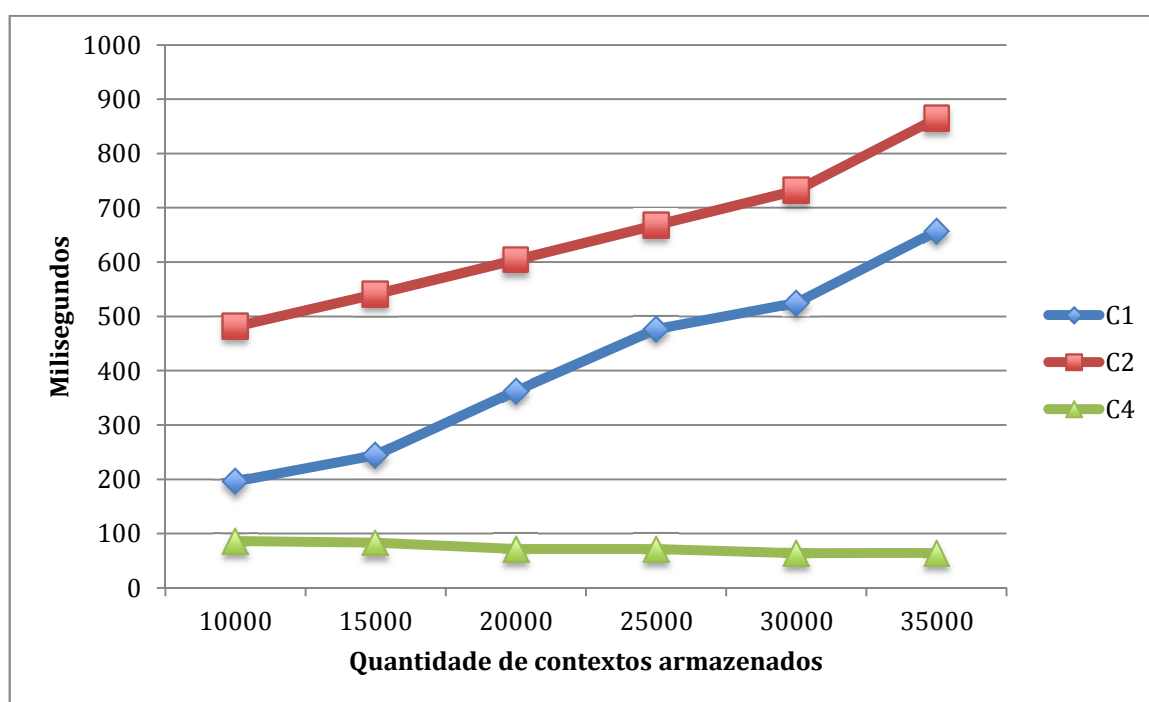


Figura 53 - Tempo de resposta por Quantidade de contextos

A Figura 54 apresenta o comparativo entre uma consulta de histórico de contextos em SPARQL otimizada (C2) e uma consulta onde os filtros estão no final da instrução. A diferença no tempo de resposta é 100 vezes superior na consulta sem otimização.

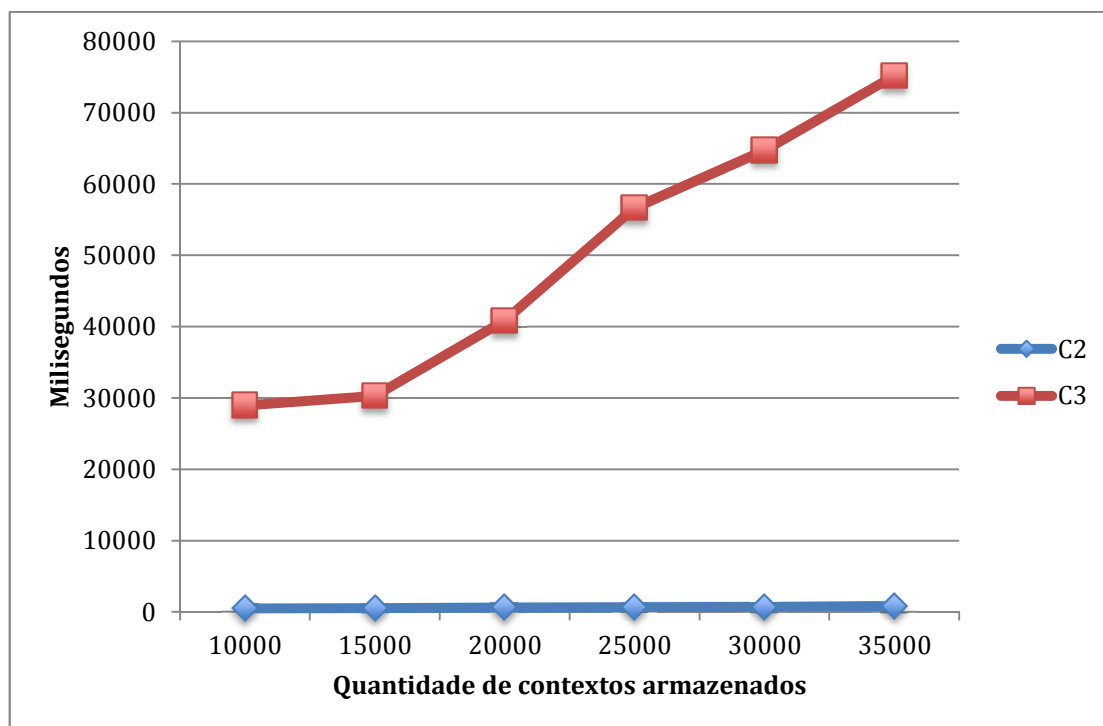


Figura 54 - Comparativo entre consulta SPARQL otimizada e consulta comum

## 6 CONCLUSÃO

O estado atual dos padrões e tecnologias web permitem a construção de uma plataforma de computação pervasiva em dispositivos móveis com suporte a histórico de contextos com capacidade de suportar o volume necessário para aplicativos com características relacionadas a computação pervasiva.

A maior contribuição deste trabalho está em desenvolver uma plataforma de computação pervasiva que supre um conjunto de requisitos não encontrados em sua totalidade em nenhum dos trabalhos semelhantes avaliados.

Nos requisitos que tratam de escalabilidade e utilização de tecnologias de computação na nuvem para um custo reduzido e um melhor aproveitamento de recursos computacionais, este trabalho consegue utilizar trabalhos anteriores e propor um novo modelo de armazenagem de triplas RDF em um banco de dados distribuído e reconhecido por sua estabilidade e escalabilidade.

Na modelagem de ontologias para computação pervasiva, este trabalho estende e combina com sucesso ontologias existentes pertinentes para o cenário proposto. Esta modelagem atende os requisitos de histórico de contexto e herda a possibilidade de reuso inerente a ontologias OWL.

Na construção da plataforma para computação pervasiva baseada em dispositivos móveis, este trabalho consegue utilizar com sucesso uma linguagem de programação suportada pelos principais navegadores para permitir a construção de agentes. Estes agentes podem ser construídos pelos usuários da plataforma ou disponibilizados em uma rede local e descobertos automaticamente pela plataforma. A utilização de uma linguagem de script interpretada permite a cópia de seu código fonte e execução imediata, o que permite a comunicação entre ambientes inteligentes e agentes no dispositivo móvel.

Devido à característica de a plataforma permitir a construção de outros sistemas, trabalhos futuros podem explorar tanto a expansão da plataforma como a utilização da mesma nos mais diversos cenários de computação pervasiva.

### 6.1 PRINCIPAIS CONTRIBUIÇÕES

A partir do estudo de trabalhos semelhantes e relacionados (detalhados no capítulo 3), bem como o detalhamento da plataforma proposta (apresentado em detalhes nas seções 4.3 e

4.4) e os resultados obtidos (conforme apresentado no capítulo 5), é possível destacar as principais contribuições deste trabalho como:

- Modelagem e implementação de um motor de execução de agentes/scripts Javascript em dispositivos móveis de código fonte aberto, conforme detalhado na seção 4.4;
- Especificação e implementação de armazenamento de histórico de contextos em ontologias utilizando banco de dados distribuídos, conforme detalhado na seção 4.3;
- Novo modelo físico para armazenamento de triplas RDF em banco de dados distribuídos, conforme detalhado na seção 4.3.5;
- Modelagem de contextos a partir da extensão de ontologias existentes, conforme detalhado na seção 4.3.3.

## 6.2 TRABALHOS FUTUROS

Este trabalho atingiu as metas definidas durante o planejamento. No entanto durante sua construção novos recursos foram percebidos como interessantes e passíveis de implementação, e dentre os possíveis trabalhos futuros é possível destacar:

- *Validar a utilidade percebida.* Utilizar o modelo de aceitação de tecnologia (*Technology Acceptance Model*) para validar a utilidade percebida da plataforma com desenvolvedores envolvidos com desenvolvimento de *software* para computação pervasiva.
- *Interface com usuário.* Adicionar capacidade de construção de interfaces padrões para que agentes JS possam interagir diretamente com o usuário da plataforma.
- *Interface com sensores externos.* Adicionar a capacidade de o agente JS realizar requisições utilizando o protocolo CoAP (*Constrained Application Protocol*), projetado especificamente para sensores e dispositivos restritos.
- *Agendamento de tarefas.* Adicionar a capacidade de agendar tarefas e restrição de uso de sinais baseado em intervalos de tempo.
- *Comunicação entre agentes.* Atualmente não existe uma forma facilitada para comunicação entre agentes na plataforma, e um formato para troca de

mensagens entre agentes baseado em padrões (como por exemplo, o FIPA) abre mais opções de integração para a plataforma.

- *Integrar com ambientes inteligentes.* Utilizar a plataforma em um ambiente inteligente a partir do desenvolvimento e disponibilização de agentes.
- *Identificar padrões no histórico de contexto.* Utilizar técnicas de identificação de padrões para identificar perfis a partir das ontologias armazenadas de forma distribuída.
- *Integrar com máquina de inferência MapReduce.* A plataforma utiliza um banco de dados distribuído para armazenar o histórico de contexto, e isso permite a construção de uma máquina de inferência baseada em *MapReduce* para inferência sobre os dados armazenados.
- *Especializar a ontologia em cenários.* A ontologia definida por este trabalho é uma forma básica para representação de contexto, e pode ser especializada para diferentes cenários de computação pervasiva.

## REFERÊNCIAS

- ABADI, D. J. *et al.* Scalable Semantic Web Data Management Using Vertical Partitioning. **Proceedings of the 33rd international conference on Very large data bases**, v. VLDB '07, p. 411-422, 2007.
- ALTINI, M. *et al.* Bluetooth indoor localization with multiple neural networks. **IEEE 5th International Symposium on Wireless Pervasive Computing 2010**, p. 295-300, 2010.
- ANTONOPOULOS, N.; GILLAM, L. **Cloud Computing: Principles, Systems and Applications**. [s.l.] Springer, 2010. v. 54p. 379
- APACHE SOFTWARE FOUNDATION. **What is Jena?** Disponível em: <[http://jena.apache.org/about\\_jena/about.html](http://jena.apache.org/about_jena/about.html)>. Acesso em: 30 out. 2012.
- APACHE SOFTWARE FOUNDATION. **ARQ - A SPARQL Processor for Jena**. Disponível em: <<http://jena.apache.org/documentation/query/>>. Acesso em: 30 out. 2012.
- APACHE SOFTWARE FOUNDATION. **HDFS Architecture Guide**. Disponível em: <[http://hadoop.apache.org/docs/r0.20.2/hdfs\\_design.html](http://hadoop.apache.org/docs/r0.20.2/hdfs_design.html)>. Acesso em: 20 jan. 2013.
- BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. JADE—A FIPA-compliant agent framework. **Proceedings of PAAM**, 1999.
- BROOKS, K. The context quintet: narrative elements applied to context awareness. **Human Computer Interaction International ...**, 2003.
- CARROLL, J.; DICKINSON, I.; DOLLIN, C. Jena: implementing the semantic web recommendations. **WWW Alt. '04**, p. 74-83, 2004.
- CHANG, F.; DEAN, J.; GHEMAWAT, S. Bigtable: A distributed storage system for structured data. **ACM Transactions on ...**, 2008.
- CHEN, H. **An Intelligent Broker Architecture for Pervasive Context-Aware Systems**. [s.l.] University of Maryland, 2004.
- CHEN, H.; FININ, T.; JOSHI, A. Intelligent agents meet the semantic web in smart spaces. **Internet Computing ...**, n. October, p. 2-12, 2004.
- CHEN, H.; FININ, T.; JOSHI, A. The SOUPA ontology for pervasive computing. **Ontologies for agents: Theory and experiences**, 2005.
- DOROKHOVA, R.; AMELICHEV, N.; KRINKIN, K. **Evaluation of Modern Mobile Platforms from the Developer Standpoint** 8th Conference of Finnish-Russian University Cooperation in Telecommunications FRUCT. **Anais...2010** Disponível em: <[http://osll.spb.ru/attachments/download/429/evaluation\\_2.doc](http://osll.spb.ru/attachments/download/429/evaluation_2.doc)>



EDWARDS, W. K. Discovery systems in ubiquitous computing. **Pervasive Computing, IEEE**, v. 5, n. 2, p. 70–77, 2006.

FISCHER, G.; DIETRICH, B.; WINKLER, F. Bluetooth indoor localization system. **Hannoversche Beiträge zur**, p. 147-156, 2004.

FRIDAY, A. *et al.* Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. **Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access MobiDe 01**, v. 10, n. 6, p. 7-13, 2004.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The Google file system. **ACM SIGOPS Operating Systems Review**, v. 37, n. 5, p. 29, 2003.

GOASDUFF, L.; PETTEY, C. **Gartner Says Worldwide Sales of Mobile Phones Declined 2 Percent in First Quarter of 2012; Previous Year-over-Year Decline Occurred in Second Quarter of 2009**. Disponível em: <<http://www.gartner.com/it/page.jsp?id=2017015>>. Acesso em: 30 out. 2012.

GRUBER, T. Ontology. *In*: LIU, L.; ÖZSU, M. T. (Eds.). **Encyclopedia of Database Systems**. [s.l.] Springer-Verlag, 2009. .

GU, Y.; LO, A.; NIEMEGEREERS, I. A survey of indoor positioning systems for wireless personal networks. **Communications**, v. 11, n. 1, p. 13-32, 2009.

HALLBERG, J.; NILSSON, M.; SYNNESE, K. Positioning with Bluetooth. **10th International Conference on Telecommunications 2003 ICT 2003**, v. 2, p. 954-958, 2003.

HALLER, S. The Things in the Internet of Things. 2010.

HAY, S.; HARLE, R. Bluetooth Tracking without Discoverability. **Lecture Notes in Computer Science**, p. 120-137, 2009.

HU, C.-L.; HUANG, Y.-J.; LIAO, W.-S. Multicast Complement for Efficient UPnP Eventing in Home Computing Network. **2007 IEEE International Conference on Portable Information Devices**, p. 1-5, maio. 2007.

KATASONOV, A.; KAYKOVA, O.; KHRIYENKO, O. Smart semantic middleware for the internet of things. **Proceedings of the 5-th ...**, 2008.

KHADILKAR, V.; KANTARCIOGLU, M. **Jena-HBase: A Distributed, Scalable and Efficient RDF Triple Store**. [s.l.: s.n.]. Disponível em: <<http://www.utdallas.edu/~vvk072000/Research/Jena-HBase-Ext/tech-report.pdf>>. Acesso em: 9 fev. 2013.

LOPEZ, M. **The Four Phases Of Enterprise Mobility**. Disponível em: <<http://www.forbes.com/sites/maribellopez/2012/09/27/the-four-phases-of-enterprise-mobility/>>. Acesso em: 1 out. 2012.

MICHAEL, M. *et al.* Scale-up x Scale-out: A Case Study using Nutch/Lucene. **2007 IEEE International Parallel and Distributed Processing Symposium**, p. 1-8, 2007.

MICROSOFT. **about on{x}**. Disponível em: <<https://www.onx.ms/#findOutMorePage>>. Acesso em: 10 fev. 2013.

MOORE, P.; HU, B.; WAN, J. Smart-Context: A Context Ontology for Pervasive Mobile Computing. **The Computer Journal**, v. 53, n. 2, p. 191-207, 4 mar. 2008.

OPEN HANDSET ALLIANCE. **Alliance members**. Disponível em: <[http://www.openhandsetalliance.com/oha\\_members.html](http://www.openhandsetalliance.com/oha_members.html)>. Acesso em: 1 out. 2012.

PARIDEL, K. *et al.* Middleware for the Internet of Things, Design Goals and Challenges. v. 28, p. 1-7, 2010.

RANGANATHAN, A.; CAMPBELL, R. A middleware for context-aware agents in ubiquitous computing environments. ... **International Conference on Middleware**, p. Baker, Nigel, et al. "Context-aware systems and im, 2003.

RIMAL, B. P.; CHOI, E.; LUMB, I. A Taxonomy and Survey of Cloud Computing Systems. **2009 Fifth International Joint Conference on INC, IMS and IDC**, p. 44-51, 2009.

ROALTER, L.; KRANZ, M.; ANDREAS, M. A Middleware for Intelligent Environments and the Internet of Things. **Ubiquitous Intelligence and Computing**, v. 6406, p. 267-281, 2010.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence, A Modern Approach**. 3. ed. [s.l.] Prentice Hall, 2009. v. 82p. 1152

SAHA, A. K. A Developer's First Look At Android. **Linux for you**, p. 48-50, 2008.

SATYANARAYANAN, M. Pervasive computing: vision and challenges. **Ieee Personal Communications**, v. 8, n. 4, p. 10-17, 2001.

SAVITZ, E. **Gartner: Top 10 Strategic Technology Trends For 2013**. Disponível em: <<http://www.forbes.com/sites/ericsavitz/2012/10/23/gartner-top-10-strategic-technology-trends-for-2013/>>. Acesso em: 24 out. 2012.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. ... **Applications, 1994. WMCSA 1994. ...**, 1994.

SHADBOLT, N.; HALL, W.; BERNERS-LEE, T. The semantic web revisited. **Intelligent Systems, IEEE**, v. 21, n. 3, p. 96-101, maio. 2006.

SONG, Z.; ALVARO, A. C.; MASUOKA, R. Semantic Middleware for the Internet of Things. **Internet of Things IOT 2010**, v. 120, n. Nov, p. 1-8, 2012.

VIRKI, T. **Android to beat Windows in 2016: Gartner**. Disponível em: <<http://www.reuters.com/article/2012/10/24/us-android-research-idUSBRE89N11J20121024>>. Acesso em: 25 out. 2012.

**W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax.**

Disponível em: <<http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>>. Acesso em: 20 jan. 2013.

**W3C. OWL Web Ontology Language Guide.** Disponível em: <<http://www.w3.org/TR/owl-guide/>>. Acesso em: 30 out. 2012.

**W3C. SPARQL is a Recommendation.** Disponível em:

<[http://www.w3.org/blog/SW/2008/01/15/sparql\\_is\\_a\\_recommendation/](http://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/)>. Acesso em: 30 out. 2012.

**W3C. OWL 2: Web Ontology Language Document Overview.** Disponível em:

<<http://www.w3.org/TR/owl2-overview/>>. Acesso em: 30 out. 2012.

WEISER, M. The Computer for the 21 st Century. **Scientific American**, v. 3, n. 3, p. 94-104, 1991.

WOOLDRIDGE, M. Intelligent Agents. *In*: WEISS, G. (Ed.). **Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence**. [s.l.] The MIT Press, 1999. .

ZHOU, S.; POLLARD, J. K. Position measurement using Bluetooth. **IEEE Transactions on Consumer**, v. 52, n. 2, p. 555-558, 2006.