UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO
EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

Andrigo Dametto

UMA ARQUITETURA PARA GERENCIAMENTO E RECOMENDAÇÃO DE AÇÕES BASEADAS EM CONTEXTO LÓGICO MEDIANTE DISPOSITIVOS MÓVEIS

São Leopoldo
2012

Andrigo Dametto

UMA ARQUITETURA PARA GERENCIAMENTO E RECOMENDAÇÃO DE AÇÕES BASEADAS EM CONTEXTO LÓGICO MEDIANTE DISPOSITIVOS MÓVEIS

Dissertação apresentada como requisito parcial para obtenção do título de Mestre 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
2012

CATALOGAÇÃO NA PUBLICAÇÃO (CIP)

D157a Dametto, Andrigo

Uma arquitetura para gerenciamento e recomendação de ações baseadas em contexto lógico mediante dispositivos móveis / Andrigo Dametto. - 2013.

70 f.

Dissertação (mestrado) – Universidade do Vale do Rio dos Sinos, Unidade Acadêmica de Pesquisa e Pós-Graduação, Programa Interdisciplinar de Pós-Graduação em Computação Aplicada, São Leopoldo, 2013.

Orientação: Sérgio Crespo Coelho da Silva Pinto

1. Informática - Software 2. Dispositivos móveis. 3. Engenharia de software 4. Agentes de sofware. 5. Sistemas de localização. 6. Sistemas de recomendação I.Título

CDU: 004.4

Ficha catalográfica elaborada por Maristela Hilgemann Mendel – CRB-10/1459

Dedico este trabalho
às pessoas mais importantes da minha vida:
minha noiva Karin, meus pais José e Santinha e minha irmã Andressa.
Vocês foram muito importantes para mim
nessa caminhada.

Resumo

Este trabalho elabora de uma arquitetura de software que contempla dentro de dispositivos móveis na plataforma Android, a coleta de informações de contexto físico de localização (informações que são apenas coletadas em ambientes externos) e geração de contexto lógico de localização (informações que precisam de um processamento dos dados para ser encontradas em ambientes internos), estas informações são armazenadas em uma estrutura *Web Semântica* a qual sofrerá inferências para gerar mais um contexto lógico de recomendação de uso de recursos disponíveis no dispositivo móvel e anteriormente utilizados pelo usuário em um dado instante e local. A funcionalidade desta arquitetura será verificada com a construção de um protótipo na plataforma Android.

Um dos desafios deste trabalho será coletar o contexto lógico de localização do dispositivo em locais internos, como prédios e casas, onde a intensidade do sinal do sistema de posicionamento global (GPS) é insuficiente para ser identificada, portanto neste trabalho será utilizado sensores acelerômetro e giroscópio presentes nos dispositivos móveis para calcular seu deslocamento. A localização interna será integrada a localização externa, formando um percurso contínuo. As informações coletadas no contexto físico são armazenadas em uma ontologia dentro do dispositivo móvel e sincronizadas com um servidor remoto.

Outro desafio deste trabalho é o desenvolvimento de um agente de software que através dos dados armazenados na ontologia local, faz inferências nos dados armazenados na forma de *Web Semântica* e disponibiliza recomendações de uso de um determinado recurso, fundamentado apenas nos dados históricos de utilização destes recursos, relacionando a aproximação em determinado local com a frequência no tempo em relação ao mesmo horário do dia ou ao mesmo dia da semana e ao mesmo dia do mês.

O armazenamento do contexto coletado, em uma estrutura *Web Semântica*, possibilita a união destas informações com demais informações coletadas de outros dispositivos contendo contextos que caracterizem um equipamento, um indivíduo ou uma sociedade.

O resultado esperado da arquitetura apresentada neste trabalho, será o maior grau possível de precisão na posição geográfica identificada e a coerência das recomendações de uso de recursos disponíveis no dispositivo móvel em um dado instante e local.

Palavras chave: Contexto lógico, sistemas de localização externa e interna, *Web* Semântica, Android, sistema de recomendação.

ABSTRACT

This paper elaborates a software architecture that addresses within mobile devices on the Android platform, collecting information from the physical context of location (only information that is collected outdoors) and generation of logical context of location (information they need processing of the data to be found indoors) and stores this information in a Semantic Web structure which suffer inferences to generate a context logical of recommendation to use resources available on the mobile device and used previously by the user at a given time and local. The functionality of this architecture will be test by construction a prototype on the Android platform.

One of the challenges of this work will be to collect the context of logical device location in indoor locations such as buildings and houses where the signal strength of the Global Positioning System (GPS) is insufficient to be identified, so this work will be used and accelerometer sensors gyroscope present in mobile devices to calculate your speed and direction. The location will be integrated inside the external location, forming a continuous path. The information collected in the physical context is stored in the ontology within the mobile device and synchronized with a remote server.

Another challenge of this work is the development of a software agent that through data stored in the ontology on device, makes inferences on the data stored in the form of Web Semantic and provides recommendations for use of a given resource, based only on historical data of these resources by relating the approach in a certain place with the frequency in time over the same time of day or the same day of the week and the same day of the month.

The architecture of this work is being called and Context Manager is integrated with the other two studies did not present this work: a Semantic Desktop with the task of identifying a resource that is being used to send and manager context; and Context's Federation, serving as a remote server, with the task of receiving context data collected by the context manager.

The storage of context collected in a Web Semantic structure enables the union of this information with other context information that characterize a device, an individual or a society.

The expected outcome of the architecture presented here will be the greatest possible degree of accuracy in the identified geographical position and consistency of recommendations for the use of resources available on the mobile device at a given time and place.

Keywords: Context logical, location systems external and internal, Web Semantic, Android, recommendation system.

Lista de figuras

Lista de códigos fontes

Lista de equações

Lista de tabelas

LISTA DE SIGLAS

GPS	*Global Positioning System*
OWL	*Web Ontology Language*
RDF	*Resource Description Framework*
XML	*eXtensible Markup Language*
W3C	*World Wide Web Consortium*
SQL	*Structured Query Language*
URI	*Uniform Resource Identifier*
J2ME	*Java 2 Platform, Micro Edition*
JVM	*Java Virtual Machine*
API	*Application Programming Interface*
NFC	*Near Field Communication*
GSM	*Global System for Mobile Communications*

SUMÁRIO

1 INTRODUÇÃO

O aumento de tecnologias de hardware disponíveis em um dispositivo móvel, como por exemplo, GPS (*Global Positioning System*), processadores mais robustos, maior capacidade de memória, acesso a internet, possibilitaram o surgimento de diversos tipos de software e assim, maior uso do dispositivos móvel por seus respectivos usuários. E para tornar ainda mais acessível, os computadores estão se tornando mais portáveis, consequentemente as pessoas desejam acessar as informações a qualquer momento e em qualquer lugar.

Um desafio para a computação móvel é dispor de aplicações capazes de perceber e explorar as características dinâmicas do ambiente em que estão inseridas. Para isso é fundamental a existência de uma infraestrutura que permita à essas aplicações tirar proveito dessas características e adaptar seu comportamento de acordo com diversos contextos em que podem estar presentes. Um pré-requisito para atender esta expectativa é disponibilizando uma aplicação base, para identificar situações ou ações que podem estar acontecendo naquele instante, gerando uma recomendação para as demais aplicações.

Uma definição do contexto é dada por (ABOWD et al. 1999), que argumentam que o contexto é qualquer informação que pode ser utilizado para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto que é relevante para a interação entre o usuário e a aplicação, incluindo o utilizador e a própria aplicação.

O contexto mesmo tendo esta definição que abrange qualquer tipo de informação que caracterize ou identifique algo, ela tem a necessidade de ser identificada automaticamente ou descrita manualmente, por exemplo, o humor de um usuário é uma informação de contexto específico e de difícil identificação, portanto, para (GONZÁLEZ, 2009) os elementos de informação de contexto seriam limitados à seguinte classificação:

- Localização, por exemplo: posição exata como um ponto geográfico (latitude, longitude e altitude), ou o tipo do local como casa, prédio ou ponte;
- Atividade, caracterizando “o que” está sendo realizado por alguém ou alguma coisa, em um local e em um tempo;
- Identidade, descrevendo informações referentes ao sujeito, seja ele uma pessoa ou alguma coisa, das atividades realizadas, por exemplo: nome, e-mail, marca e potência;
- Tempo, indica o momento que o sujeito realizou a uma atividade em um local específico.

Para cada classificação acima, ou outra nova classificação desejada, existe outras duas subclassificações de contexto:

- Contexto físico é semelhante ao contexto externo acima mencionado, também se refere a informações do ambiente em geral, valores diretamente captados por sensores e normalmente é atualizado com frequência para ter sempre um valor adequado da situação atual (GONZÁLEZ, 2009).
- Contexto lógico derivado a partir do contexto físico, um valor obtido a partir do raciocínio ou processamento do contexto físico. Mais difícil de obter, mas também muito mais complexo e mais significativo. Por exemplo: se uma determinada informação sobre a atividade refere-se ao trabalho ou ao estudo; ou ainda caracterizar o humor de um usuário a partir do conjunto de informações recolhidas; ou até mesmo realizar estimativas diversas, tendo como parâmetro as informações de contexto-físico (GONZÁLEZ, 2009).

As duas subclassificações de contexto, contexto físico e contexto lógico, também são encontradas na literatura como sendo respectivamente: contexto de nível baixo e contexto de

nível alto (MOREIRA; SANTOS, 2005).

Para que aplicações façam uso das informações de contexto em que estão sendo utilizadas, independentemente de quais informações sejam elas, é necessário que estas informações primeiramente sejam coletadas e organizadas de forma compreensiva. Qualquer informação que envolva o dispositivo (por exemplo: conectividade, custo de processamento e consumo de memória), o usuário (por exemplo: perfil do usuário, estado de espírito, pessoas próximas) ou os dois juntos (por exemplo: posição geográfica, posição em ambientes internos, velocidade) pode ser utilizadas pelas aplicações para disponibilizar automaticamente os recursos de forma coerente ao desejo do usuário.

Informações referentes a contextos submetem a extrair informações que caracterizam o ambiente e usuário de um sistema, informações estas que são em alguns momentos subjetivas e não são coletadas de fora estática. O uso de Web Semântica possibilita a extração de informações que podem estar registradas de forma ambígua (BERNERS-LEE; LASSILA, 2001). A Web Semântica é frequentemente utilizada por computadores, mas seu uso em dispositivos móveis é recente e de forma diferente para cada plataforma móvel.

1.1 **Motivação**

Como motivação a este trabalho está a situação em que se encontram os sistemas nos computadores e principalmente em dispositivos móveis, os quais necessitam da intervenção ou solicitação do usuário e muito pouco existe sobre a interação automática com o ambiente do usuário. Desta forma, estudam-se modelos que utilizem e identifique de forma automática os diversos contextos em que um usuário se encontra. Isso servirá como base para geração de informação na forma de recomendação de uso de recursos, e disponível para qualquer outro aplicativo, ou até para o próprio sistema operacional. Disponibilizar as informações de contexto coletadas servirá também como base para sistemas instalados em outros hardwares utilizados pelo usuário, ou de uma sociedade.

1.2 Questão da pesquisa

A questão central deste trabalho é: Como modelar uma arquitetura de software para predição e recomendação de ações ao usuário de dispositivos móveis, baseado em seu contexto lógico armazenado em ontologias?

1.3 Objetivos

Este trabalho apresenta um modelo de monitoramento e coleta de informações que envolvem o contexto físico: localização, tempo e situação de movimento durante o uso de qualquer software (recursos) no dispositivo móvel, além de dados que caracterizam e identificam estes recursos e o próprio dispositivo móvel. Na coleta do contexto físico localização tem como tarefa a identificação da posição em um ambiente interno, que não dispõem de sinal Global Positioning System (GPS), utilizando-se para isso de sensores presentes nos dispositivos os quais medem a aceleração, giro e a indicação do norte magnético através de uma bússola. Desta forma o caminho percorrido por um dispositivo não sofrerá interrupções, ele será contínuo, integrando ambientes externos, utilizando-se do sinal GPS, e ambientes internos, utilizando-se da resposta dos sensores do próprio dispositivo móvel.

Além disso, a arquitetura fará uso das informações do contexto físico coletadas, para disponibilizar um contexto lógico: uma função de recomendação que retorna a informação de que o dispositivo encontra-se com alguma probabilidade de utilizar um determinado recurso no dispositivo móvel.

Faz parte do trabalho organizar e armazenar as informações de contexto em uma estrutura *Web Semântica* no formato de ontologia OWL-DL, para assim, melhor gerenciar as informações de contexto coletadas e possibilitar em trabalhos futuros, a realização de novas inferências e novas informações de contexto lógico a partir das informações de contexto coletadas neste trabalho.

1.4 Metodologia

A metodologia deste trabalho inicia pela identificação do problema a ser resolvido para que, a partir disto, possa se realizar uma revisão bibliográfica sobre cada parte do assunto proposto referente a elaboração de uma arquitetura de software que contempla, dentro de dispositivos móveis Android, a coleta de informações de contexto físico de localização (informações que são apenas coletadas em ambientes externos) e geração de contexto lógico de localização (informações que precisam de um processamento dos dados para serem encontradas em ambientes internos) e armazenando estas informações em uma estrutura *Web Semântica* a qual sofrerá inferências para gerar mais um contexto lógico referente a recomendação de uso de recursos em um dado instante e local.

Após a revisão bibliográfica foi feita uma análise dos principais trabalhos relacionados que serviram de base para verificar até onde o assunto já havia sido explorado, bem como verificar a abrangência de algumas soluções já propostas por outros autores.

Com base na revisão bibliográfica e nos trabalhos relacionados foi elaborada uma arquitetura de software baseada em agentes de software que possibilitasse o gerenciamento dos contextos coletados e seu armazenamento realizado em uma estrutura *Web Semântica*.

A arquitetura apresentada foi incorporada em um protótipo na plataforma Android, com propósito de provar o seu funcionamento.

O passo seguinte foi a verificação do protótipo com a utilização do mesmo em locais com ambientes internos e externos. Esta verificação permitiu avaliar o comportamento da arquitetura sob circunstâncias reais em diferentes locais e possíveis limitações de hardware.

1.5 Organização deste trabalho

O presente trabalho encontra-se dividida em cinco sessões dispostas da seguinte forma:

- no primeiro capítulo o trabalho é contextualizada no problema proposto, as motivações para resolve-lo, os principais objetivos, a metodologia utilizada e a organização do trabalho;
- o capítulo 2 trata do referencial teórico dos conceitos envolvidos no trabalho: *Web Semântica*, Android, agentes de software e *webservices*.
- o capítulo 3 descreve-se alguns trabalhos relacionados com o assunto de coleta de informações de contexto físico e/ou lógico em dispositivos móveis.

- o capítulo 4 descreve a arquitetura de gerenciamento de contexto como defesa do titulo de mestrado e o modelo ao qual este trabalho está inserido.
- o capítulo 5 descreve a forma a qual foi verificado o protótipo desta arquitetura, os resultados atingidos e problemas identificados.
- o capítulo 6 descreve a conclusão a partir da análise do trabalho atingido, suas dificuldades e possíveis utilidades deste trabalho como base para demais trabalhos futuros.

2 TECNOLOGIAS UTILIZADAS

2.1 *Web Semântica*

Um dos maiores desafios na computação desde o seu início é viabilizar um entendimento humano-computador de modo pleno. Para isso, foram criadas linguagens, de forma que instruções representadas através delas sejam entendidas pelos computadores. Mas isso estabelece apenas um entendimento sintático (BERNERS-LEE; LASSILA, 2001). Quando entra-se no âmbito semântico, esse entendimento não é assim tão simples. Como fazer que uma máquina extraísse um significado de uma informação, que muitas vezes pode ser dotada de ambiguidade?

É nesse contexto que a *Web Semântica* se insere, para justamente organizar e representar informações de modo que os agentes computacionais possam trabalhar com elas, aumentando a eficácia e eficiência dos sistemas. Segundo a definição de Tim Berners-Lee, criador da *World Wide Web* e diretor do *World Wide Web Consortium* : "A *Web Semântica* não é uma web separada, mas uma extensão da atual, na qual a informação é utilizada com significado bem definido, aumentando a capacidade dos computadores para trabalharem em cooperação com as pessoas" (BERNERS-LEE; LASSILA, 2001). Na *Web Semântica*, o conteúdo está organizado de modo a permitir inferências, ao contrário da web tradicional, onde buscas retornam apenas as referências ao termo procurado, sem nenhuma distinção semântica. Para a representação da Web Semântica foram criadas as seguintes linguagens:

- RDF (*Resource Description Framework*)
- DAML+OIL
- OWL (*Web Ontology Language*)

A *Web Semântica* dividi-se em três camadas:

- Camada de Estrutura: Responsável por organizar os dados e definir seu significado
- Camada de Esquema: Responsável por definir as relações entre os dados. Eliminando conflitos de terminologia quando uma informação é solicitada.
- Camada Lógica: Responsável por definir mecanismos para fazer inferência sobre os dados, tendo para isso, um conjunto de regras de inferência utilizadas pelos a agentes computacionais para poder raciocinar sobre as estruturas de dados.

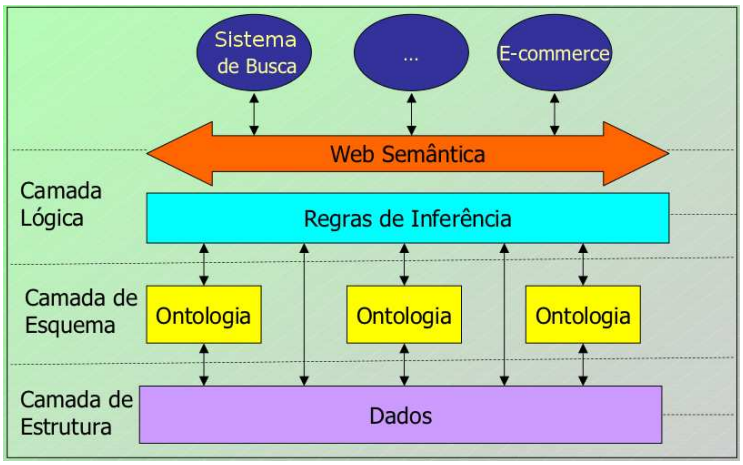

Figura 1. Camadas da Web Semântica.

Para suportar esses novos requisitos da Web Semântica faz-se necessário a construção de uma arquitetura coerente. Por causa disso, a W3C, organização internacional responsável pela criação de padrões para a web, já vem há muito tempo trabalhando em padrões para a Web Semântica, até chegar na arquitetura representada na figura 2. Dividida em camadas, ela faz uso de diversas recomendações, como o XML (eXtensible Markup Language), XML Schema, RDF e RDF-S (RDF Schema), para servirem de base para uma linguagem de representação de ontologias. Essa linguagem usa as características desses outros padrões, como a capacidade de marcação e estruturação do XML e o modelo de dados do RDF, e os potencializa, aumentando sua capacidade de expressão.

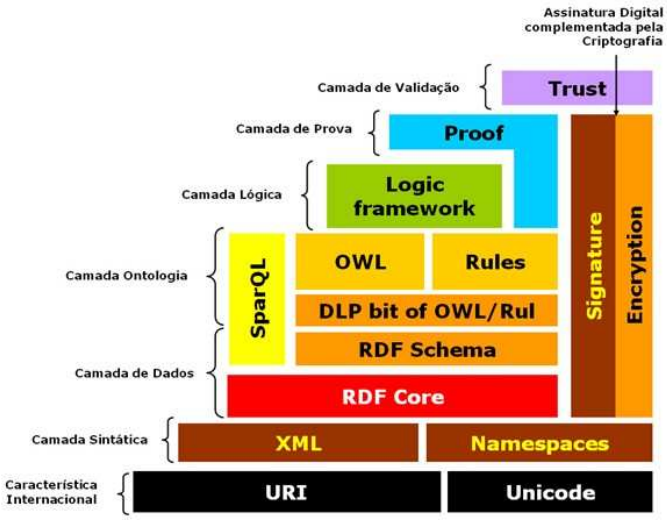

Figura 2. Arquitetura proposta pela W3C para a Web Semântica (SANTOS; ALVES, 2009).

2.1.1 Ontologias

O termo “Ontologia” origina-se da filosofia e estuda o ser e sua existência, sendo definido como a teoria da natureza da existência. Na década de 80 foi adotado por pesquisadores de inteligência artificial, os quais identificaram sua aplicabilidade e construíram modelos computacionais com algum tipo de raciocínio automatizado (GRUBER, 2007). Na década seguinte, ontologias começaram a ser tratadas como parte integrante de sistemas baseados em conhecimento, sendo definida como uma especificação explicita de conceitualizações. Na informática a ontologia é utilizada para referenciar o que existe em um modelo de sistema (WONGTHONGTHA et al, 2009).

O uso de computadores disseminou-se em diversos campos do conhecimento humano, dentre eles, prover informações para apoiar a resolução de problemas. Entretanto, como resultado deste esforço, um novo problema surgiu: o excesso de recursos de informação, associado à falta de semântica para guiar uma busca por recursos realmente relevantes para o contexto em mãos. Assim como a falta de informação, o excesso de recursos de informação

também constitui um problema, onde pode não ser possível coletar em tempo hábil a informação necessária para apoiar a tomada de decisão durante a resolução de problemas.

Em linhas gerais para a construção de uma ontologia precisam-se definir as informações para:

- Domínio: o domínio representa a área de atuação da ontologia que está sendo criada. Ela serve como guia para a construção das informações seguintes;
- Classes: as classes detalham os membros que formam o domínio em questão;
- Indivíduos: os indivíduos representam as instâncias das classes criadas;
- Propriedades: as propriedades são as características encontradas em um indivíduo;
- Objetos (relações): os objetos representam as relações que existem entre os indivíduos de diferentes classes. É através destas relações que é formado o conhecimento de uma ontologia.

Uma ontologia apresenta os seguintes elementos:

- Hierarquia de conceitos:
    - Entidades: onde cada entidade é definida por conjunto de pares atributo-valor. Elas representam as classes dos modelos orientados a objetos, as entidades do modelo relacional ou aos termos do modelo lógico;
    - Relações: correspondem às associações, agregações e atributos dos modelos orientados a objetos cujos valores são objetos e também correspondem às relações do modelo relacional e aos predicados do modelo lógico;
- Restrições: correspondente aos valores possíveis dos atributos dos conceitos, por exemplo, as assinaturas de classes em modelos orientados a objetos, normas quantificadas em modelos lógicas e restrições de integridade nos esquemas de banco de dados;
- Regras Dedutivas: sobre os atributos ou conjuntos de conceitos. Elas permitem inferência automática da existência de instâncias. Correspondem às regras dos sistemas especialistas e programação em lógica, aos métodos dos modelos orientados a objetos e às visões em banco de dados;
- Instâncias de Conceitos: É a definição de entidades e relações específicas (indivíduos), correspondentes aos fatos de sistemas especialistas e programação em lógica, aos objetos dos modelos orientados a objetos e aos dados dos bancos de dados.

O desenvolvimento de uma ontologia requer um esforço considerável a fim de atingir um resultado satisfatório. Diante disso e baseada na engenharia de software, surge a engenharia de ontologias, preocupando-se com o conjunto de atividades, o processo de desenvolvimento de ontologias, o ciclo de vida de ontologias, as metodologias para desenvolver ontologias e as ferramentas e linguagens para a construção de ontologias (GÓMEZ; FERNÁNDEZ; CORCHO, 2004).

Como principais atividades encontradas nas literaturas de engenharia de ontologias, destacam-se:

- **Especificação**: identificar o propósito e o escopo da ontologia, isto é, o porque da construção da ontologia e quais as intenções de uso e usuários da ontologia.
- **Conceitualização**: descrever, em modelo conceitual, a ontologia a ser construída, de

acordo com as especificações encontradas no estágio anterior. Cabe ressaltar que o modelo conceitual de uma ontologia pode ser construído mediante ferramentas formais e informais. Tal modelo consiste em conceitos do domínio, as relações entre os conceitos e as propriedades dos conceitos.

- **Formalização**: transformar a descrição conceitual em um modelo formal, onde os conceitos são definidos através de axiomas que restringem as possíveis interpretações de seu significado e também organizados hierarquicamente através de relações de estruturas, tais como “é-um” ou “parte-de”.
- **Implementação**: implementar a ontologia formalizada em uma linguagem de representação do conhecimento.
- **Manutenção**: atualizar e corrigir a ontologia desenvolvida, de acordo com o surgimento de novos requisitos.
- **Verificação**: garantir a correção da ontologia de acordo com o entendimento aceito sobre o domínio em fontes de conhecimento especializadas;
- **Validação:** garantir que a ontologia corresponde a sua suposta, de acordo com os documentos de especificação de requisitos.
- **Avaliação**: avaliar a ontologia do ponto de vista do usuário, em relação a sua usabilidade e utilidade; e principalmente do ponto de vista da reutilização da ontologia em outras possíveis aplicações.
- **Documentação**: descrever o que, como e por que foi feito. Uma documentação se faz necessária não somente para melhorar a clareza da ontologia, mas também para facilitar a manutenção, uso e reuso.
- Algumas metodologias que surgiram na engenharia de ontologias estão descritas n próximo capítulo.

2.1.2 Metodologias para o desenvolvimento de ontologias

Na literatura sobre o desenvolvimento de ontologias, existem várias metodologias de desenvolvimento propostas, cada qual com suas características em relação ao ciclo de vida de uma ontologia. Por conseguinte, não há uma metodologia estabelecida como padrão para o propósito de desenvolvimento completo de ontologias, pois cada metodologia preocupa-se com determinadas atividades do processo de desenvolvimento em relação a outras (RAUTENBERG et al, 2008).

A seguir estão descritas algumas metodologias adotadas durante o ciclo de vida de ontologias.

Ontology Development 101: para esta metodologia as tarefas consistem apenas em uma lista de processos ou passos iterativos, livremente executados no desenvolvimento de ontologias, sem definir a ordem a qual estes passos devem ser trabalhados. A Ontology Development 101 é formada pelos setes passos a seguir:

- Determinar o domínio e o escopo da ontologia: deve-se identificar claramente o propósito e os cenários de utilização da ontologia a ser desenvolvida. Para identificar o domínio e o escopo no desenvolvimento de uma ontologia, pode-se realizar as seguintes questões: “O que abrange o domínio da ontologia?”, “para que se utilizará a ontologia?”, “que questões a ontologia deveria responder?”, “quem utilizará e manterá a ontologia?”.
- Considerar o reuso de ontologias existentes: é aconselhável verificar a existência de

ontologias que podem ser reutilizadas em um novo projeto de ontologia, a fim de não se "reinventar a roda" ou proporcionar a interação da ontologia desenvolvida com outras aplicações.

- Enumerar termos importantes do domínio da ontologia: relacionar uma lista de termos presentes no discurso do domínio da ontologia. A relação de termos é importante para os passos subsequentes do guia, como definir classes, definir propriedades e definir instâncias.
- Definir as classes do domínio e a hierarquia de classes: a partir da lista de termos, extraem-se aqueles que descrevem objetos, os quais genericamente representam classes. Com um conjunto de classes definido, deve-se organizar as classes de forma hierárquica, considerando um nível de abstração mais geral em direção as classes específicas.
- Definir as propriedades das classes: a partir da lista remanescente de termos, deve ser observados se eles correspondem a propriedades de dados ou de relações de classe para uma determinada classe.
- Definir as restrições das propriedades: caso uma propriedade de classe seja de dados, observa-se o tipo de dado que a propriedade comporta (string ou número, por exemplo). Caso a propriedade seja uma relação, deve-se definir a que classes a relação aponta. Restrições sobre cardinalidade e valores válidos para as propriedades também devem ser considerados neste passo.
- Criar as instâncias do domínio: criar instâncias da ontologia a partir da definição das classes, preenchendo suas propriedades de dados e relações.

On-to-knowledge: uma metodologia criada com o intuito de desenvolver ontologias para serem empregadas em Sistemas de Gestão do Conhecimento. Esta metodologia está dividida em 5 fases:

- Estudo de viabilidade: é uma fase anterior ao desenvolvimento de ontologias. O estudo de viabilidade destina-se a identificar problemas e oportunidades de uma organização, objetivando mapear a real necessidade do desenvolvimento de uma ontologia.
- Início da ontologia: na metodologia, o desenvolvimento de uma ontologia inicia-se nesta fase. Fazendo uma analogia ao processo de software, aqui se objetiva produzir documentos de especificação de requisitos, definindo o domínio e objetivos da ontologia, utilizando padrões de projeto, identificando as fontes de conhecimento, definindo atores e cenários, enumerando questões de competência, definindo o ambiente de desenvolvimento da ontologia, entre outros.
- Refinamento: o objetivo desta fase é desenvolver uma ontologia a ser utilizada em um Sistema de Gestão do Conhecimento, de acordo com os documentos produzidos nas fases anteriores. Para tanto, os engenheiros do conhecimento se valem de técnicas de elicitação do conhecimento ao interagir com os especialistas de domínio, modificando e estendendo a ontologia em desenvolvimento na direção de uma versão estável.
- Avaliação: o objetivo desta fase é a aferição da completude e precisão da ontologia mediante a documentação gerada durante o desenvolvimento da ontologia e um *frame* de referência, o qual pode corresponder às questões de competência enumeradas na fase "início da ontologia".
- Manutenção e Evolução: esta é uma fase de responsabilidade da organização que utilizará

a ontologia. É importante ter ciência dos atores responsáveis pela manutenção da ontologia e das regras para sua manutenção.

Methontology : Esta metodologia foi desenvolvida por um grupo de pesquisa em Engenharia de Ontologias da Universidade Politécnica de Madri. A Methontology inclui a identificação do processo de desenvolvimento da ontologia, um ciclo de vida baseado na evolução de protótipos, além de técnicas para realizar cada atividade no gerenciamento, orientado ao desenvolvimento e as atividades de apoio.

Para a Methontology, cada protótipo inicia com um calendário de atividades identificadas para serem executadas em um determinado tempo e com as ferramentas necessárias para completar a ontologia.

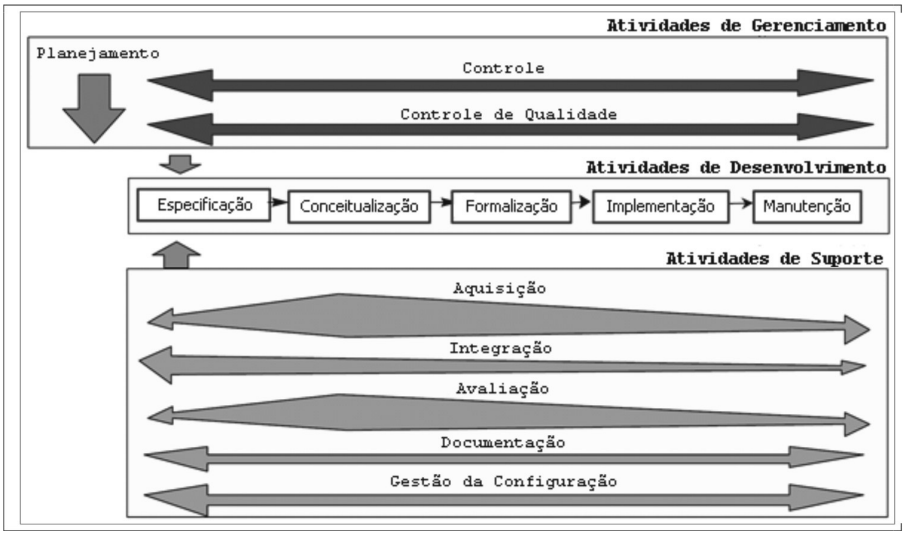

Figura 3. Processo de desenvolvimento de ciclo de vida da metodologia Methontology (RAUTENBERG et al, 2008).

Observa-se na figura 3 que os processos da atividades de gerenciamento, desenvolvimento e de suporte são realizadas simultaneamente.

Nas atividades de suporte observa-se um maior esforço na aquisição do conhecimento, integração e avaliação durante a conceitualização da ontologia e depois diminui durante a formalização e implementação. Isso ocorre devido:

- a maior parte do conhecimento é adquirida no início da construção da ontologia;
- a importância de saber as integrações que a ontologia terá para a elaboração correta da respectiva ontologia;
- a concepção da ontologia deve ser avaliada com precisão para evitar que erros sejam propagados para as fases posteriores do ciclo de vida da ontologia.

2.1.3 OWL

Existem diversas linguagens de representação de ontologias disponíveis, dentre elas *Resource Description Framework* (RDF) (KLYNE; CARROLL, 2004) e *Web Ontology Language* (OWL) (MCGUINNESS; HARMELEN, 2004).

A OWL é uma recomendação da W3C que objetiva prover uma representação eficiente para ontologias. Uma característica importante da linguagem, herdada de RDF, é a capacidade de interligar ontologias distribuídas em diversos sistemas. Em outras palavras, uma determinada ontologia pode referenciar conceitos de outra ontologia. A OWL foi especificamente projetada para as necessidades da *Web* e *Web* Semântica (SHADBOLD; HALL; BERNERS-LEE, 2006).

Arquiteturalmente, a OWL é subdividida em três sub-linguagens (MCGUINNESS; HARMELEN, 2004) conforme representada na figura 4 e descrita abaixo:

• OWL *Lite*: provê estruturas de classificação e restrições muito simples, o que facilita a migração de taxonomias existentes, mas limita as capacidades da linguagem.

• OWL *Description Logics* (DL): provê a máxima expressividade em termos de computação, todas as conclusões são computáveis e todas as computações têm tempo finito, é assim chamada dada a correspondência com a lógica de descrição e inclui todas as construções da linguagem, porém impõe algumas restrições em seu uso.

• OWL *Full*: também provê a máxima expressividade, além da liberdade sintática existente no RDF, porém não há garantias computacionais nas construções.

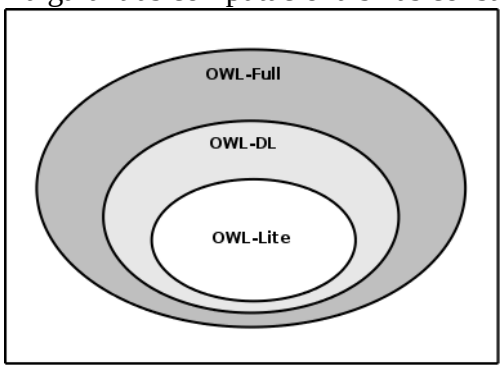

Figura 4. Sub-linguagens da OWL.

2.1.4 PROTÉGÉ

O Protégé é uma plataforma para criação e modelagem de ontologias desenvolvidas pelo Stanford Center for Biomedical Informatics Research4 na Stanford University School of Medicine. Desenvolvida em Java, ele permite uma vasta gama de ações sobre ontologias, que são expandidas pelo seu suporte a plugins. Por ser um projeto de código aberto, ele é mantido por um grande número de colaboradores, contando com parceiros corporativos, acadêmicos e governamentais, além de incentivar seus usuários ao desenvolvimento através do Protégé Programming Development Kit, pacote para criação de plugins que atendam a suas necessidades específicas (FONOU; HUISMAN, 2011).

O OWLViz, um plugin de visualização desenvolvido na Universidade de Manchester, Inglaterra. Ele constrói uma representação em forma de grafo da ontologia - tanto da ontologia declarada, quanto da inferida pelo raciocinador - mostrando todos os relacionamentos existentes. Uma visão da interface desse plugin pode ser vista na figura 5. Esse *plugin* é muito importante por fornecer uma abstração diferente da ontologia, facilitando seu entendimento, principalmente para aqueles que possuem um entendimento mais visual. Há também a possibilidade que tal representação seja salva em diversos formatos de imagem. O OWLViz já acompanha o Protégé em sua instalação.

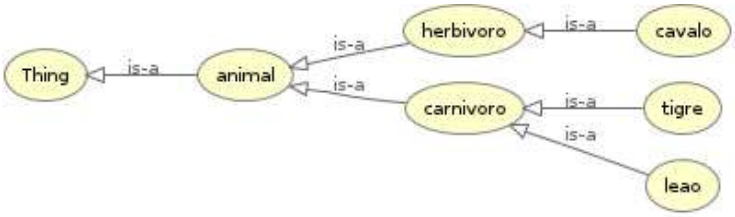

Figura 5. Visualização de uma ontologia com o OWLViz no software PROTÉGÉ.

2.1.5 JENA

JENA é um *framework* em linguagem Java para *Web* Semântica. Ela fornece classes para criar e manipular ontologias nos formatos RDF, DAML+OIL e OWL.

Por se tratar de um framework para programação, será listado a seguir a forma de utilização de sua API (DICKINSON, 2012). Na listagem 1 é demonstrado a criação de uma ontologia utilizando o JENA.

Listagem 1. Criação de ontologia utilizando API Jena.

```
OntModel ontologia = ModelFactory.createOntologyModel
                                        (URIdaOntologia);
```

ou

```
OntModel ontologia = ModelFactory.createOntologyModel
                                        (ProfileRegistry.OWL_LITE_MEN);
```

A segunda opção apresentada na Listagem 1, apresenta a constante OWL_LITE_MEN, a qual representa a criação de uma ontologia OWL-Lite, conforme tabela 1.

Tabela 1. Constantes do Jena que representam formatos de ontologias.

Constante	Representação
DAML_LANG	DAML+OIL
OWL_DL_MEN	OWL-DL
OWL_MEN	OWL-Full
OWL_LITE_MEN	OWL-Lite
RDFS_MEN	RDFS

O carregamento de ontologias no Jena pode ser feito através da leitura de um arquivo, por *streams* de dados ou de um endereço *web*.

Listagem 2. Carregamento de uma ontologia existente.

```
OntModel m = ModelFactory.createOntologyModel();
OntDocumentManager dm = m.getDocumentManager(<arquivo>);
```

A manipulação de uma ontologia subdivide-se em:

- Recursos: todo item de uma ontologia, por exemplo, uma classe ou um indivíduo.

- os recursos contém propriedades para caracterizar o respectivo indivíduo ou classe. Em uma ontologia a propriedade representa o nome de uma relação entre recursos, ou entre um recurso e um valor. A propriedade corresponde a um predicado nas representações lógicas. A tabela 2 descreve as possíveis propriedades que podem ser assumidas em uma ontologia.

- Classes: representa um agrupamento de sub-classes ou de indivíduos.
  - Para demonstrar a manipulação de uma classe a listagem 3, 4 e 5 demonstram respectivamente a criação, a recuperação e a listagem de sub-classes de uma classe.
- Indivíduo: representa o elemento ou informação final.

Na figura 6 exemplifica uma ontologia contendo classes e propriedades assumidas em uma ontologia. Nesta figura, a classe “OrganizedEvent” possui uma propriedade range “hasProccedings” com a classe “Proceeding”.

Listagem 3. Criação de uma classe de ontologia.

```
OntClass classAnimais = m.createClass( "Animais" );
```

Listagem 4. Recuperação de uma classe de ontologia.

```
OntClass classAnimais = m.getOntClass( "Animais" );
```

Listagem 5. Listagem de sub-classes de uma classe de ontologia.

```
OntClass classAnimais = m.getOntClass( "Animais" );
for (Iterator i = classAnimais.listSubClasses(); i.hasNext(); ) {
        OntClass c = (OntClass) i.next();
        System.out.print( c.getLocalName() + " " );
}
```

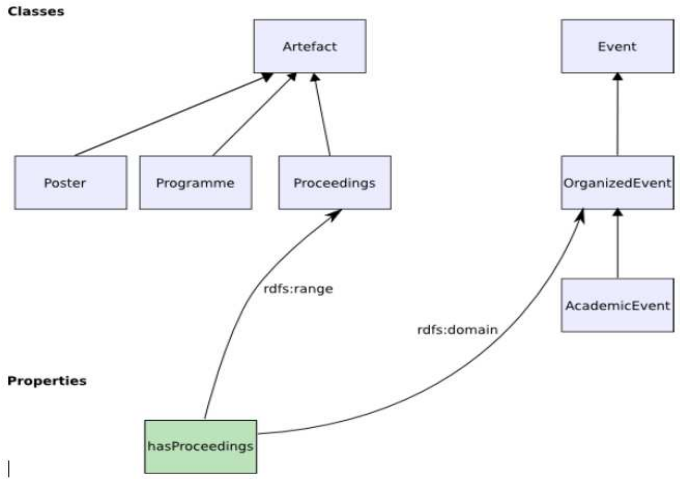

Figura 6. Exemplo de classes e propriedades de uma ontologia.

Tabela 2. Propriedades assumidas em uma ontologia (DICKINSON, 2012).

Atributo	Significado
subProperty	Indica uma sub-propriedade de uma propriedade. Com isso pode-se entender que, se “p” é uma sub-propriedade de “q” e sabe-se que A “p” B é verdadeiro, então conclui-se que também A “q” B é verdadeiro.
superProperty	Indica uma super propriedade de uma propriedade.

domain	Indica a classe ou classes que formam o domínio da propriedade. Multiplos domínios são interpretados como um conjunto.
range	Indica a classe ou classes que formam um conjunto de propriedades.
equivalentProperty	Indica a propriedade que é equivalente a esta propriedade.
inverse	Indica a propriedade que é contrária a esta propriedade. Por exemplo, se 'q' é inverso a 'p' e sabe-se que A 'q' B então infere-se que B 'p' A.

2.1.6 SPARQL

SPARQL é uma linguagem inspirada na Structured Query Language (SQL), que viabiliza a busca de informações em grafos RDF. Uma característica importante da linguagem é a capacidade de compor consultas tendo vários modelos RDF como fonte de dados. Complementarmente, a SPARQL também viabiliza consultas em modelos especificados através de OWL, facilitando a recuperação de indivíduos e propriedades de uma ontologia. A tabela 3 apresenta as principais cláusulas da SPARLQL (PRUD'HOMMEAUX; SEABORNE, 2006).

Tabela 3. Principais cláusulas SPARQL.

Cláusula	Descrição	Sintaxe
PREFIX	Possibilita a criação de abreviaturas para referenciar a Uniform Resource Identifier (URI) dos recursos do grafo.	`PREFIX□ name:□ <http://uri.resource>□`
SELECT	Possibilita a seleção das variáveis que deseja-se obter como resultado da consulta.	`SELECT□ ?var_1□?var_2□?var_n□`
FROM	Possibilita definir o grafo padrão para consulta.	`FROM□<http://uri.resource>□`
FROM NAMED	Possibilita que uma mesma consulta busque dados em mais de um grafo.	`FROM□NAMED□<http://uri.resource>□`
FILTER	Possibilita a filtragem do resultado, comparando variáveis com algum valor.	`FILTER□(?var□<=□10)` `FILTER□(?var□=□'SPARQL')□`

A Figura 7 apresenta um exemplo de consulta simples em SPARQL estruturada da seguinte forma: na parte superior estão os dados do grafo RDF, no meio está a consulta SPARQL propriamente dita e na parte inferior está o resultado obtido. Observando a consulta, pode ser visto na cláusula *SELECT* a seleção da variável *?title*, a qual, na cláusula *WHERE*, é inicializada com dado recebido do recurso <http://purl.org/dc/elements/1.1/title>.

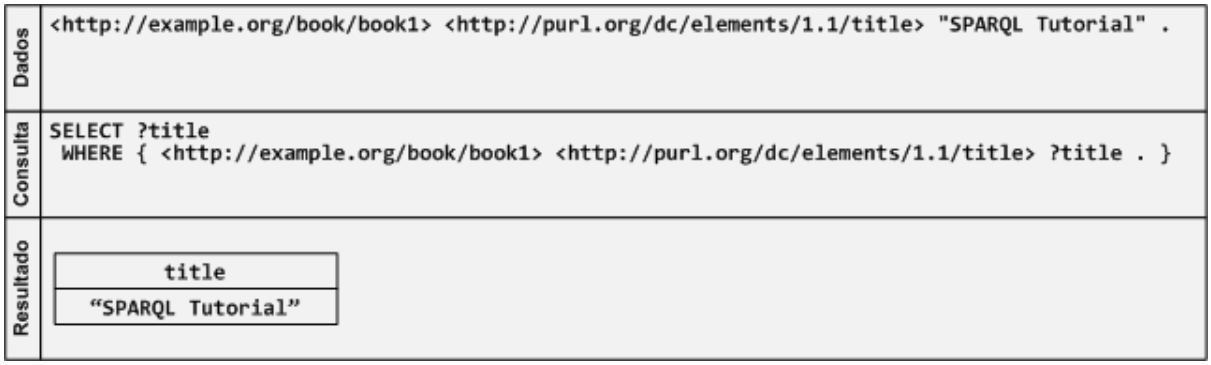

Figura 7. Exemplo de uma consulta com SPARQL.

A Figura 8 apresenta um exemplo de consulta SPARQL que retorna múltiplo valor como resultado. Inicialmente, fica caracterizado na consulta o uso da cláusula PREFIX, abreviando a URI do grafo RDF. Também pode ser visto na cláusula WHERE a definição de duas regras para atribuição de valores, onde todos os registros de *?x* que possuírem nome e e-mail serão atribuídos às variáveis *?name* e *?mbox*, respectivamente.

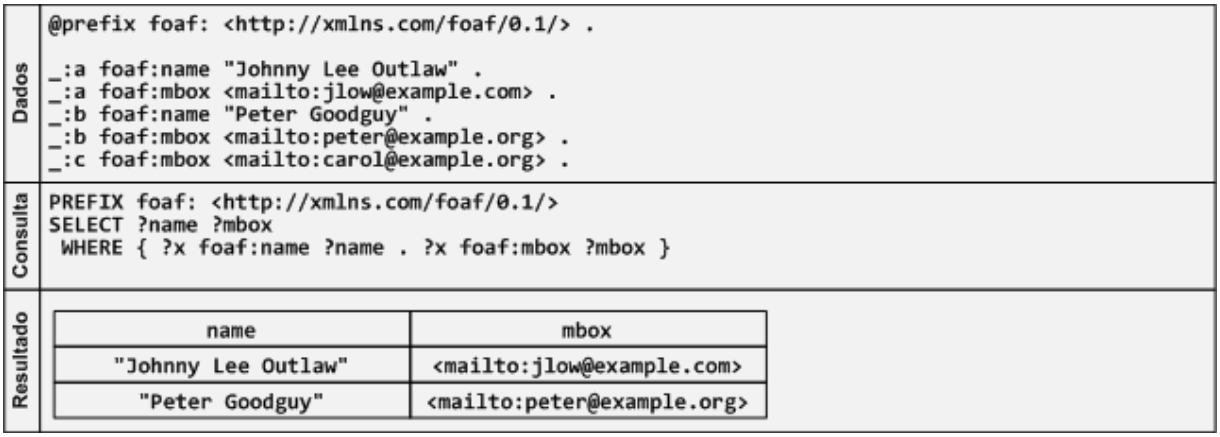

Figura 8. Exemplo de uma consulta com múltiplos retornos no SPARQL.

A Figura 9 tem como resultado o mesmo apresentado na Figura 6, porém este é obtido de uma forma mais elaborada, já que há mais dados no grafo da Figura 8 do que no grafo da Figura 6. Para se obter o mesmo resultado, além de se realizar a atribuição de recursos às variáveis na cláusula WHERE, também é necessário criar regras de filtragem de registros. Estas regras são criadas com a cláusula FILTER, inclusa na cláusula WHERE da consulta SPARQL. Pode-se notar o uso de uma expressão regular para filtrar o resultado da consulta, mas esta não é a única forma, pois a cláusula FILTER permite o uso de expressões aritméticas e relacionais.

Figura 9. Exemplo de uma consulta com filtro de dados no SPARQL.

2.2 Android

Android é uma plataforma completa, contendo um sistema operacional Android e um *framework* Android baseado na linguagem Java J2ME (Java 2 Platform, Micro Edition). O projeto iniciou no ano de 2008 pela empresa Google em conjunto com diversas outras empresas na área de comunicação móvel, tanto na parte de comunicação de telefonias de celulares, quanto fabricantes de celulares.

Android é a primeira plataforma móvel totalmente livre e de código aberto (open-source). Um motivo para ter uma rápida evolução, uma vez que diversos programadores de todo o mundo poderão contribuir para melhorar a plataforma [Lecheta (2010)].

O sistema operacional do Android é baseado no kernel 2.6 do linux, que também é de código aberto. A linguagem de programação Android, por ser baseada na linguagem Java, necessita de uma máquina virtual para executar seus programas, mas no caso do Android, ele não utiliza a máquina virtual Java (JVM), foi criada uma máquina virtual específica para o Android, a qual é otimizada para executar em dispositivos móveis.

2.2.1 Serviços baseados em localização

A versão 4 da plataforma Android evoluiu e facilitou o uso dos serviços baseados em localização, principalmente em dois itens: o mapeamento e o uso da API base de localização. O mapeamento refere-se ao uso dos mapas do pacote *com.google.android.maps* e o pacote de localização é o *android.location*. O mapeamento prove facilidades para exibir e manipular mapas, por exemplo: zoom em um plano, mudar o tipo de mapa a ser exibido, ou mesmo adicionar informações em um determinado ponto do mapa. A API de localização fornece informações de GPS em tempo real (KOMATINENI; MACLEAN, 2012).

A API (Application Programming Interface) de mapeamento solicita serviços dos servidores do Google, portanto é necessário que tenha conectividade com a Internet para o uso desta API. Para o uso dos mapas também deve-se aceitar o Termo de Serviço Google, o qual retorna uma chave para a API de mapas.

A necessidade neste trabalho é o uso da API de localização a fim de encontrar a posição geográfica do dispositivo móvel. O uso desta API requer a permissão de

android.permission.ACCESS_FINE_LOCATION.

2.2.2 Sensores

Com o uso de sensores, o usuário de um dispositivo móvel tem mais formas de interagir com o dispositivo além do teclado ou tela. Os sensores vão ao encontro do ambiente em que o usuário se encontra e a maneira como o usuário se movimenta neste ambiente. Ainda não temos sensores para identificar o comportamento do usuário do dispositivo, provável futuros sensores.

Nos dispositivos que utilizam Android, os sensores são peças de hardware que foram adicionadas para dispor uma determinada informação física do ambiente em que está situado o dispositivo para as aplicações de software. Muitos destes sensores são utilizados como comandos para jogos a partir dos movimentos realizados com o dispositivo. A versão 4 do Android dispõem API para controlar os seguintes sensores:

- sensor de luz;
- sensor de proximidade;
- sensor de temperatura;
- sensor de pressão;
- sensor de giroscópio;
- acelerômetro;
- sensor de campo magnético;
- sensor de orientação;
- sensor de gravidade;
- sensor de aceleração linear;
- sensor de rotação do vetor;
- sensor de umidade relativa do ar;
- sensor de área de comunicação (*Near Field Communication* – NFC).

Não são todos dispositivos que dispõem de todos os sensores listados acima, a lista é distinta para cada modelo de aparelho. Além disso, o emulador do Android contém apenas um acelerômetro.

Existem várias maneiras de descobrir se um dispositivo contém determinado sensor, por exemplo, instalando uma aplicação que utiliza um sensor específico, ou, aqui está descrito na listagem 6 o código fonte para a classe *MainActivity* de uma simples aplicação para identificar os sensores presentes em um dispositivo (KOMATINENI; MACLEAN, 2012).

Listagem 6: Código Android para listagem de sensores (KOMATINENI; MACLEAN, 2012).

```
public class MainActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    TextView text = (TextView)findViewById(R.id.text);
    SensorManager mgr =
```

```
        (SensorManager) this.getSystemService(SENSOR_SERVICE);

    List<Sensor> sensors = mgr.getSensorList(Sensor.TYPE_ALL);
    StringBuilder message = new StringBuilder(2048);
    message.append("The sensors on this device are:\n");
    for(Sensor sensor : sensors) {
      message.append(sensor.getName() + "\n");
      message.append(" Type: " +
      sensorTypes.get(sensor.getType()) + "\n");
      message.append(" Vendor: " +
      sensor.getVendor() + "\n");
      message.append(" Version: " +
      sensor.getVersion() + "\n");
      message.append(" Resolution: " +
      sensor.getResolution() + "\n");
      message.append(" Max Range: " +
      sensor.getMaximumRange() + "\n");
      message.append(" Power: " +
      sensor.getPower() + " mA\n");
    }
    text.setText(message);
  }

  private HashMap<Integer, String> sensorTypes =
                new HashMap<Integer, String>();
  {
    sensorTypes.put(Sensor.TYPE_ACCELEROMETER, "TYPE_ACCELEROMETER");
    sensorTypes.put(Sensor.TYPE_AMBIENT_TEMPERATURE,
          "TYPE_AMBIENT_TEMPERATURE");
    sensorTypes.put(Sensor.TYPE_GRAVITY, "TYPE_GRAVITY");
    sensorTypes.put(Sensor.TYPE_GYROSCOPE, "TYPE_GYROSCOPE");
    sensorTypes.put(Sensor.TYPE_LIGHT, "TYPE_LIGHT");
    sensorTypes.put(Sensor.TYPE_LINEAR_ACCELERATION,
          "TYPE_LINEAR_ACCELERATION");
    sensorTypes.put(Sensor.TYPE_MAGNETIC_FIELD, "TYPE_MAGNETIC_FIELD");
          sensorTypes.put(Sensor.TYPE_ORIENTATION,
          "TYPE_ORIENTATION (deprecated)");
          sensorTypes.put(Sensor.TYPE_PRESSURE, "TYPE_PRESSURE");
          sensorTypes.put(Sensor.TYPE_PROXIMITY, "TYPE_PROXIMITY");
```

```
        sensorTypes.put(Sensor.TYPE_RELATIVE_HUMIDITY,
            "TYPE_RELATIVE_HUMIDITY");
        sensorTypes.put(Sensor.TYPE_ROTATION_VECTOR,
                "TYPE_ROTATION_VECTOR");
        sensorTypes.put(Sensor.TYPE_TEMPERATURE,
                "TYPE_TEMPERATURE (deprecated)");
    }
}
```

Para utilizar um sensor deve-se apenas indicar qual sensor será utilizado através da classe *android.hardware.SensorManager* e o método *getDefaultSensor(int type)*. Este método necessita como parâmetro do número do tipo de sensor a ser utilizado, através de uma respectiva constante, por exemplo, *Sensor.TYPE_ACCELEROMETER*, *Sensor.TYPE_GIROSCOPE* entre outros.

Neste capítulo também está detalhado o funcionamento de alguns sensores que podem ser utilizados para, em conjunto, mapear o deslocamento de um dispositivo ao longo de um determinado tempo e local.

2.2.2.1 Utilizando o acelerômetro

Com o uso do sensor acelerômetro é possível determinar o deslocamento físico de um dispositivo no espaço relativo às três coordenadas conforme mostrado na figura 10.

As coordenadas do sistema do acelerômetro contém seu ponto (0,0,0) no campo superior esquerdo, e a partir dele, o valor de X aumenta para a esquerda, o valor de Y aumenta para baixo, e o valor de Z aumenta para a parte frontal do dispositivo. O acelerômetro informa a velocidade em metros por segundo em cada uma das direções, portanto, considerando a aceleração da gravidade terrestre, ao manter o dispositivo parado em uma superfície plana, o eixo do Z informará uma velocidade aproximada à 9,81 m/s. Quando o dispositivo estiver parado, ou estiver se movimentando em uma velocidade constante, os valores retornados pelo acelerômetro representarão apenas a aceleração da gravidade.

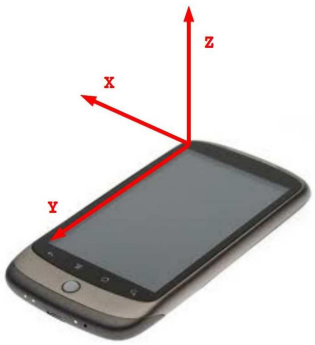

Figura 10. Sistema de coordenadas do acelerômetro na plataforma Android.

Cuidados devem ser tomados ao observar a aceleração em um determinado eixo do dispositivo. O dispositivo poderá mudar a direção em que está percorrendo no espaço, mas o sensor informa uma aceleração em um mesmo eixo. Para identificar mudanças de direção com o dispositivo e traçar um caminho percorrido, pode-se utilizar um ponto de referência externo ao

dispositivo, como por exemplo, o norte magnético.

Uma mudança na orientação do dispositivo, da visualização em forma de retrato para a forma de paisagem, tem como consequência a alteração do sistema de coordenadas do sensor de acelerômetro, isto significa que, o ponto de origem do sistema de coordenadas do sensor de acelerômetro sempre será o canto superior esquerdo para a sua respectiva orientação de visualização.

Para identificar a mudança na orientação do dispositivo, a aplicação deve utilizar o sensor de rotação. Um sensor simples contendo o método *Display.getRotation()*. O valor de retorno será um de *Surface.ROTATION_0*, *Surface.ROTATION_90*, *Surface.ROTATION_180*, ou *Surface.ROTATION_270*, tendo o primeiro valor como sendo a orientação padrão de cada dispositivo. Mas esta forma apenas identifica uma mudança completa na orientação do dispositivo, portanto para identificar um grau de mudança de orientação de forma mais exata deve-se utilizar ou sensor, como por exemplo, um sensor de giro ou torção.

A listagem 7 está demonstrando a forma de utilização da coleta do valores de retorno de um evento do sensor acelerômetro.

Listagem 7. Exemplo de classe para coleta dos valores do acelerômetro.

```
public class MainActivity extends Activity implements SensorEventListener
{
  private SensorManager mgr;
  private Sensor accel;
  private float aceleracaoX;
  private float aceleracaoY;
  private float aceleracaoZ;

  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mgr = (SensorManager) this.getSystemService(SENSOR_SERVICE);
        accel = mgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  }
  public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() != Sensor.TYPE_ACCELEROMETER)
      return;
    aceleracaoX = event.values[0];
    aceleracaoY = event.values[1];
    aceleracaoZ = event.values[2];
 }
}
```

2.2.2.2 Utilizando o giroscópio

O giroscópio é um sensor que mede a torção do dispositivo nos 3 eixos: x, y e z. Quando o dispositivo é girado em torno de um ou mais eixos, o sensor retorna valores diferentes de zero (KOMATINENI; MACLEAN, 2012).

A utilização do giroscópio normalmente é combinada com outros sensores, como por exemplo, o acelerômetro, para indicar um movimento percorrido. Em outros momentos o giroscópio é utilizado sozinho para indicar comandos de entrada em um jogo no dispositivo móvel.

Os valores retornados pelo sensor giroscópio são unidades em radianos por segundo, e os valores representam a taxa de rotação em torno de cada um dos eixos. Uma maneira de trabalhar com esses valores é integrá-los ao longo do tempo para calcular uma mudança de ângulo. Este é um cálculo semelhante à integração da velocidade linear ao longo do tempo para calcular a distância.

Para a utilização deste sensor utiliza-se a indicação de sensor: *Sensor.TYPE_GYROSCOPE*.

2.2.2.3 Utilizando o sensor magnético

Um sensor que identifica mudanças na orientação do norte magnético, uma bússola digital. Seu uso é através da classe *GeomagneticField* a qual recebe como parâmetro a latitude, longitude, altitude em metros e o tempo em milisegundos em que o sensor deverá funcionar até retornar o resultado.

Um objeto da classe *GeomagneticField* obtem o ângulo de inclinação através do método *getInclination()*. Antes de considerar como aceito o resultado deste método, deve-se conferir a intensidade do campo magnético no respectivo local com o método *getFieldStrength()*.

Lembrando que a diferença entre o norte magnético e o norte geográfico é gradual ao logo de um percurso no globo terrestre. Para identificar o ângulo entre o norte magnético e o norte geográfico, utiliza-se um novo sensor, o sensor de inclinação. Para exemplificar o valor de retorno deste sensor, se um dispositivo estiver localizado exatamente entre o norte magnético e o norte geográfico, o sensor retornará o valor de 180 graus. Para uso do sensor de inclinação identificando o ângulo de diferença entre o polo norte magnético e o norte geográfico, utiliza-se o método *getDeclination()*. O valor de retorno deste método será positivo caso o norte magnético estiver a leste do norte geográfico.

2.2.3 Comunicação com arquivos OWL no Android

Para trabalhar com uma ontologia e seus respectivos arquivos RDF ou OWL precisa-se de uma biblioteca ou *framework* que permita esta comunicação entre tecnologias diferentes. Conforme descrito no capítulo anterior sobre ontologias, o *framework* Jena aproxima a linguagem Java da estrutura de uma ontologia. Na plataforma Android necessita-se de um famework equivalente ao Jena e que possa ser reconhecido pela máquina virtual do Android, *Darvik Virtual Machine*. O projeto Androjena[1] adapta o *framework* Jena para ser trabalhado na plataforma Android.

Tendo o Androjena integrado a um projeto no Android, todas as classes e métodos do

[1] Disponível em: http://code.google.com/p/androjena/, acesso em dez 2012.

Jena estão disponíveis para serem utilizados por esta nova aplicação. O Androjena permite trabalhar com a linguagem SPARQL para manipular ontologias em formato RDF ou OWL (AQUIN; NIKOLOV; MOTTA, 2011).

Para o funcionamento do Androjena em um projeto no Android, suas bibliotecas devem ser adicionadas ao projeto e marcadas para serem exportadas para dentro do arquivo executável Android, arquivo com extensão ".apk". Para realizar esta exportação das bibliotecas para dentro do arquivo executável, utilizando para desenvolvimento do projeto a IDE Eclipse, seguem-se os seguintes passos:

1. propriedades do projeto;
2. Java Build Path
3. na aba "Order and Export", marcar as bibliotecas adicionadas ao projeto e que deverão ser exportadas para o arquivo executável.

2.3 Agentes de software

A definição para agentes de software inteligentes consiste em uma parte dos agentes inteligentes segundo (BRENNER; ZARNEKOW; WITTIG, 1998). A figura 11 demonstra as categorias de agentes inteligentes e onde se encontra os agentes inteligentes de software.

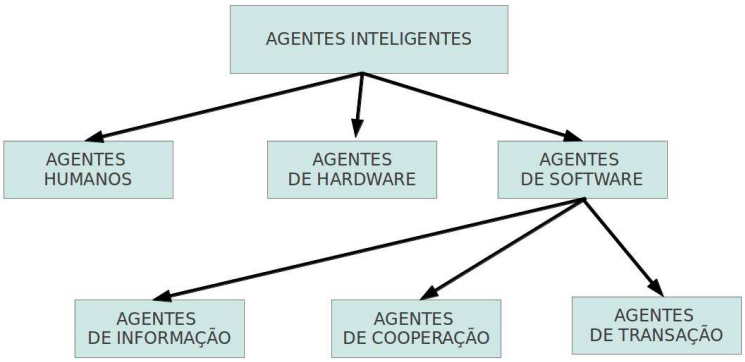

Figura 11. Categorias dos agentes inteligentes (BRENNER; ZARNEKOW; WITTIG, 1998).

Os agentes sempre precisam de uma certa quantidade de inteligência para responder de forma eficiente as questões. Pode ser considerado um agente de software não inteligente qualquer programa de software tradicional. Um agente de software inteligente tem a capacidade de executar sua tarefa de forma autônoma e requer intervenção do usuário apenas em decisões muito importantes ou não previstas, nas demais ocasiões, o agente terá a possibilidade de comunicar-se diretamente com o ambiente, conforme indicado na figura 12. Também é conhecido como paradigma orientado a agentes os sistemas de *software* que incorporam inteligência, permitindo interatividade entre componentes de software conhecidos como agentes.

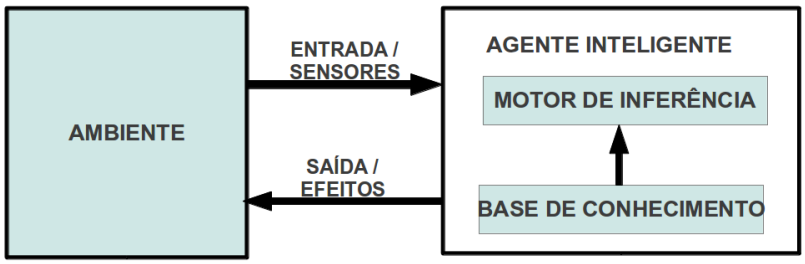

Figura 12. Arquitetura de um agente (TECUCI, 1998).

Os agentes de software apresentam as seguintes características:

- Interação: Um agente comunica-se com o ambiente e outros agentes;
- Adaptação: Um agente adapta e modifica seu estado de acordo com as mensagens recebidas do ambiente;
- Autonomia: uma das principais diferenças entre agentes e um programa de software tradicional é a a capacidade de um agente executar seu objetivo de forma autônoma, isto é, sem a interação ou comandos originados do ambiente e possui seu próprio controle podendo aceitar ou recusar uma mensagem;
- Aprendizagem: Um agente pode aprender, baseado em experiências anteriores enquanto reage e interage com o ambiente;
- Mobilidade: a mobilidade descreve a habilidade do agente de software caminhar dentro de uma rede de comunicação. Este tipo de agente consegue realizar uma tarefa em diversos computadores, executando em um computador após o outro. Ao contrário deste, os agentes estáticos permanecem apenas em um computador, mas podem trocar informações na rede comunicação com outros agentes.
- Comunicação e cooperação: estas características colocam o agente em contato com o ambiente envolvido e também com demais agentes de software para alcançar seus objetivos e as metas do sistema.
- Reatividade: caracteriza a capacidade de reagir a mudanças ocorridas no ambiente;
- Proatividade: caracteriza a capacidade de tomar iniciativas que visem alcançar seus objetivos.

2.3.1 Sistemas Multi-Agentes

Sistemas multi-agentes são sistemas constituídos de múltiplos agentes que interagem ou trabalham em conjunto, ou até mesmo trabalham de forma independentes, para realizar um determinado conjunto de tarefas ou objetivos. Esses objetivos podem ser comuns a todos os agentes ou não. Os agentes dentro de um sistema multi-agente podem ser heterogêneos ou homogêneos, colaborativos ou competitivos, ou seja, a definição dos tipos de agentes depende da finalidade da aplicação que o sistema multi-agente está inserido.

Os sistemas multi-agentes com agentes reativos são constituídos por um grande número de agentes. Estes são bastante simples, não possuem inteligência ou representação de seu ambiente e interagem utilizando um comportamento de ação e reação. A inteligência surge conforme ocorrem as interações entre os agentes e o ambiente. Ou seja, os agentes não são inteligentes individualmente, mas o comportamento é global.

Já os sistemas multi-agentes constituídos por agentes cognitivos são geralmente compostos por uma quantidade bem menor de agentes se comparado aos sistemas multi- agentes reativos. Estes, conforme a definição de agentes cognitivos, são inteligentes e contêm uma representação parcial de seu ambiente e dos outros agentes. Podem, portanto, comunicar- se entre si, negociar uma informação ou um serviço e planejar uma ação futura. Em geral os agentes cognitivos são dotados de conhecimentos, competências, intenções e crenças que lhes permite coordenar suas ações visando à resolução de um problema ou a execução de um objetivo muitas vezes com características de estarem fisicamente distribuídos ou até mesmo, entre sistemas distintos, fornecendo maior robustez e eficiência.

O desenvolvimento de sistemas multi-agentes através da utilização de uma metodologia agrega produtividade e qualidade na organização dos componentes do sistema que compõem todo o conhecimento necessário para a continuidade ou manutenção do sistema. Isso se faz necessário devido a complexidade e desafios presente no projeto e na implementação de um sistema multi-agente, por exemplo: definir como deve ser realizada a comunicação entre os agentes e qual o protocolo a ser utilizado, definir como vão ocorrer as interações entre os agentes e que linguagem eles devem usar para interagirem entre si, definir como garantir essa coordenação entre os agentes para que haja uma coerência na solução do problema ou objetivo que estão tentando resolver.

2.3.2 Metodologias de engenharia de software orientada a agentes

A seguir estão listadas algumas metodologias de engenharia de software voltadas para o desenvolvimento de agentes ou multi-agentes de software. As metodologias escolhidas apresentam vasto uso na engenharia de software orientada a agentes e existência de ferramentas para auxiliar a construção dos resultados sugeridos em cada etapa das metodologias além de conterem tecnologias novas que tendem a estarem cada vez mais presentes no universo da engenharia de software.

MaSE (*Multiagent Systems Engineering*), é uma das metodologias existentes na engenharia de software para sistemas multi-agente, ela permite projetar e modelar a aplicação desde seus objetivos até o diagrama de objetos que serão criados. A utilização de uma ferramenta para o desenvolvimento da modelagem é de extrema importância pois fornece padronização e preserva um nível semântico uniforme. A metodologia MaSE contém um processo de desenvolvimento dividido em duas fases principais: fase de análise e fase de projeto, conforme representado na figura 13.

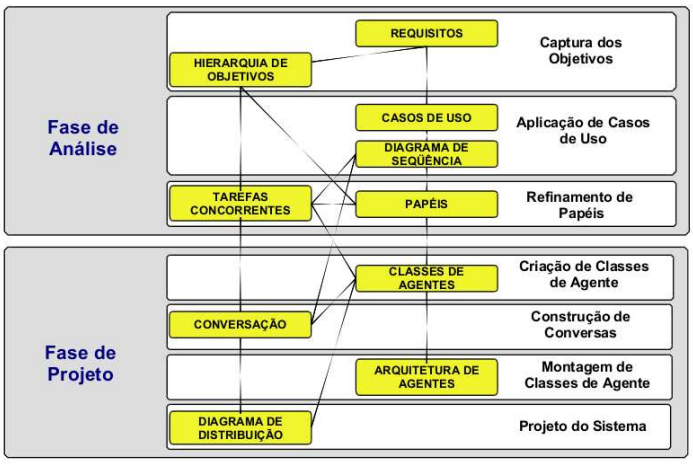

Figura 13. Fases do processo MaSE (DELOACH, 2001).

- Fase de Análise: Captura os objetivos do sistema, aplica os casos de uso e executa o refinamento dos papéis.
- Fase de Projeto: Cria as classes de agentes, construção da conversa entre os agentes, monta as classes de agente e projeta o sistema.

A metodologia MaSE foi desenvolvida para ser aplicada de forma interativa, passando entre as etapas em diversos momentos, resultando em um modelo completo e consistente (DELOACH, 2001).

A identificação dos objetivos do sistema multi-agente visa conhecer as especificações do sistema e transformá-las em uma estrutura de conjunto de objetivos.

Os casos de uso têm por finalidade identificar os diversos cenários descritos na fase de levantamento de objetivos do sistema, além de representar os comportamentos desejáveis ao sistema. Os casos de uso resultaram em diagramas de sequência, onde se constrói a sequência de eventos projetos para ocorrerem entre os diferentes papéis definidos a partir da identificação de objetivos.

No diagrama de papeis, definem-se os papéis (retângulos), tarefas (elipses) e protocolos de comunicação (setas) que o sistema multi-agente deverá conter.

O diagrama de tarefas é definido a partir do diagrama de papéis, considerando as tarefas assumidas pelos diferentes papéis a serem desempenhados pelos agentes.

No diagrama as classes de agentes são identificadas a partir dos papéis, representadas em caixas, e os diálogos, em linhas, conectando as classes.

O diagrama do sistema procura representar o sistema multi-agente de sua forma mais abstrata possível no nível de blocos que podem ser considerados subsistemas.

Cada um dos diagramas propostos pela metodologia MaSE podem ser construídos utilizando a ferramenta *agentTool* (DELOACH, 2001).

Prometheus: é uma metodologia para desenvolvimento de sistemas multi-agentes que abrange desde a modelagem até a implementação. Esta metodologia é composta por três fases, onde os artefatos produzidos são utilizados tanto na geração do esqueleto do código, como também para depuração e teste.

A primeira fase, correspondente à especificação do sistema, compreende duas atividades: determinar o ambiente do sistema e determinar os objetivos e funcionalidades do sistema. O ambiente do sistema é definido em termos de percepções (informações provenientes do ambiente) e ações. Além disso, são definidos dados externos. As funcionalidades do sistema são definidas através da identificação de objetivos, da definição das funcionalidades necessárias para se alcançar esses objetivos e dos cenários de casos de uso.

A segunda fase, projeto arquitetural, utiliza as saídas da fase anterior para determinar quais agentes existirão no sistema e como os mesmos irão interagir. Esta fase envolve três atividades: definição dos tipos de agentes, definição da estrutura do sistema e definição das interações entre os agentes. A última fase desta metodologia (projeto detalhado) é responsável por definir capacidades dos agentes, eventos internos, planos e uma estrutura de dados detalhada de cada tipo de agente identificado na fase anterior.

Para esta metodologia existe a ferramenta Prometheus Design Tool (PDT) (PADGHAM; THANGARAJAH; PARESH, 2012). Ela permite que o usuário entre e edite o projeto utilizando

os seguintes conceitos: verificar projeto para um conjunto de possíveis inconsistências, gerar automaticamente um conjunto de diagramas de acordo com a metodologia e gerar automaticamente a descrição do projeto, o que inclui descritores para cada entidade, um dicionário para o projeto e os diagramas gerados.

PASSI (Process for Agent Societies Specification and Implementation) é uma metodologia para desenvolvimento de sistemas multi-agentes que integra a definição da modelagem e filosofia de sistema multi-agentes como também de orientação a objetos, utilizando UML. Esta metodologia é composta de cinco modelos que endereçam diferentes visões e doze passos durante seu processo de desenvolvimento.

Os modelos existentes nesta metodologia estão dispostos da seguinte forma:

- Requisitos do sistema: O modelo relacionado aos requisitos do sistema compreende quatro passos: descrição do domínio, identificação dos agentes, identificação dos papéis e especificação das tarefas;
- Sociedade dos agentes: Já o modelo de sociedade dos agentes é responsável por modelar as interações e dependências entre os agentes presentes na solução. A elaboração deste modelo envolve os três passos seguintes: descrição das ontologias, descrição dos papéis e descrição dos protocolos;
- Implementação dos agentes: tem por objetivo modelar a arquitetura utilizada como solução em termos de classes e métodos. Este modelo compreende os seguintes passos: definição da estrutura do agente e descrição do comportamento do agente;
- Código: o modelo de código visa modelar a solução a nível de código;
- Distribuição: que descreve a distribuição das partes do sistema.

Como comparativo entre os três métodos orientados a agentes, a tabela 4 descreve uma analise das características mais relevantes identificadas em cada um dos métodos aqui descritos.

Tabela 4. Características de métodos orientados a agentes estudados.

Característica	**MaSE**	**Prometheus**	**PASSI**
Modelagem de sistemas multi-agentes	X	X	X
Ferramentas com geração automática da implementação de agentes		X	
Utilização de ontologias nas suas inferências.			X
Permite projetar um sistema multi-agente sem ter conhecimento claro de todos agentes necessários e a iteração entre eles.	X		

2.3.3 JADE

JADE é uma plataforma de software que provê uma camada *middleware* de funcionalidades básicas, as quais são independentes de uma aplicação específica e podem simplificar a distribuição de soluções orientadas a agentes. Basicamente, o *framework* provê aos programadores o seguinte núcleo de funcionalidades (BELLIFEMINE; CAIER GREENWOOD, 2007):

• sistemas totalmente distribuídos habitados por agentes, onde cada agente roda em uma

thread independente, potencialmente em máquinas remotas e diferentes;

• compatibilidade total com as especificações FIPA, participando de todos os eventos de interoperabilidade;

• transporte eficiente de mensagens assíncronas via uma interface de programação, onde a plataforma garante a escolha do melhor meio de comunicação;

• implementação de páginas brancas e páginas amarelas, onde sistemas podem ser implementados para representar domínios e subdomínios de um grafo de diretórios;

• um conjunto de ferramentas de suporte gráficas que facilitam o trabalho de depuração e monitoramento realizado pelos programadores;

• suporte a ontologias e linguagens de conteúdo, com verificação e codificação automática baseadas em Extensible Markup Language (XML) e RDF;

• bibliotecas de protocolos de interação que estabelecem modelos de comunicação orientados a realização de um ou mais objetivos.

• integração com diversas tecnologias orientadas a *web*, tais como: servlets, applets e *web* services; e suporte para plataformas móveis e ambientes com redes sem fio.

• interface de processos para controle de uma plataforma e seus componentes distribuídos fora da aplicação.

• *kernel* desenhado para permitir que programadores extendam o seu comportamento de acordo com suas necessidades específicas.

2.4 *WebServices*

A tecnologia *WebServices* oferece uma solução abrangente para a representação e invocação de serviços em uma ampla variedade de ambientes.

Os *WebServices* são formados por dois esquemas XML (*Extensible Markup Language*):

- WSDL (*Web Service Description Language*)
- SOAP (*Simple Object Access Protocol*)

Baseado em uma estrutura XML, SOAP garantiu a extensibilidade, a transparência e a interoperabilidade de sistemas heterogêneos.

Em cada linguagem de programação existe uma forma ou biblioteca para implementar o protocolo SOAP. Para dispositivos móveis baseados na linguagem J2ME existe a biblioteca KSOAP2 (HAUSTEIN; SEIGEL, 2012).

Duas tarefas básicas devem ser realizadas ao iniciarmos a construção da aplicação:

1. Importar a biblioteca KSOAP2 ao projeto

Após a importação desta biblioteca, o conjunto de classes implementadas por ela, passarão a fazer parte do conjunto de classes do *framework* Android do respectivo projeto.

2. Toda aplicação Android que fizer acesso a algum componente físico do dispositivo móvel, deverá ter explicitamente uma permissão para acessá-lo. Existem diversos tipos de permissões no aplicativo Android, a permissão necessária para esta biblioteca é “INTERNET”, pois trata-se de uma transmissão de informações transmitidas através do protocolo HTTP.

A estrutura de um pacote SOAP é formada pelo pacote principal chamado Envelope e dentro dele os pacotes *Header* e *Body*, conforme mostrado na figura 14.

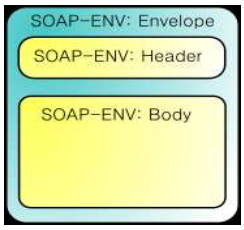

Figura 14. Estrutura do SOAP.

Os comandos da biblioteca KSOAP2 seguem a seguinte estrutura:

1. Criação do objeto SOAP, contendo o endereço do servidor *WebService* e o nome da função solicitada:

```
SoapObject soap = new SoapObject(
      "http://10.3.0.1/ServidorSoap/TSoapServer.class.php",
          "chamaMetodo");
```

2. Adicionar as informações que serão enviadas para a função:

```
//primeiro parâmetro
soap.addProperty("key","poklamx");
//segundo parâmetro
soap.addProperty("method","validaCpf");
//terceiro parâmetro
soap.addProperty("cpf","00100200344");
```

3. Cria um envelope para armazenar o conteúdo SOAP:

```
SoapSerializationEnvelope envelope =
                  new SoapSerializationEnvelope( SoapEnvelope.VER11);
envelope.setOutputSoapObject(soap);
```

4. Cria o transportador para o envelope SOAP:

```
String url =  "http://10.3.0.1/ServidorSoap/ TSoapServer.class.php";
HttpTransport httpTransport =
        new HttpTransport(url);
try {
        httpTransport.call("", envelope);
        Object msg=envelope.getResponse();
        Log.i("Retorno validação:", msg);
} catch (Exception e) {
```

```
            e.printStackTrace();
    }
```

3 TRABALHOS RELACIONADOS

São apresentados aqui, alguns trabalhos de pesquisa que tiveram em seu foco de trabalho a coleta de informações de contexto.

3.1 Coleta de dados sensoriais usando dispositivos móveis - CADAMULE

O trabalho (JAYARAMAN; ZASLAVSKY; DELSING, 2008) apresenta uma coleta de informações referentes ao contexto em que estão inseridos os dispositivos móveis. A abordagem construída chama-se CADAMULE e possibilita a descoberta dinâmica de atributos de contexto. Como resultado é apresentado os custos para a montagem desta arquitetura.

Para a identificação do contexto foi sugerido os seguintes atributos: intensidade do sinal com sensores fixos, capacidade disponível de bateria, atenuação do sinal das antenas GSM (*Global System for Mobile Communications*), localização geográfica (GPS), posição na linha do tempo (data e hora).

A figura 15 mostra a arquitetura para a coleta de informações de contexto elaborada por (JAYARAMAN; ZASLAVSKY; DELSING, 2008), contendo os dispositivos móveis, os sensores para ambientes internos, o qual está preparado para identificar a intensidade do sinal de antenas *Wi-Fi* fixadas em pontos específicos, e também o sinal de antenas GSM.

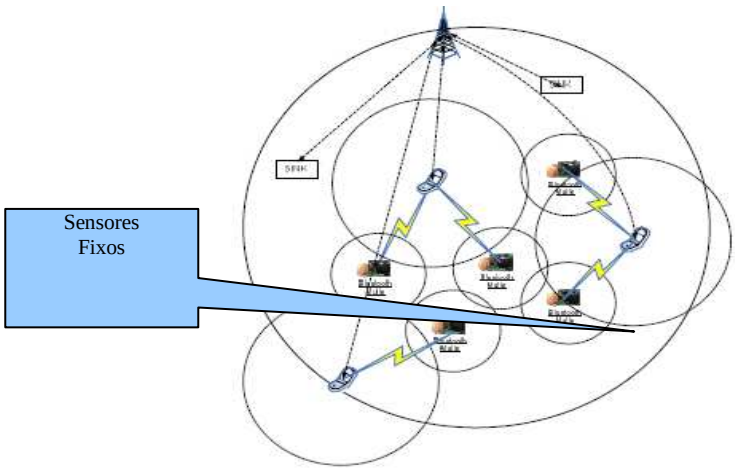

Figura 15. Arquitetura da coleta de informações de contexto (JAYARAMAN; ZASLAVSKY; DELSING, 2008).

Neste trabalho utilizou-se o contexto individual dos dispositivos móveis para organizar o contexto das aplicações utilizadas, por exemplo, informações que indicam os locais e a frequência de uso de um aplicativo de *software* instalado nos dispositivos móveis.

Durante a utilização dos dispositivos móveis em um ambiente, o software CADAMULE poderá transmitir as informações de contexto de um dispositivo para outro através de conexão Bluetooth[1] que apresentam baixo consumo de energia e possibilita a comunicação à dispositivos próximos, transformando assim um simples dispositivo móvel em um transmissor das informações de contexto percorrendo por vários dispositivos móveis até que cheguem em um

[1] http://www.bluetooth.com

servidor central.

3.2 Mobile Apps: It's Time to Move Up to CondOS

Este artigo apresenta um novo sistema operacional para dispositivos móveis, o Context Dataflow Operating System (CondOS), desenvolvido pela empresa Microsoft. Este sistema trabalha gerenciando e armazenando o contexto do usuário, com objetivo de acelerar a inicialização de seus respectivos aplicativos e otimizar o gerenciamento do recursos controlados pelo sistema operacional (CHU et al, 2012). A figura 16 indica alguns destes serviços beneficiados pelas informações de contexto.

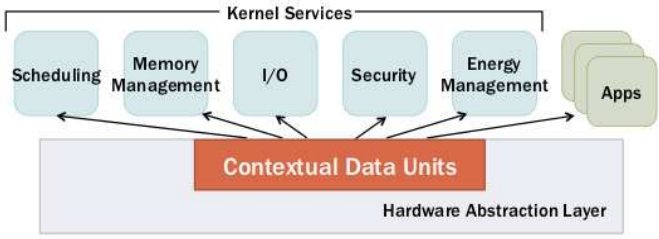

Figura 16. Serviços beneficiados pelas informações de contexto.

Os dados de contexto são armazenados no dispositivo em um local chamado de *Contextual Data Units* (CDUs) e são de uso exclusivo do sistema operacional, contendo para isso uma camada de segurança com objetivo de não disponibilizar informações confidenciais outros aplicativos instalados no dispositivo móvel.

Para o CondOS, o contexto é composto por:

- Informações gerenciadas pelo sistema operacional, como por exemplo, uso de energia e memória, comunicação e poder de processamento;
- Informações de uso dos aplicativos de usuário;
- Informações de localização do dispositivo móvel.

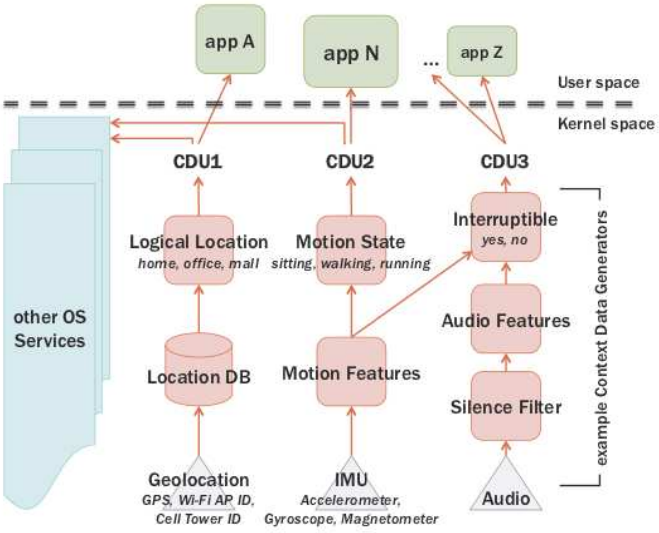

Figura 17. Algumas informações de contexto gerenciadas pelo CondOS.

A figura 17 mostra algumas informações de contexto e seu fluxo de dados gerenciadas pelo CondOS. Nota-se que o sistema de localização é baseado no sinal GPS, identificador da torre de celular e identificador do roteador Wi-Fi. Além da localização do dispositivo, é coletado através das informações dos sensores acelerômetro, giroscópio e bússola, a característica do movimento, seja ela sentar, andar ou correr, o qual este dispositivo está realizando enquanto utiliza uma determinada aplicação.

As informações de contexto armazenadas nas CDUs são utilizadas pelo sistema operacional para inicializar, em segundo plano, uma aplicação que geralmente é utilizada em um determinado local. A aplicação é inicializada no momento em que o dispositivo se aproxima do local de utilização, assim quando chega a seu destino, o usuário terá acesso ao seu aplicativo de forma mais rápida. O próprio sistema operacional utiliza-se das informações de contexto para o gerenciamento otimizado do uso do *hardware*, como por exemplo, uso da placa de rede somente quando um aplicativo requisita informações da rede, ou, desligar a tela do dispositivo baseado no movimento que está realizando, ou, escalonar e gerenciar a frequência do processador baseado nos dados históricos de um determinado aplicativo.

3.3 Indoor/Outdoor Management System Compliant with Google Maps and Android OS

Este artigo apresenta um sistema de gestão para localização em ambiente interno e externo completamente compatível com dois sistemas, Google Maps e Android. O sistema é integrado com base em módulos ZigBee[1], GPS (*Global Positioning System)*) e GPRS (*Global Packet Radio Service)* (GUENDA, 2011).

A figura 18 apresenta a arquitetura deste sistema composto por quatro partes:

- Sistema de localização ZigBee: sistema utilizado para localização interna baseado em valores RSSI (*Received Signal Strength Indication*), isto é, a intensidade do sinal por rádio frequência de outros dispositivos móveis próximos. A triangulação entre dispositivos que estão em ambientes externos, recebendo sinal GPS, indicará uma posição em valores de latitude, longitude aproximada. Esta posição aproximada, também servirá de referência para outros dispositivos presentes no ambiente interno para calcular sua respectiva posição. Observa-se na arquitetura que se utiliza da funcionalidade do hardware ZigBee para retransmitir a localização de um dispositivo móvel nas suas proximidades para os demais ZigBee até chegar no servidor central;
- Sistema de localização GPS: sistema o qual os dispositivos móveis enviam ao servidor sua posição identificada através de um módulo GPS, ou quando inexistente, sobre um sinal GPRS;
- Servidor de dados: responsável por armazenar todas as informações de posições e envia-las para a interface do usuário do dispositivo. A informação recebida dos dois sistemas de localização, ZigBee e GPS, está estruturada em um arquivo em formato XML, indicando o valor para a latitude, longitude, data e hora para cada dispositivo móvel. Estas informações são armazenadas em um banco de dados relacional para posterior utilização;
- Serviços e interface com usuário: fornece um aplicativo amigável de gerenciamento de usuário. Este serviço comunica-se como o servidor através de *webservice*, portanto necessita de comunicação com a rede *internet* para o seu funcionamento. Todo o percurso de um determinado dispositivo é integrado ao Google Maps e exibido em um sistema

[1] http://www.zigbee.org

nativo para a plataforma Android, utilizado pelo respectivo dispositivo móvel, e um sistema *web* para o gerenciamento através de qualquer computador conectado a rede *internet*.

A coleta de informações no ambiente interno através da tecnologia ZigBee foi criada com objetivo para comunicação *Wi-Fi*, mas com alcance menor (aproximadamente até 10 metros) e menor consumo de energia, portando é ainda pouco utilizada como meio de localização e dependerá de existir, de forma integrada, em muitos dispositivos móveis, para assim ser utilizada por usuários comuns. Segundo (GUENDA, 2011), este sistema está sendo testado com sucesso em aplicações como limitador de percursos segundo permissões gerenciadas por um usuário no sistema.

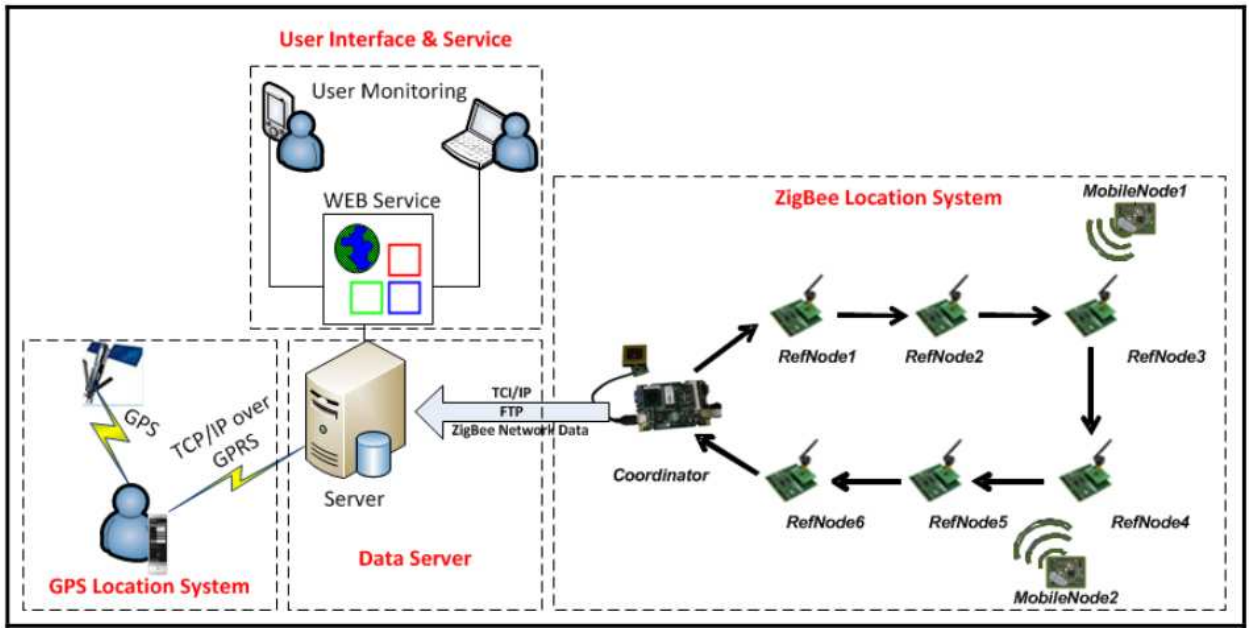

Figura 18. Arquitetura do sistema de localização interno e externo (GUENDA, 2011).

3.4 From GPS tracks to context

Para (MOREIRA; SANTOS, 2005) os aplicativos baseados em localização usam a localização de usuários para adaptar o seu comportamento e para selecionar as informações relevantes para os usuários em uma situação particular. Esta informação sobre a localização é obtida através de um conjunto de sensores de localização, ou de serviços de rede baseados em localização, e muitas vezes é utilizada diretamente, sem qualquer processamento adicional, como um parâmetro em um processo de seleção. Neste trabalho, é proposto um método para inferir informações de contexto de alto nível (contexto lógico) de uma série de registros obtidos a partir da posição de um receptor GPS. Este método, baseado em um algoritmo de agrupamento espacial, automaticamente calcula a localização dos lugares onde os usuários vivem e trabalham, isto é, o algoritmo apresentado realiza um agrupamento dos pontos de localização para classifica-los como sendo de um contexto lógico trabalho ou casa.

A todo instante é armazenada a localização de todo percurso em que foi utilizado o dispositivo móvel.

O algoritmo apresentado neste trabalho utiliza-se de um cálculo de confiança para o ponto onde o usuário se encontra:

$$cf = \frac{Tempo\ de\ permanência\ neste\ ponto}{\sum Tempos\ de\ permanência\ neste\ ponto}$$

Equação 1. Cálculo de confiança da recomendação (MOREIRA; SANTOS, 2005).

Um modelo de probabilidade de estar em casa em um determinado horário (*dh*) é definido abaixo, sendo "*PeakHour*" o provável horário em que a respectiva pessoa encontra-se em casa, e "A" mais um parâmetro que indica a amplitude que pode variar o provável horário de estar em casa.

$$P_{bah}(dh) = \frac{1}{2}\left(1 + A_1 \times Cos\left(\frac{(dh - PeakHour)\times 2\pi}{24}\right)\right)$$

Equação 2. Modelo de probabilidade (MOREIRA; SANTOS, 2005).

Para o modelo de probabilidade de estar no trabalho o "*PeakHour*" é substituido por "*PeakHour – 12*", indicando que o provável horário de estar no trabalho é o oposto do estar em casa.

Os resultados disponíveis agora podem ser utilizados para inferir informações de contexto para o usuário. Em um dado instante, dada a posição do usuário, é possível estimar se o usuário estiver em casa ou no seu local de trabalho através da multiplicação da confiabilidade pelo modelo de probabilidade:

$$context = \begin{cases} at\ HOME, with\ probability\ p = P_{bah} \times cf_{home} \\ at\ WORK, with\ probability\ p = P_{baw} \times cf_{work} \end{cases}$$

Desta forma, o local que tiver maior probabilidade é indicado como sendo o contexto provável.

Os resultados obtidos mostram que o método proposto rapidamente converge para os locais reais, e que o algoritmo proposto pode ser utilizado para estimar simultaneamente tanto a moradia como o local de trabalho e, enquanto se adapta a uma vasta gama de espaço-temporais e comportamentos humanos.

3.5 Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users

No trabalho de (ASHBROOK; STARNER, 2003) foi utilizado uma coleta de informações de contexto físico referentes à localização de computadores portáteis através de um receptor GPS. Estes são utilizados por vários usuários para então criar um modelo preditivo de movimentos do utilizador.

Primeiramente são coletados os pontos de destino e os percursos dos diversos usuários. Os pontos de destino são encontrados a partir do momento que o receptor GPS estiver se movimentando com uma velocidade considerada como andando de algum meio de transporte motorizado como automóvel, ônibus ou trem e após esta movimentação o receptor fixa-se em um

determinado local por um tempo considerável, este local então é considerado como um ponto de destino. Os diversos pontos encontrados são considerados como possíveis rotas e destino para o algoritmo preditivo. No momento quem que o receptor GPS estiver se movimentando com velocidade considerável, um ponto geográfico é coletado a cada segundo, consequentemente formando a trajetória percorrida.

Para certa localização é criada uma tabela de probabilidades para transitar do respectivo ponto até cada um dos pontos de destino possíveis. Em seguida, um modelo de cadeia de Markov é criado para cada local, com transições para cada outro local. Cada nó no modelo de Markov é um local, e uma transição entre dois nós representa a probabilidade de o usuário que viajam entre os dois locais. Se o utilizador nunca viajou entre dois locais, a probabilidade de transição é definida como zero. Este trabalho apresenta como parâmetro a possibilidade de trabalhar em ambos os cenários: com a frequência de deslocamento de um único utilizador ou de um conjunto de utilizadores.

Na figura 19 é ilustrada a probabilidade de transição entre 3 pontos no mapa.

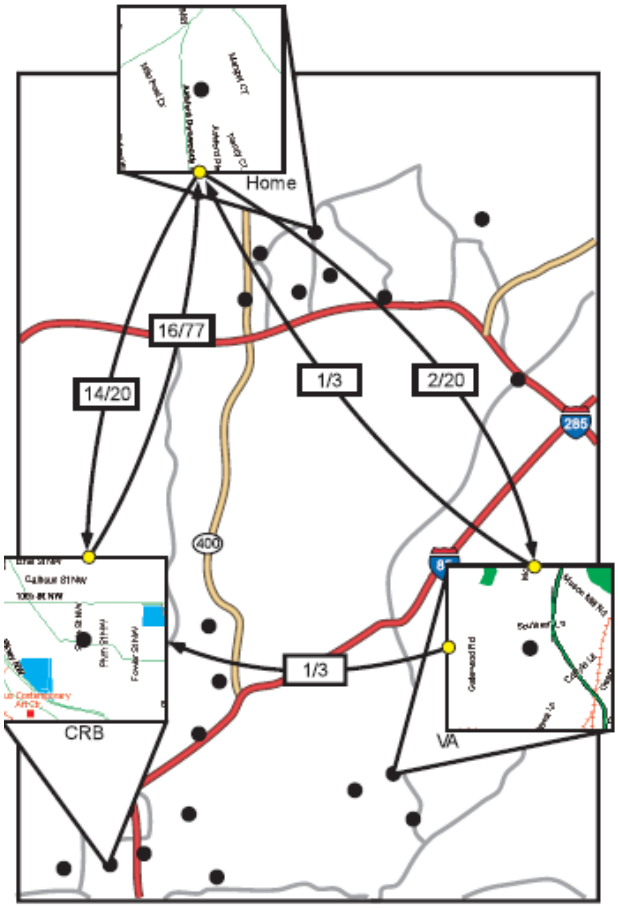

Figura 19. Possibilidade de percurso com suas probabilidades para três pontos no mapa.

Para determinar a probabilidade de sair de um ponto A e ir até o ponto B é calculada através do cálculo:

$$probabilidade\ A|B = \frac{quantidade\ de\ vezes\ que\ foi\ do\ ponto\ A\ até\ B}{quantidade\ de\ vezes\ que\ saiu\ do\ ponto\ A}$$

Equação 3. Probabilidade de sair do ponto A até ponto B (ASHBROOK; STARNER, 2003).

Para a predição de deslocamento proposta neste trabalho é considerado o local de destino que apresentar a maior probabilidade de deslocamento.

3.6 Conclusão do capítulo

As principais características dos trabalhos relacionados estão indicados na tabela 5 de forma a compara-las com as características do trabalho proposto desta dissertação.

Tabela 5. Comparativo dos trabalhos relacionados e trabalho proposto

	(ASHBROOK; STARNER, 2003)	**(MOREIRA; SANTOS, 2005)**	**(JAYARAMAN; ZASLAVSKY; DELSING, 2008)**	**(GUENDA, 2011)**	**(CHU et al, 2012)**	**Trabalho proposto**
Tipo de localização externa	GPS	GPS	GPS	GPS, GPRS	GPS	GPS
Tipo de localização interna	(não apresenta)	(não apresenta)	Antenas WIFI presente em locais específicos.	Sensor: ZigBee presentes em locais específicos.	WIFI AP id, Cell Tower id	Sensores: acelerômetro, orientação.
Informações de contexto-físico coletadas	localização geográfica	localização geográfica; posição na linha do tempo (data e hora)	intensidade do sinal com sensores fixos; capacidade disponível de bateria; atenuação do sinal das antenas GSM; localização GPS; posição na linha do tempo (data e hora).	localização geográfica; posição na linha do tempo (data e hora)	intensidade da bateria; memória; processador; aplicações em uso; localização; situação de movimento: sentar, andar ou correr.	localização geográfica; posição na linha do tempo (data e hora); situação de movimento: sentar, andar ou correr; características do hardware; memória; processador; aplicações em uso; identificação do dono do hardware.
Informações de contexto-lógico	A partir do ponto atual estimar o próximo ponto que será percorrido.	Caracterizar local como “casa” ou “trabalho”	(não apresenta)	(não apresenta)	Iniciar o funcionamento de uma lista de possíveis aplicativos a serem utilizados em um determinado local.	Lista de possíveis aplicativos a serem utilizados em um determinado instante e local.
Forma de armazena-mento dos dados	Base de dados no dispositivo móvel.	Base de dados no dispositivo móvel.	Base de dados no dispositivo móvel. E em servidor remoto.	Banco de dados XML no dispositivo móvel e em servidor remoto.	Banco de dados XML no dispositivo e em servidor remoto.	Em ontologia no dispositivo móvel e replicadas a um servidor remoto.

4 GERENCIADOR DE CONTEXTOS

Este trabalho se propõe a construir de um protótipo que verifique a funcionalidade de uma arquitetura para gerenciamento de informações que formam o contexto físico do uso de um dispositivo móvel na plataforma Android 4.0 e gerar informações de contexto lógico referentes à recomendação de uso de recursos disponíveis nos dispositivos móveis e anteriormente utilizados pelo usuário em um dado instante e local. Entre suas tarefas está a catalogação e organização das informações do uso de todos recursos de hardware e software utilizados em um dispositivo móvel (contexto lógico). O resultado será um histórico em um momento e espaço, formado por informações de localização geográfica para ambientes externos, com uso de GPS, e em ambientes internos através do uso sensores do próprio dispositivo móvel. As informações são armazenadas em uma ontologia, proporcionando a inferência destes dados por outros sistemas, independente da plataforma utilizada. A recomendação para a utilização de um determinado recurso é baseada nas localizações e instantes em que um recurso foi utilizado, para em um momento posterior, indicar que o dispositivo encontra-se próximo ao último local em que o recurso foi utilizado.

Este trabalho faz parte de um trabalho maior com objetivo de ser um conselheiro móvel pessoal, formado por este trabalho sendo um gerenciador de contextos, além de um *desktop* semântico e uma federação de contextos. Esta estrutura está representada na figura 20. Neste modelo, outros dois colegas também estão trabalhando no *desktop* semântico e na federação de contextos.

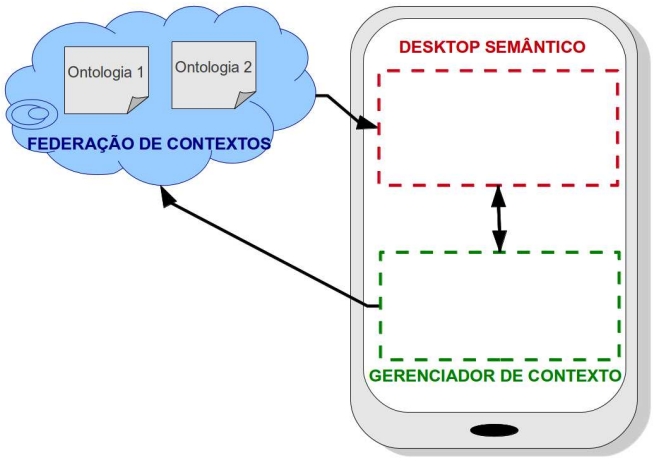

Figura 20. Visão geral do **Conselheiro de contexto móvel pessoal**.

O modelo da figura 20 contém os seguintes sistemas:

- *Desktop* Semântico: este sistema analisa informações recebidas da federação de contextos, enquanto o dispositivo estiver conectado a rede *web*, ou do gerenciador de contexto local, enquanto não tiver conectado a rede *web*, para gerenciar a atualização do *desktop* e dos recursos baseados em seu contexto lógico, isto é, tendo conhecimento sobre as ações e sobre o ambiente atual do usuário, o aplicativo terá condições de alterar o desktop do dispositivo móvel de maneira a deixar evidente o ícone de um aplicativo qualquer, como e-mail por exemplo. Também será tarefa deste sistema o monitoramento dos recursos utilizados no dispositivo móvel, avisando o gerenciador de recursos que uma nova aplicação está sendo inicializada. Como resposta recebe informações de

localização e tempo;

- Gerenciador de contexto, representando este respectivo trabalho, tendo como tarefas principais a coleta e gerenciamento do contexto de recursos do dispositivo móvel e a recomendação de uso dos recursos disponíveis no dispositivo móvel;
- Federação de contextos: armazenando diversas ontologias sobre contextos lógicos no uso de tecnologias e recursos de cada usuário cadastrado, sendo um servidor na nuvem armazenando e gerenciando ontologias.

Metodologia adotada para a construção da arquitetura seguiu alguns conceitos de UML Components (CHEESMAN; DANIELS, 2001) que é constituída pelas tarefas abaixo na sua respectiva ordem:

- Identificar as interfaces de negócio, identificadas a partir da evolução do modelo de casos de uso: que são: indicar recurso a ser utilizado em um contexto físico, identificar contexto físico, identificar recurso utilizado;
- Identificar os componentes de negócio, representando as partes que fornecerão as interfaces e as partes que farão uso destas interfaces. Em alguns momentos é possível haver mais de um componente para prover uma interface;
- Detalhamento do contrato de uso de cada interface, representando as condições necessárias para o fornecimento das interfaces e a forma de utilização da mesma. O contrato de uso prevê o que tem que ser feito independente de como será construído, isto é, o que importa para ele é o resultado final dos componentes.

Os componentes apresentados na arquitetura proposta, seguem os princípios de unificação de dados e funções, encapsulamento e identidade única, através da elaboração de objetos a noção de uma especificação de objeto com uma representação explícita de sua dependência chamada de interface (CHEESMAN; DANIELS, 2001).

A relação entre componentes pode ser caracterizada por dois tipos de contrato:

- Contrato de uso, o contrato entre um componente de objeto de interface e seus clientes;
- Contrato de realização, o contrato entre um componente específico e sua implementação.

A arquitetura deste trabalho apresenta apenas contratos de uso entre os componentes existentes.

Para o modelo apresentado na figura 20, este trabalho tem como responsabilidade, a disponibilização do gerenciador de contexto. Este trabalho é formado por agente de localização, agente de recursos, agente de armazenamento e agente de recomendação de recursos, conforme representado na figura 21.

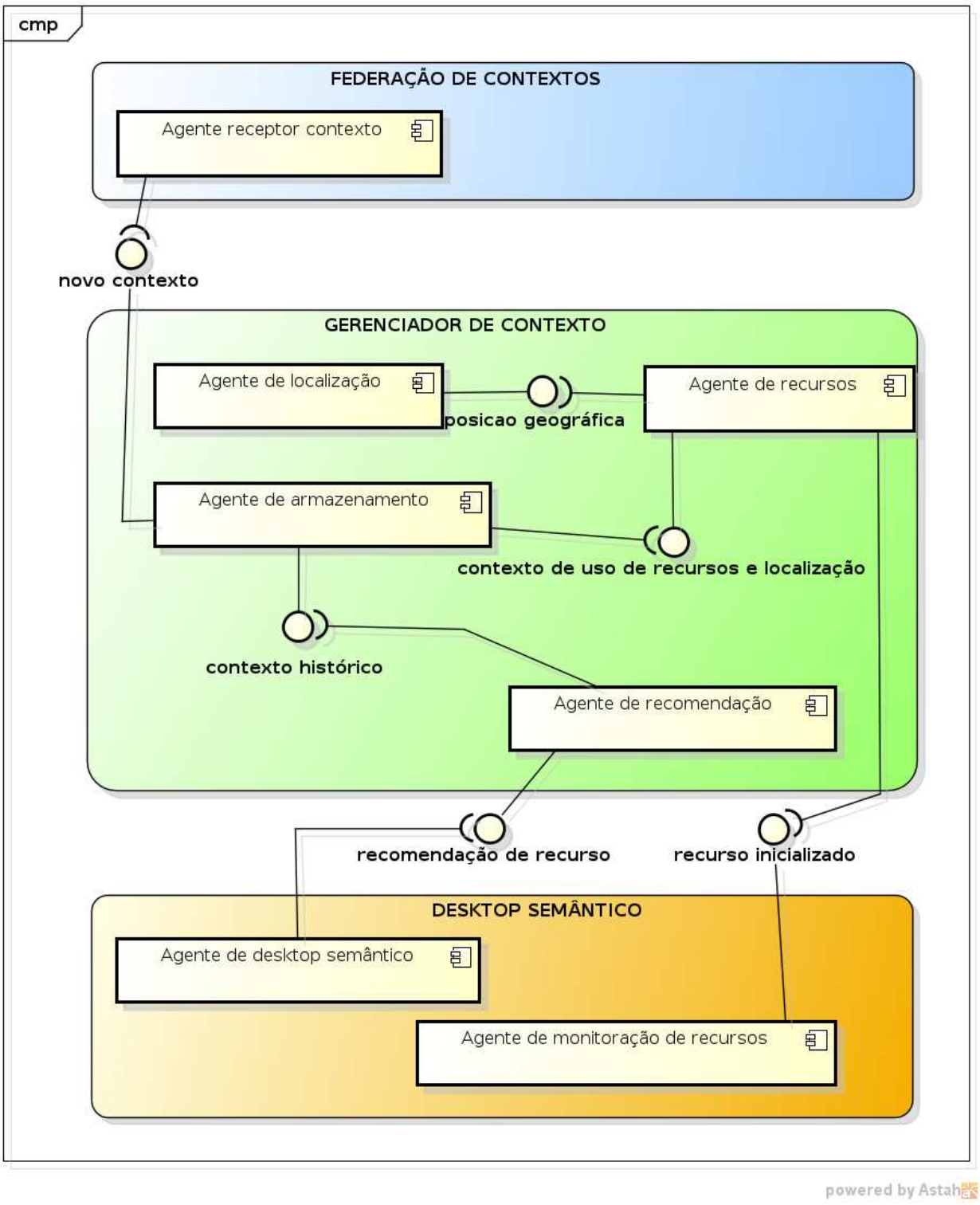

Figura 21. Arquitetura do Gerenciador de contexto.

A seguir, está sendo detalhado o funcionamento de cada um dos elementos contidos no gerenciador de contexto instalado nos dispositivos móveis Android.

Para a comunicação adotada entre os agentes dentro da plataforma Android, foi utilizado publicações de mensagens que podem ser coletadas por outros agentes que estejam aguardando o padrão de mensagem enviado. Na plataforma Android esta forma de comunicação é comumente utilizada para comunicação entre aplicações e o envio de informações de sensores, como GPS e

acelerômetro.

A função responsável pela publicação das mensagens é "*sendBrodcast*", a qual está contida na classe base das aplicações na plataforma Android: "*Context*". Esta função publica um "*Intent*", um objeto com descrição abstrata a qual caracteriza a sua intenção de ação.

Esta intenção de ação deve ser conhecida entre as aplicações que reconhecerão o conteúdo de suas mensagens. Abaixo está a demonstração de um envio de mensagem realizada pelo agente de localização:

Listagem 8. Envio de *broadcast* da localização atual pelo agente de localização.

```
Intent intent = new Intent("BROADCAST_POSICAO");
intent.putExtra("mLatitude", "30.32454");
intent.putExtra("mLongitude","-81.6584");
intent.putExtra("mAltitude","102");
intent.putExtra("recurso_nome", nomeRecurso);
sendBroadcast(intent);
```

A função "*sendBrodcast*" envia mensagens de forma assíncrona, isto é, podendo ter mais de uma mensagem enviada no mesmo instante por agentes ou aplicações distintas e que estejam em funcionamento no mesmo instante.

O recebimento destas mensagens é realizado pela classe "BroadcastReceiver", a qual deve ser estendida para ser construído os detalhes do recebimento de mensagem na classe "onReceive(Context contexto, Intent intent)". Uma mesma classe pode estar preparada para ouvir mensagens enviadas por mais de um *broadcast*, seja ele do respectivo aplicativo, ou seja, de outro aplicativo em execução dentro do mesmo aplicativo móvel. A identificação de cada um deles é possível pela função "intent.getAction". Abaixo está a demonstração desta classe identificando um *intent* chamado "BROADCAST_POSICAO", enviado no exemplo acima.

Listagem 9. Exemplo de identificação de recebimento de um *broadcast*.

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    public void onReceive(Context contexto, Intent intent) {
        if (intent.getAction() ==  "BROADCAST_POSICAO") {
```

A classe "BroadcastReceiver" é chamada automaticamente pela aplicação assim que identificados os *broadcasts* definidos dentro do arquivo *AndroidManifest.xml* do projeto construído:

Listagem 10. Configuração do AndroidManifest.xml para receber *broadcast* sobre recursos inicializados.

```
<receiver android:name=".MyBroadcastReceiver">
 <intent-filter>
  <action android:name="com.unisinos.AppDispatcher">
      <categoryandroid:name="android.intent.category.DEFAULT"> </category>
  </action>
 </intent-filter>
```

</receiver>

Cada *broadcast* recebido cria uma instância da classe MyBroadcastReceiver definida dentro do arquivo *AndroidManifest.xml*. Isso significa que, se duas mensagens são enviadas no mesmo instante, dentro de um mesmo dispositivo móvel, serão criadas duas instâncias da classe MyBroadcastReceiver.

A classe MyBroadcastReceiver, criada neste protótipo, tem a responsabilidade de gerenciar a comunicação entre todos os agentes, tendo conhecimento da sequência e os agentes receptores de cada tipo de *broadcast* gerado na arquitetura.

Para o gerenciamento dos recursos, o trabalho inicia no agente de localização com a identificação da posição atual. A partir deste momento, o agente de recursos aguarda o *broadcast* BROADCAST_RECOMENDACAO_RECURSO, indicando um novo recurso em utilização no dispositivo móvel, em seguida busca a posição atual no agende de localização e demais informações de contexto do dispositivo móvel, onde são publicadas no dispositivo através do *broadcast* BROADCAST_NOVO_CONTEXTO_RECURSO e também enviadas diretamente para o agente de armazenamento, o qual tem a tarefa de armazenar as informações de contexto no formato da ontologia de contextos. A figura 22 contém um diagrama desta sequência de atividades destes agentes.

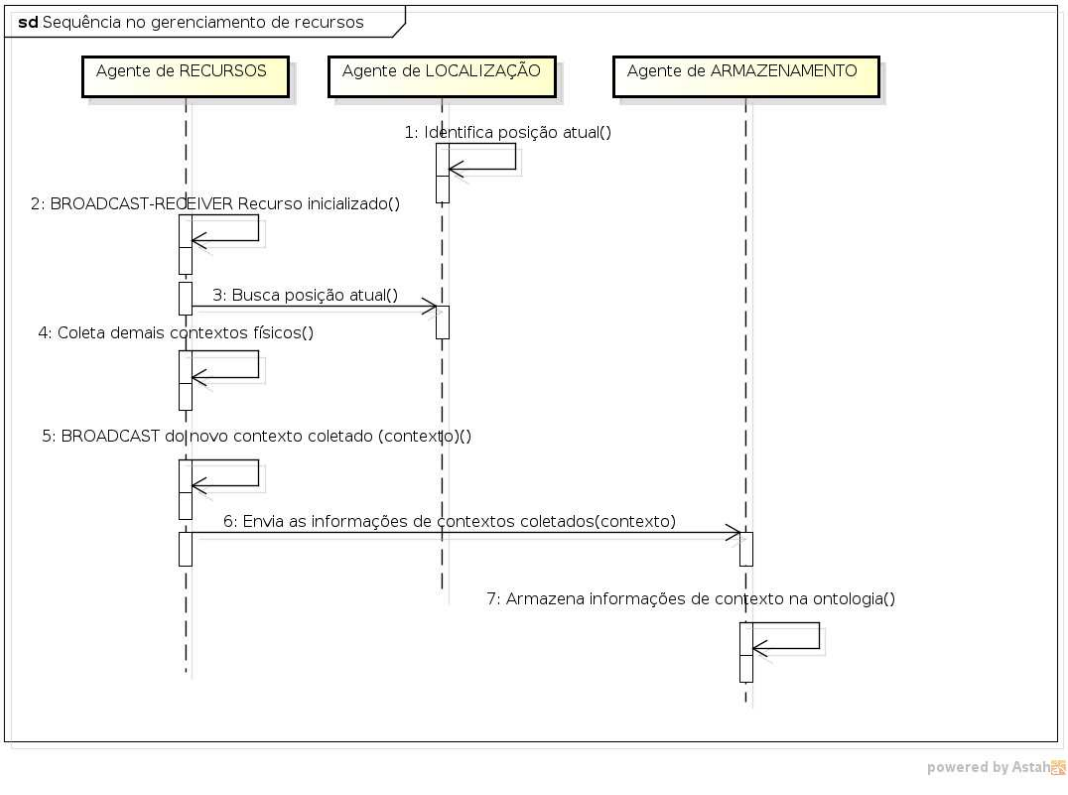

Figura 22. Diagrama de sequência do gerenciamento do recurso inicializado.

Para a realização da recomendação de recursos, o trabalho inicia com o agente de recomendação buscando mudanças de posição e de tempo ao longo do uso do dispositivo móvel, ocorrendo, solicita ao agente de armazenamento recursos utilizados na respectiva posição e tempo. Havendo recursos encontrados, estes são publicados no dispositivo móvel pelo *broadcast*

BROADCAST_RECOMENDACAO_RECURSO. A figura 23 contém um diagrama desta sequência de atividades destes agentes.

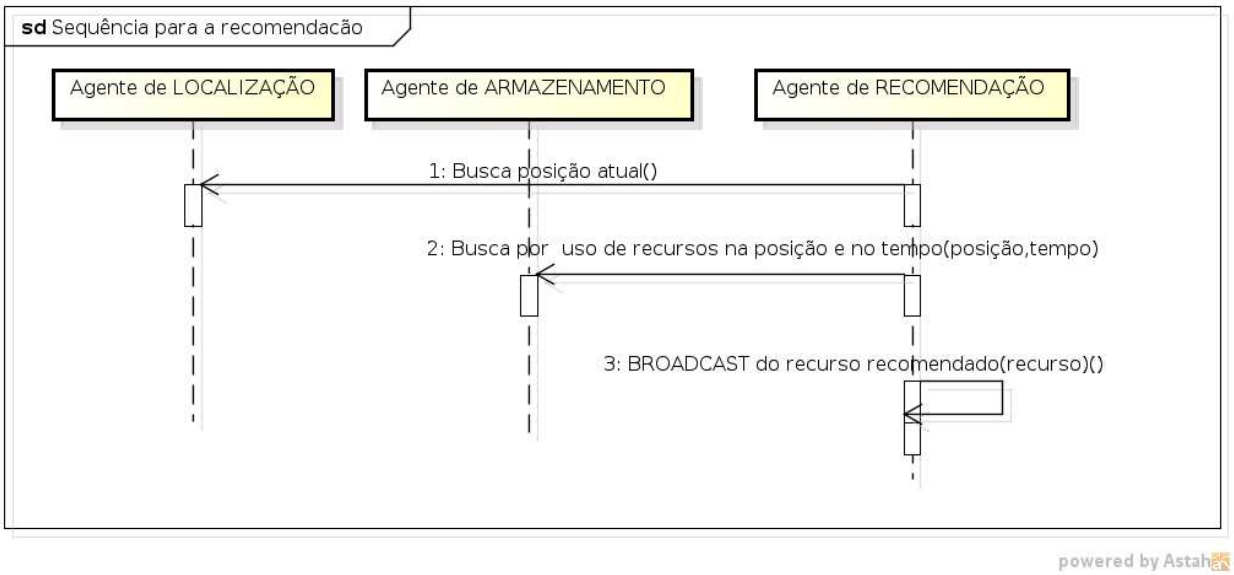

Figura 23. Diagrama de sequência de funcionamento dos agentes envolvidos na recomendação de recursos.

As sessões a seguir descrevem com mais detalhes o funcionamento de cada agente desta arquitetura de gerenciamento de contextos.

4.1 Agente de localização

As informações de localização durante o uso do dispositivo móvel é baseada na posição geográfica de latitude, longitude e altitude. A coleta destas informações necessita de tecnologias diferentes e interdependentes, com objetivo de atingir o registro completo da trajetória contínua do uso do dispositivo móvel. Basicamente são tecnologias diferentes dependendo se o dispositivo encontra-se em um local externo ou interno.

Para coletar a posição geográfica, o sistema identifica se o presente local, caracterizado como externo, está com sinal satisfatório e com quantidade de satélites suficientemente visíveis. A partir do momento que isso não será mais possível, encontrando-se em um local interno, é traçado um caminho imaginário na posição geográfica baseando-se dos sensores presentes no dispositivo.

A figura 24 demonstra locais percorridos por um dispositivo móvel, sendo eles em ambiente externo, tendo visibilidade aos satélites do sistema de posicionamento global (GPS - Global Positioning System), ou em ambiente interno, por exemplo, um prédio ou uma casa, onde não há visibilidade ao sinal de GPS e se faz necessário o uso de uma combinação de sensores para identificar os diferentes locais percorridos pelo dispositivo móvel.

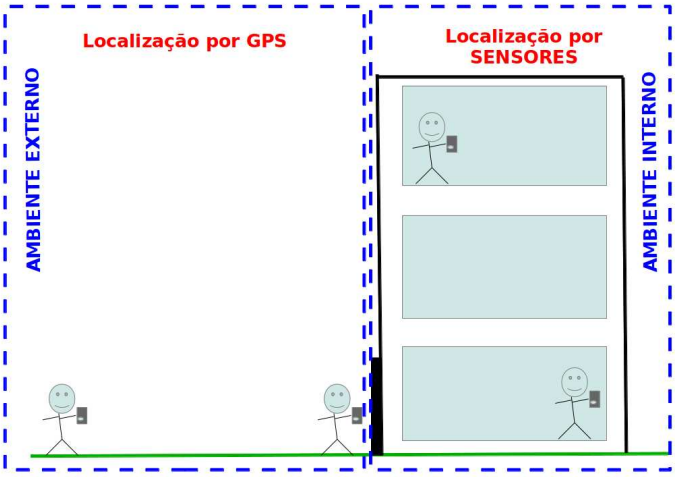

Figura 24. Ambientes e forma de localização de dispositivo móvel.

Neste trabalho, é mapeado o local e momento em que um recurso é solicitado. A informação de local é baseada na posição geográfica, isto é, a posição na longitude, latitude e altitude, servindo respectivamente em um plano cartesiano em três dimensões como eixo x, y e z.

A identificação da posição geográfica na plataforma Android é através da classe *android.location.Location*, contendo os métodos: *getLatitude()*, *getLongitude()* e *getAltitude()* para encontrar respectivamente a latitude, longitude e altitude.

Constantemente o agente de recursos deve identificar se há ou não sinal de GPS no ponto em que está situado o dispositivo. Caso não exista sinal satisfatório de GPS, inicia-se um mapeamento do local percorrido utilizando os recursos de sensores presentes no dispositivo móvel. A indicação do sinal de GPS é identificada através de dois métodos:

- a classe *android.location.Location* utilizando o método *getAccuracy()*. O retorno deste método é a indicação da distância em metros da precisão das coordenadas (latitude e longitude) indicadas pelo sensor, isto significa que, quanto menor este número mais preciso será a coordenada indicada para latitude e longitude. Para o uso desta classe, é necessário adicionar as permissões de usuário no projeto Android: ACCESS_FINE_LOCATION e INTERNET;
- com o objeto retornado do método anterior, utiliza-se o método *getMaxSatellites()* para saber o número de satélites visíveis. Para ter uma localização consistente é necessário formar uma triangulação, isto é, a quantidade mínima de três satélites visíveis.

O mapeamento de um percurso em ambientes internos é um trabalho combinado de dois sensores suportados na plataforma 4.0 do Android:

- **Acelerômetro**: identifica a distância percorrida pelo dispositivo multiplicando a diferença de aceleração e tempo.
- **Giroscópio ou orientação**: indica a torção do dispositivo nos 3 eixos: x, y e z, por exemplo, a posição em que se encontra no bolso ou na mão do usuário. Ele serve como comparativo à direção indicada pelo acelerômetro para saber se o usuário permanece em movimento na direção indicada pelo acelerômetro ou mudou de direção.

O trabalho dos sensores é indicar a continuação do último ponto de latitude, longitude e altitude mapeado através do sinal de GPS com a utilização da classe *android.location.Location*,

aumentando ou diminuindo os seus valores conforme o dispositivo muda de posição em um ambiente interno.

No momento em que o agente de recursos identificar que existe sinal de GPS satisfatório, voltará o mapeamento através deste. A figura 25 demonstra o diagrama de atividades deste agente de localização.

A intensidade do sinal GPS é identificada através da classe android.location GpsStatus, utilizando a função getMaxSatellites(). Se a quantidade de satélites visíveis for de no mínimo três é possível realizar uma triangulação do ponto onde está, porém nos experimentos realizados, os resultados da leitura GPS estavam de maneira aceitável com no mínimo seis satélites.

Outro parâmetro utilizado para decidir se é possível ler os valores informados na leitura do GPS, é a precisão informada por este sensor. A precisão é um valor aproximado, informado pelo sensor, que indica a medida em metros do raio de precisão em torno das coordenadas retornadas pelo GPS. Quanto maior este número, menor será a precisão do GPS. Para este trabalho, foi utilizado um limite máximo de precisão de quinze metros, isto é, se a precisão retornada pelo sensor for maior que este número, então estas coordenadas não serão aceitas.

Em resumo, a posição geográfica com o uso do GPS pode ser utilizada verificando a intensidade do sinal GPS através de duas medidas:

- mínimo de seis satélites visíveis;
- máximo de quinze metros de precisão.

A coleta das coordenadas GPS é realizada com o uso da classe “LocationListener”. Uma vez instanciada esta classe no construtor do agente de localização, a plataforma Android chama automaticamente a função “onLocationChanged” contida nesta classe, para informar alterações nas coordenadas GPS. Está demonstrado abaixo o uso desta classe no agente de localização.

Listagem 11. Leitura da posição geográfica juntamente com quantidade de satélites e precisão.

```
myLocationListener = new LocationListener() {
@Override
public void onLocationChanged(Location location) {
        mAccuracy   = location.getAccuracy();
        mSatellites = location.getExtras().getInt("satellites");
        if (mAccuracy <= LIMITE_ACCURACY && mSatellites >= LIMITE_SATELITES) {
                mLatitude =  location.getLatitude();
                mLongitude = location.getLongitude();
                mAltitude  = location.getAltitude();
                mLastTimeGPS = System.currentTimeMillis();
        }
}
};
```

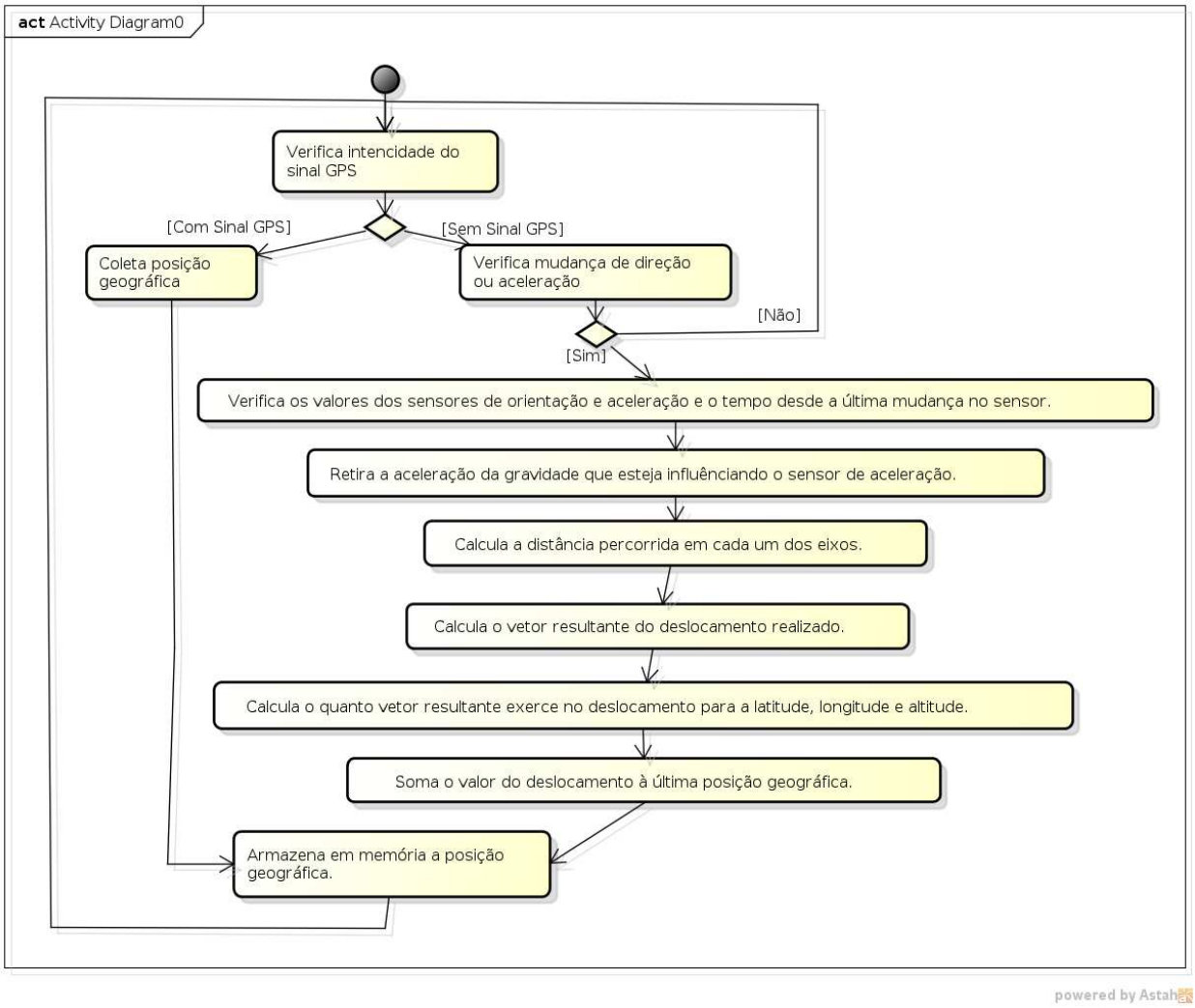

Figura 25. Diagrama de atividades do agente de localização.

A classe principal do agente de localização estende a classe “Activity” e implementa a classe “SensorEventListener” para poder realizar as leituras dos sensores de aceleração e orientação. Abaixo está a declaração da classe do agente de localização.

As mudanças identificadas nos sensores são informadas automaticamente na função “onSensorChanged”, onde recebe mudanças de todos os sensores definidos na classe, portanto, um teste para identificar qual sensor notificou mudança deve ser realizado. Abaixo está a demonstração desta função no agente de localização.

Listagem 12. Identificação de mudança de valores nos sensores.

```
@Override
public void onSensorChanged(SensorEvent event) {
        switch(event.sensor.getType()) {
        case Sensor.TYPE_ACCELEROMETER:
          for(int i=0; i<3; i++) {
            accelValues[i] = event.values[i];
```

```
            }
        break;
           case Sensor.TYPE_MAGNETIC_FIELD:
             for(int i=0; i<3; i++) {
                    compassValues[i] = event.values[i];
     }
 }
 // Verifica se ocorreu uma mudança de direção
 if(SensorManager.getRotationMatrix( inR, null, accelValues, compassValues)) {
      SensorManager.remapCoordinateSystem(inR, SensorManager.AXIS_X, SensorManager.AXIS_Y,
outR);
      SensorManager.getOrientation(outR, orientacaoValues);
  }
}
```

Esta sendo utilizado um atraso mínimo entre os eventos de leitura dos sensores de aceleração e orientação. Ele não pode ser muito rápido lendo variações pequenas ocasionadas por um ruído, e nem muito lento para não identificar uma mudança de direção. Este atraso é definido pela constante da classe "SensorManager". A plataforma Android tem a capacidade de realizar leitura aos eventos dos sensores sem indicação alguma de atraso entre as leituras, para isso utiliza-se a constante SENSOR_DELAY_FASTEST. A constante que se mostrou mais coerente neste trabalho foi SENSOR_DELAY_GAME. Abaixo está a demonstração da definição do intervalo de tempo para leituras do sensor acelerômetro.

Listagem 13. Definição do intervalo de tempo entre leituras dos sensores.

```
protected void onResume() {
 super.onResume();
 mSensorManager.registerListener(this,accel,SensorManager.SENSOR_DELAY_GAME);
}
```

A aceleração da gravidade influencia nos eixos X e Y do dispositivo móvel. A figura 10 indica a direção dos eixos X, Y e Z na plataforma Android e a constante "SensorManager.STANDARD_GRAVITY" indica o valor para a aceleração da gravidade.

Quando Y está em 0 graus a aceleração da gravidade está influenciando apenas no eixo Y e, quando X está em 0 graus a aceleração da gravidade está influenciando apenas no eixo X.

Segundo os experimentos realizados, a influencia da aceleração da gravidade é aplicada totalmente sobre um eixo conforme a relação abaixo:

- sobre eixo Z: quando ângulo sobre eixos X e Y for igual a 0 (zero) ou 180;
- sobre eixo X: quando ângulo sobre eixo X for igual a 0 (zero) ou 180 e eixo Y for igual a 90 ou 270 (independente da posição do eixo Z);
- sobre eixo Y: quando ângulo sobre eixo Y for igual a 0 (zero) ou 180 e eixo X for igual a 90 ou 270 (independente da posição do eixo Z);

Isso significa que a inclinação nestes eixos entre 0 (zero) e 90 representam uma relação de percentual sobre o quanto a aceleração da gravidade está atuando sobre o respectivo eixo.

Sendo assim, conclui-se a seguinte formula para a influencia da aceleração da gravidade e tendo como base os valores radianos de rotação em cada eixo para calcular os senos e cossenos:

- sobre eixo Y: aceleração da gravidade * sen (X) * cos (Y)
- sobre eixo X: aceleração da gravidade * sen (Y)
- sobre eixo Z: aceleração da gravidade * cos (X) * cos (Y)

Abaixo está a demonstração do cálculo para identificar a quantidade de aceleração da gravidade está exercendo em cada um dos eixos, utilizando a orientação no formato de valores radianos.

Listagem 14. Cálculo para retirar a influência da aceleração de gravidade nos sensores.

```
double x_gravidade = SensorManager.STANDARD_GRAVITY *
                    Math.sin( orientacaoValues[2]) * Math.cos(orientacaoValues[1]);
double y_gravidade = SensorManager.STANDARD_GRAVITY *
                    Math.sin(orientacaoValues[1]) ;
double z_gravidade = SensorManager.STANDARD_GRAVITY *
                   Math.cos(orientacaoValues[2]) * Math.cos(orientacaoValues[1]);
```

A cada mudança ocorrida com os sensores, é calculada a distância percorrida em cada um dos eixo é encontrada pela multiplicação da aceleração e do tempo. Mas no caso dos sensores analisados, mediremos a diferenças de aceleração ocorrida desde a última mudança nos sensores.

Distância = $\sum$ (aceleração atual – aceleração anterior) * (tempo atual – tempo anterior)

Equação 4. Distância percorrida.

Listagem 15. Cálculo da distância percorrida em cada eixo do dispositivo móvel.

```
double distanciaX = ((double)lastAccelValues[ACCEL_X] -          (double)accelValues[ACCEL_X] )
*
        ( (System.currentTimeMillis() - mLastTimeSensor)/(double)100);
double distanciaY = ((double)lastAccelValues[ACCEL_Y] -      (double)accelValues[ACCEL_Y] ) *
        ( (System.currentTimeMillis() - mLastTimeSensor)/(double)100);
double distanciaZ = ((double)lastAccelValues[ACCEL_Z] -      (double)accelValues[ACCEL_Z] ) *
        ( (System.currentTimeMillis() - mLastTimeSensor)/(double)100);
```

O próximo passo deste agente é calcular o vetor resultante das acelerações identificadas nos três eixos.

Portanto, utiliza-se o Teorema de Pitágoras e relação de um ângulo pela sua tangente para descobrir um vetor resultante. Para isso, calcula-se em duas partes, primeiro encontrando o vetor resultante entre os eixos X e Y, e após a resultante final entre a primeira resultante encontrada e o eixo Z.

Listagem 16. Cálculo do vetor resultante de deslocamento na posição geográfica.

```
double resultante_XY =
   Math.sqrt( (distanciaX * distanciaX) + (distanciaY * distanciaY) );
double angulo_XY     = Math.atan( distanciaY / distanciaX );
```

```
double resultante_final =
   Math.sqrt( (resultante_XY * resultante_XY) + (distanciaZ * distanciaZ) );
double angulo_final_Z_XY   = Math.atan( resultante_XY / distanciaZ );
```

Utilizando as propriedades trigonométricas, identifica-se o deslocamento realizado nas coordenadas geográfica (latitude, longitude, altitude). O uso dos sensores nos apresentam o ângulo formado em cada eixo, juntamente com o resultado do cálculo acima, retornando a distancia percorrida, encontra-se o deslocamento nas coordenadas cartesianas, segundo a relação abaixo:

deslocamento final Z = cos ângulo de orientação Z * deslocamento resultante

deslocamento final X = cos ângulo de orientação X * deslocamento resultante

deslocamento final Y = cos ângulo de orientação Y * deslocamento resultante

O deslocamento final em Y indica deslocamento na direção latitude, por sua vez, o deslocamento final em X indica deslocamento na direção longitude e o deslocamento final em Z indica deslocamento na direção altitude. Abaixo está o calculo realizado no protótipo construído.

Listagem 17. Cálculo da influência do vetor resultante no deslocamento na posição geográfica.

```
double deslocamento_horizontal_Z =
            Math.cos( orientacaoValues[ORIENT_Z])  *  resultante_final;
double deslocamento_horizontal_X =
            Math.cos( orientacaoValues[ORIENT_X])  *  resultante_final;
double deslocamento_horizontal_Y =
            Math.cos( orientacaoValues[ORIENT_Y])  *  resultante_final;
```

A última tarefa na localização por sensores é somar o deslocamento realizado nas direções da latitude, longitude e altitude. Para isso é necessário somar um valor em metros em uma coordenada geográfica. Segundo (LONGLE et al, 2005) a relação de uma distância em uma latitude se difere da relação de distância em uma longitude, isso por causa da grande diferença no tamanho da circunferência para a linha de longitude dependendo da distância da Linha do Equador, isto é, dependendo da posição da latitude. Esta relação apresenta as seguintes medidas:

1' latitude = aproximadamente 1852 metros

1' longitude = aproximadamente 1852 metros * cos (grau latitude)

Para realizar este cálculo no protótipo, foram convertidas as coordenadas latitude e longitude para o formato de minutos utilizando a função “Location.convert(<posição>, Location.FORMAT_MINUTES)”. A listagem 18 demonstra o cálculo construído no protótipo.

Listagem 18. Cálculo para somar o deslocamento em metros em uma coordenada latitude e longitude.

```
String latitude_formato_minutos =
        Location.convert(mLatitude,Location.FORMAT_MINUTES);
int lat_graus = Integer.parseInt(latitude_formato_minutos.split(":")[0]);
double lat_minutos =
```

```
Double.parseDouble(latitude_formato_minutos.split(":")[1].replace(',', '.'));
lat_minutos = ((lat_minutos * DISTANT_1_MINUTO_LATITUDE ) +
        deslocamento_horizontal_X) / DISTANT_1_MINUTO_LATITUDE;
lat_minutos = lat_minutos * Math.cos(lat_graus);
lat_graus   = lat_graus + (int)(lat_minutos / 60);
lat_minutos = lat_minutos - ( ((int)(lat_minutos / 60)) * 60 );
latitude_formato_minutos = lat_graus + ":" +       String.format("%.5f",lat_minutos).replace('.', ',');

String longitude_formato_minutos =       Location.convert(mLongitude,Location.FORMAT_MINUTES);
int lon_graus = Integer.parseInt(longitude_formato_minutos.split(":")[0]);
double lon_minutos = Double.parseDouble( longitude_formato_minutos.split(":") [1].replace(',', '.') );
lon_minutos = ((lon_minutos * DISTANT_1_MINUTO_LATITUDE ) +      deslocamento_horizontal_Y) /
DISTANT_1_MINUTO_LATITUDE;
lon_minutos = lon_minutos * Math.cos(lon_graus);
lon_graus   = lon_graus + (int)(lon_minutos / 60);
lon_minutos      = lon_minutos - ( ((int)(lon_minutos / 60)) * 60 );
longitude_formato_minutos = lon_graus + ":" +
        String.format("%.5f",lon_minutos).replace('.', ',');
```

O resultado final da localização pelos sensores acelerômetro e orientação substitui a última posição para latitude, longitude e altura, independentemente de qual foi o processo que gerou a posição anterior.

4.2 Agente de recursos

O agente de recursos tem a responsabilidade de acompanhar e armazenar todas as atividades referentes ao contexto lógico do uso dos recursos de um dispositivo móvel. Ele recebe a informação que uma aplicação está em uso e adiciona dados de espaço, constituído pela localização global recebida através do agente de localização; e instante que estas informações estão sendo processada. Quando executado em um novo hardware, é este agente que realiza a identificação destes demais atributos. Ele organiza as informações no formato exigido pelo agente de armazenamento e as envia na forma de *broadcast* dentro da plataforma Android, mantendo assim a independência e coesão dos agentes desta arquitetura.

Para cada recurso utilizado no dispositivo móvel, o agente de recurso verifica as seguintes variáveis de contexto:

- o recurso que está sendo utilizado, neste trabalho;
- data e hora em que o recurso foi solicitado;
- ponto de localização do uso do recurso;
- tipo do local do uso do recurso (externo ou interno);
- situação da movimentação da pessoa, isto é, armazenar a indicação se a pessoa está

parada, caminhando ou correndo. Esta classificação é realizada através da verificação da velocidade atual, segundo os seguintes critérios listados abaixo, e coletados no Android quando situado em ambientes externos pela função “Location.getSpeed()”, quando em ambientes internos esta situação sempre é tomada como “caminhando”.

- parada, indicada quando tiver velocidade igual a zero;
- caminhando, indicada quando tiver velocidade até 6 m/s;
- correndo, indicada quando tiver velocidade acima de 6 m/s.

A indicação que um recurso está sendo inicializado no dispositivo móvel vem do sistema *Desktop* Semântico. Sendo assim, o agente de recursos inicia seu trabalho no momento que receber esta indicação. Como resposta a esta indicação do *desktop* semântico, serão enviadas todas as informações coletas sobre o contexto de localização.

A primeira vez que este agente executa, ele coleta as informações específicas do *hardware* o qual está instalado (modelo, memória, cpu, tela), para isso ele utiliza na plataforma Android a classe “Build” com os atributos: “Build.MODEL”, “Build.MEMORY”, “Build.CPU”, “Build.DISPLAY”.

Todas as informações coletadas são encaminhadas e disponibilizadas dentro do dispositivo móvel na forma de *broadcast*, o qual na arquitetura apresentada é utilizado pelo agente de armazenamento. A ação deste *broadcast* é chamada de BROADCAST_NOVO_CONTEXTO_RECURSO. As informações são repassadas na forma de atributo carregados pela função “intent.putExtra(<atributo>,<conteúdo>)”, e nomeados por:

- local_id, local_latitude, local_longitude, local_altitude, local_tipo;
- software_id (formado pelo nome do pacote), software_titulo;
- acao_id, acao_situacao_movimentacao;
- momento_id (formado por um valor de data e hora do momento atual – *timestamp*);
- hardware_id (formado pelo número serial), hardware_tela, hardware_memoria, hardware_cpu, hardware_modelo

4.3 Agente de armazenamento

O agente de armazenamento tem a responsabilidade de organizadas e armazenadas em uma ontologia as informações recebidas referentes ao contexto coletado, produzindo assim, um arquivo OWL na estrutura OWL-DL, permitindo as descrições lógicas necessárias para a inferência proposta neste trabalho. Todas as novas informações armazenadas na ontologia são divulgadas na forma de *broadcast* para serem recebidas pelo agente do trabalho de federação de contextos.

Este agente armazena informações na ontologia utilizando a biblioteca Androjena, a qual disponibiliza utilização do *framework* Jena dentro da plataforma Android.

Para a elaboração da ontologia foi utilizada a metodologia Ontology Development 101 (RAUTENBERG et al, 2008), por se tratar de uma metodologia simples e bem objetiva para a pequena ontologia que propunha a montar neste trabalho.

A ontologia para o armazenamento e inferência dos dados de contexto do uso de dispositivos móveis é formada pelas classes listadas abaixo. A construção das regras de

inferências foram apoiadas nos conceitos de (LIMA, 2005).

- **Local**: representando um ponto geográfico;
- **Momento**: representando um ponto na linha de tempo;
- ***Software***: representando os *software* instalados em um *hardware*. Esta classe contém a propriedade de objeto "contem", uma propriedade do tipo *functional* para a classe *Hardware*, representando respectivamente, que para cada instância da classe *Software* esta ligado a apenas uma instância da classe *Hardware.*
- ***Hardware*:** representando as máquinas, *hardwares*, independente da sua mobilidade ou tamanho.
- **Ação**: representando todas as ações de utilização dos software instalados em um dispositivo móvel, uma caixa-preta, indicando os locais e momentos em que uma ação ocorreu. Esta classe contém as propriedades de objetos, sendo todas do tipo *functional*:
  - "executadaPor", tendo como *domain* a classe ação e *range* a classe *Software*;
  - "iniciadaEm", tendo como *domain* a classe ação e *range* a classe Momento;
  - "realizadaEm", tendo como *domain* a classe ação e *range* a classe Local.

  O código abaixo está a declaração da propriedade de objeto funcional "executadaPor".

Listagem 19. Declaração da propriedade funcional na estrutura da ontologia.

```
<owl:FunctionalProperty rdf:ID="executadaPor">
   <rdfs:domain rdf:resource="#acao"/>
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
   <rdfs:range rdf:resource="#software"/>
</owl:FunctionalProperty>
```

Na figura 26 está o modelo de classes que representa a estrutura da ontologia elaborada, contendo as propriedades de objetos representadas pela ligação entre as classes, e dentro das classes, a representação das propriedades dos tipos de dados.

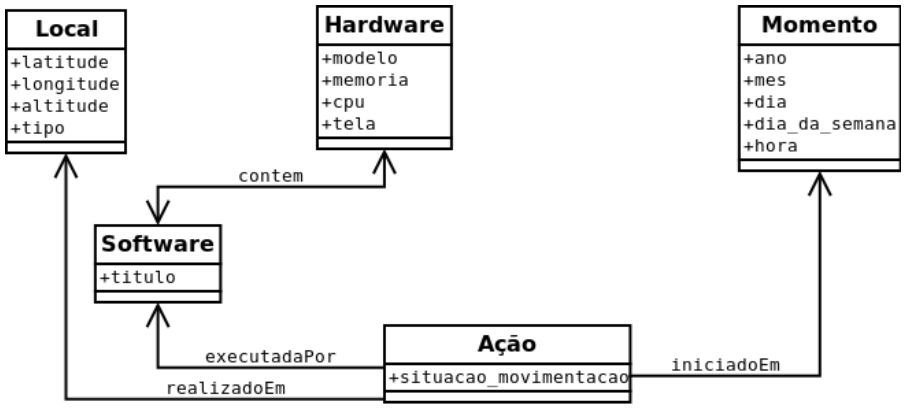

Figura 26. Estrutura da ontologia de contexto.

A ontologia foi criada no software Protege 4.2, e utilizando o *plugin* OWLViz é possível visualizar as classes na figura 27.

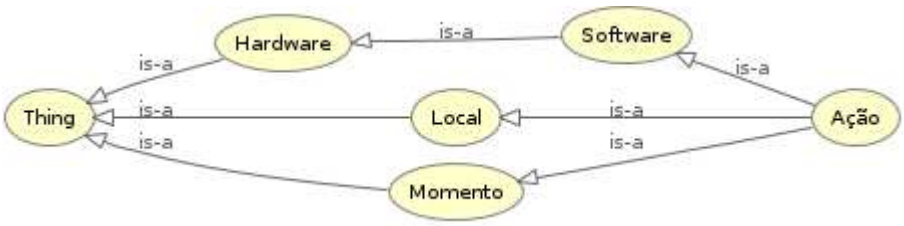

Figura 27. Visualização da ontologia com o *plugin* OWLViz.

Na implementação do agente de armazenamento, a primeira atividade é a leitura do arquivo da ontologia, especificando o modelo de estrutura que é encontrado no arquivo:

Listagem 20. Leitura inicial da ontologia no agente de armazenamento.

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setNamespaceAware(true);
dbf.setFeature("http://xml.org/sax/features/namespace-prefixes", true);
OntModel myOnt = ModelFactory.createOntologyModel();
dm = myOnt.getDocumentManager();
ontologiaModelo = dm.getOntology(caminhoOntologia, OntModelSpec.OWL_DL_MEM );
```

A especificação do modelo de ontologia *OWL_DL_MEM* indicado acima, representa a linguagem OWL-DL (DICKINSON, 2012).

A listagem 21 descreve a criação de um indivíduo da classe “acao” com suas propriedades de dados e objetos no protótipo desenvolvido.

Listagem 21. Inclusão de indivíduo da classe “acao” utilizando o Androjena.

```
OntClass classAcao       = ontologiaModelo.getOntClass(NS + "acao");
Individual individuoAcao  =
        ontologiaModelo.createIndividual(NS + b.getStringExtra("acao_id"),
        classAcao);
individuoAcao.addProperty       ( ontologiaModelo.getProperty("situacao_movimentacao"),
                        b.getStringExtra("acao_situacao_movimentacao"));
individuoAcao.addProperty
        (ontologiaModelo.getProperty("realizadaEm"), individuoLocal );
individuoAcao.addProperty
        (ontologiaModelo.getProperty("iniciadaEm"), individuoMomento );
individuoAcao.addProperty
        (ontologiaModelo.getProperty("executadaPor"), individuoSoftware );
```

Na primeira vez que este agente for executado em um dispositivo, todas as informações contidas na ontologia do gerenciador de contexto serão enviadas para a federação, isto é, desde as informações referentes ao hardware até a ação de utilização do primeiro recurso. Portanto,

quando estas informações são recebidas, faz-se necessária a verificação da existência deste *hardware* e deste *software* antes de inseri-los, conforme demonstrado na listagem 22.

Listagem 22. Verificação da existência de um indivíduo na ontologia com Androjena.

```
Individual individuoSoftware =
        ontologiaModelo.getIndividual(NS + b.getStringExtra("software_nome") );
if (! individuoSoftware.isIndividual())  {
        individuoSoftware =
                ontologiaModelo.createIndividual(
                        NS + b.getStringExtra("software_nome"), classSoftware);
}
```

Após a sincronização inicial, o agente de armazenamento envia para a federação apenas as informações da ação com suas respectivas propriedades para as classes software, local e momento. Basicamente serão informações recebidas do agente de recursos, porém, neste agente elas são enviadas já indicando a estrutura de classes e propriedades contida na ontologia.

Este agente disponibiliza as informações através do envio de um *broadcast* de ação chamado BROADCAST_NOVOS_CONTEXTOS. Seu objetivo é manter informada a federação de contextos sobre todas as informações de contexto armazenadas. As informações armazenadas na ontologia são repassadas na forma de atributos, seguindo a mesma estrutura do *broadcast* BROADCAST_NOVO_CONTEXTO_RECURSO gerado pelo agente de recursos, sendo que as informações das propriedades de dados de um novo indivíduo da classe “hardware” e “software” são enviadas somente se eles ainda não existirem na ontologia. Se o indivíduo da classe “software” já existir na ontologia, então, apenas o seu nome é enviado neste *broadcast* informando a propriedade de objeto com a classe “acao”.

4.4 Agente de recomendação de recursos

O agente de recomendação faz uso das informações geradas pelo agente de recursos, identificando que o dispositivo móvel encontra-se em situação similar a encontrada quando utilizou algum recurso. Desta maneira, a medida que aumenta a quantidade de informações geradas pelo agente de recursos, aumentará a possibilidade de acerto dos recursos indicados pelo agente de recomendação, isto é, será um aprendizado constante.

Desta forma, quanto maior o tempo de utilização do software de gerenciamento de contexto mapeando os dados do contexto de uso do dispositivo, maior será a precisão de suas recomendações.

Um histórico de contextos armazenados formará um novo contexto lógico a partir da análise do agrupamento em um plano cartesiano das seguintes informações de:

- Localização juntamente com a mesma hora do dia e mesma situação de movimento: são consideradas ações na mesma hora do dia os que estiverem em uma faixa de trinta minutos para mais ou para menos do respectivo horário atual;
- Localização juntamente com o mesmo dia da semana e mesma situação de movimento;
- Localização juntamente com o mesmo dia do mês e mesma situação de movimento.

Um algoritmo analisará a densidade de uso de cada recurso para cada um dos três tipos de contexto lógico. Para isso necessitará de:

$$d = |\overline{AB}| = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2}$$

Equação 5. Distância entre dois pontos.

- Distância para o vizinho mais próximo, calculado a partir da distância entre dois pontos cartesianos de localização geográfica (latitude, longitude e altitude), isto representa um plano cartesiano em três dimensões. Na plataforma Android é possível calcular a distância entre dois pontos utilizando a função "Location.distanceBetween", porém ela é restrita a calcular distâncias entre pontos no plano, utilizando apenas a latitude e longitude. Para isso, utiliza-se a equação abaixo, onde "A" e "B" representam o ponto atual e um ponto armazenado na ontologia, e "x", "y" e "z" representam a latitude, longitude e altitude:

- Distância máxima para vizinhos mais próximos: neste agente de recomendação a distância máxima permitida até o ponto mais próximo em que foi registrado um uso do respectivo recurso será um parâmetro e portando poderá ser alterado conforme desejo de cada usuário.
- Percentual de confiabilidade da densidade para cada agrupamento de contexto lógico, onde será considerada:
  - a) quantidade de indivíduos da classe "acao" encontrados próximo a este local e segundo o critério de agrupamento no tempo encontrado (mesmo dia da semana, mesmo dia do mês ou mesma hora do dia). Se um software for selecionado por mais de um critério de agrupamento no tempo, considera-se o que tiver maior quantidade de indivíduos.
  - b) a quantidade total de períodos segundo o critério de agrupamento no tempo encontrado no item "a)", desde o "momento" mais antigo deste "software" até a data atual.

A confiabilidade será a divisão do item "a" pelo item "b":

$$\%\,confiabilidade = \frac{a : quantidade\ de\ períodos\ com\ ocorrência\ de\ uso}{b : quantidade\ total\ de\ períodos\ do\ contexto\ lógico}$$

Equação 6. Percentual de confiabilidade da recomendação.

Um exemplo desta confiabilidade seria um software que foi utilizado no mesmo dia da semana em quatro registros, e encontra-se do momento do primeiro indivíduo até a semana atual o número de dez semanas, conforme demonstrado na figura 28. Portanto para este exemplo, a confiabilidade é igual a (4 / 10), resultando em 0,4 de confiabilidade, ou, 40%.

Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8	Semana 9	Semana ATUAL

Figura 28. Exemplo de frequência de uso de um software no mesmo dia da semana.

Como resultado deste agente de recomendação estará:

- Nome do recurso que teve ocorrência em algum dos contextos lógicos e que o vizinho mais próximo esteja abaixo da distância máxima permitida.

- Percentual de confiabilidade para o contexto lógico o qual teve a maior quantidade de vizinhos abaixo da distância máxima permitida.

Devido ao alto tempo de processamento deste agente para cada novo ponto geográfico a sua execução tem um intervalo estabelecido de quinze segundos para verificar se ocorreu uma nova mensagem do agente de localização informando um novo ponto geográfico.

A figura 29 descreve as atividades realizadas por este agente de recomendação.

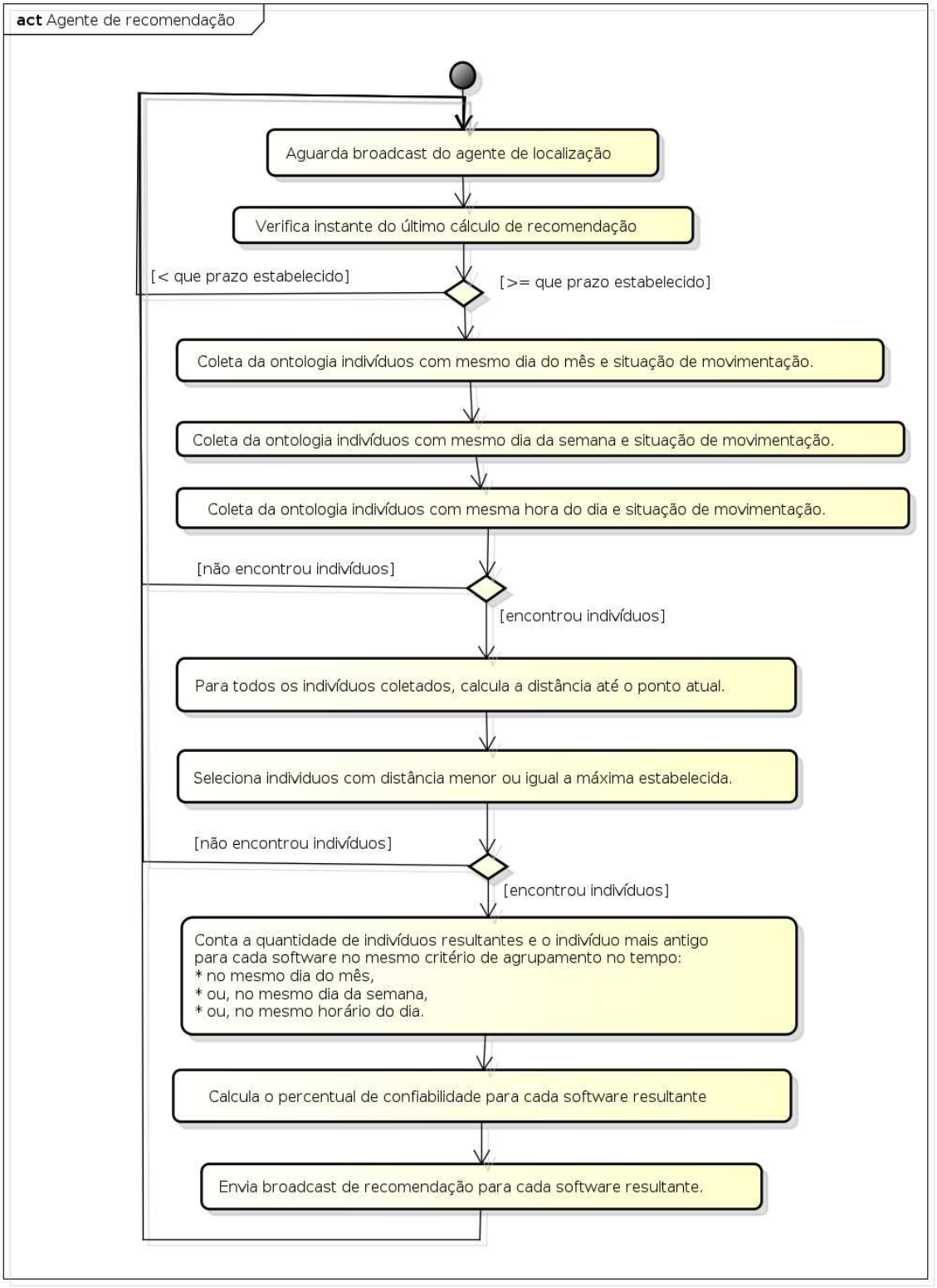

Figura 29. Diagrama de atividades do agente de recomendação.

Na figura 30 visualiza-se um exempl

o de registros de contexto localização (apenas latitude e longitude) juntamente com a hora do dia para dois recursos diferentes (em vermelho e em azul). Percebe-se um agrupamento do uso do recurso azul próximo às 18:00 horas e do recurso vermelho próximo das 06:00 e das 12:00 horas. Se um dispositivo móvel se encontrar próximo a estes locais, então, uma recomendação será gerada.

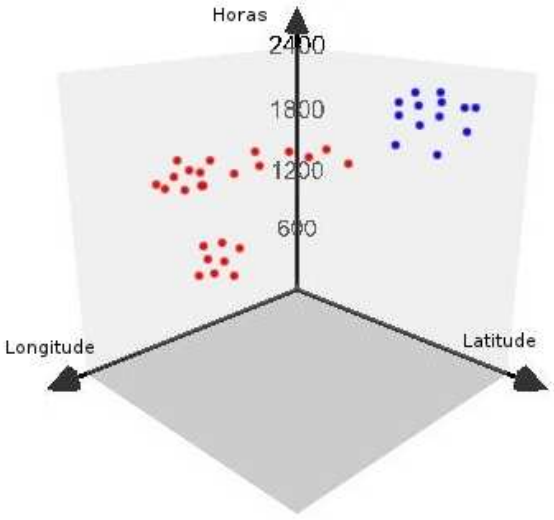

Figura 30. Visualização do contexto localização e hora para dois recursos.

Portanto, quanto maior o tempo que um dispositivo móvel estiver usando o gerenciador de contexto, mais coerente será a confiabilidade do agente de recomendação.

Para consultar e inserir os dados utiliza-se a linguagem SPARQL executada pela classe "*Query*" do Jena. A listagem 23 descreve a consulta SPARQL utilizada para encontrar indivíduos da classe "acao" com seu respectivo "local" no mesmo dia da semana que o atual e mesmo tipo de movimentação que o identificado pelo agente de localização.

Listagem 23. Consulta SPARQL para encontrar indivíduos da classe "acao" no mesmo dia da semana e mesmo tipo de movimentação.

```
PREFIX :<http://ensino.univates.br/~andrigodametto/owl/Ontology_gerenciador_contexto.owl#>
SELECT ?id_sw ?latitude ?longitude ?altitude ?mes ?dia ?dia_semana ?hora ?situacao
WHERE {
?id_acao :situacao_movimentacao ?situacao .
?id_momento :dia ?dia .
?id_momento :dia_semana ?dia_semana .
?id_momento :mes ?mes .
?id_momento :hora ?hora .
?acao :iniciadaEm ?momento .
?acao :realizadaEm ?local .
?acao :executadaPor ?software .
?id_local :latitude ?latitude .
?id_local :longitude ?longitude .
?id_local :altitude ?altitude .
```

```
 ?id_local :tipo ?tipo_local .
 ?id_sw :titulo ?titulo_sw .
FILTER  (
  ?acao = ?id_acao &&
  ?momento = ?id_momento &&
  ?local = ?id_local &&
  ?id_sw = ?software &&
  ?situacao = \"" + b.getStringExtra("POSICAO_SITUACAO") +"\"  &&
  ?dia_semana = " + mDiaSemanaAtual + "
)
}
ORDER BY DESC (?id_momento )
```

Para encontrar os indivíduos da classe "acao" com mesmo dia do mês, utiliza-se a mesma consulta SPARQL da listagem 23 alterando o filtro "?dia_semana = " + mDiaSemanaAtual + "" pelo filtro "?dia = " + mDiaAtual + "" .

A recomendação é disponibilizada na forma de mensagens *broadcast* disponível para qualquer outra aplicação instalada no dispositivo móvel, desde que para isso, seja conhecido a estrutura deste *broadcast*. A aplicação que utilizará esta informação será o *Desktop* Semântico. A estrutura deste *broadcast* é composta por:

- Nome de sua ação, chamada BROADCAST_RECOMENDACAO_RECURSO;
- Parâmetro chamado RECURSO_NOME, contendo o nome completo do recurso recomendado;
- Parâmetro chamado RECURSO_CONFIABILIDADE, contendo um grau de confiabilidade da recomendação, baseado na quantidade de vezes que este recurso foi utilizado.

5 VERIFICAÇÃO DO SISTEMA

A realização de atividades de verificação se faz necessário para garantir que o produto atende às suas especificações e é plenamente funcional quando inserido em um ambiente real de uso.

Durante a verificação do produto desenvolvido, foi utilizado um campo de futebol por propiciar efetiva identificação da localização por meio do GPS, evitando assim, a possibilidade do não funcionamento de um dos agentes desenvolvidos em decorrência de erros na identificação da localização. A arquitetura neste trabalho não tem foco na visualização de interfaces no dispositivo móvel, apenas gerenciar a informações recebidas e enviar mensagens, consequentemente foi criado algumas telas para dispor os avisos gerados pelos agentes de recomendação e de armazenamento e também uma mudança no agente de localização para que armazene todos os pontos de localização do percurso realizado. Primeiramente estão descritas verificações realizadas na localização e a seguir está descrita a verificação da recomendação que faz uso de todas as informações geradas pelos demais agentes desta arquitetura, demonstrando os resultados alcançados de toda a arquitetura.

É importante especificar aqui o *hardware* utilizado, pois há diferenças na capacidade e sensibilidade dos sensores presentes em cada modelo de *hardware* comercializado, podendo haver diferença em algum resultado demonstrado nesta verificação quando testado em outro modelo, portanto, o modelo utilizado foi Samsung GT-I9300.

Portanto, a verificação do protótipo do modelo é dividida em duas etapas:

- Verificação da localização;
- Verificação da recomendação.

5.1 Verificação da localização

Verificação da localização através da identificação do percurso em ambientes externos e internos, além da situação do movimento (sentado, andando, correndo) em cada ponto. Para possibilitar este teste foi realizado uma coleta de pontos de localização (latitude, longitude e altitude). A lista de pontos coletados em cada tipo de ambiente é utilizada por uma aplicação de testes no dispositivo móvel Android com objetivo de integrar estes pontos à API de mapas *com.google.android.maps*, gerando alguns percursos por onde foi utilizado cada recurso.

Para esta verificação do agente de localização o protótipo foi modificado para armazenar em memória a sequência de coordenadas geográfica identificadas pelo GPS. Juntamente a isso, a parte do protótipo que calcula a localização em ambientes internos, utilizaria apenas a primeira coordenada GPS, simulando assim a entrada em um ambiente interno e também modificada para armazenar em memória a sequência de novas coordenadas geográfica. Desta forma o teste da localização em ambiente interno foi testado em ambiente externo para ser comparado com o percurso registrado através da leitura do GPS.

Dos testes realizados com o agente de localização, a figura 31 demonstra o caminho percorrido utilizando coordenadas GPS na linha em cor azul e na linha vermelha está o mesmo percurso, mas este foi registrado pela leitura dos sensores de aceleração e orientação partindo de um mesmo ponto.

Figura 31. Visualização de um percurso comparativo entre GPS e sensores.

Realizando verificações conforme apresentado na figura 31, percebeu-se muita variação na consistência do caminho indicado apenas pelos sensores de aceleração e orientação. Em alguns momentos a direção e distância eram identificadas e em outros momentos não identificava mudança de direção. O acúmulo da divergência entre GPS e sensores ao passar por caminhos mais distantes e por um tempo maior, devido a imprecisão dos sensores e arredondamentos existente nos cálculos, pode inviabilizar a funcionalidade do sensor de localização interna.

A característica identificada para atingir melhores resultados é a forma como acontece as mudanças de direção: elas devem ter uma aceleração durante um pequeno instante, pois caso contrário, a posição é retornada para o ponto anterior. A figura 32 demonstra o deslocamento do valor da latitude um percurso onde as mudanças de direção foram bem suaves, sem gerar mudanças no deslocamento, o mesmo ocorreu para a longitude. Este gráfico, utilizado apenas a escala dos minutos, mostra que mesmo havendo deslocamento, o algoritmo de localização interna registrou o mesmo ponto, neste caso foi -29°:19,9050' indicado pela linha em cor azul. Partindo do mesmo ponto e realizando o mesmo percurso, porém com mudanças mais bruscas de direção, o algoritmo de localização interna registrou os pontos indicados na linha em cor vermelha, sendo o percurso correto.

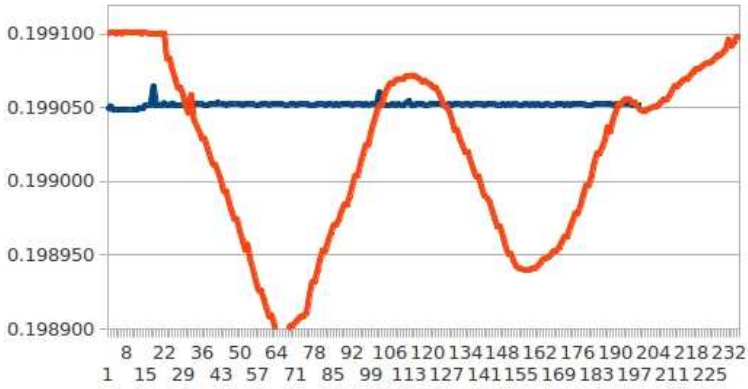

Figura 32. Gráfico da latitude medida com sensores e com mudança de direção de forma suave.

Em outro percurso, o gráfico da figura 33 indica os valores de latitude encontrados pelo GPS (em linha azul) e pelos sensores (em linha vermelha). Percebe-se também um deslocamento

do ponto inicial neste percurso, isso foi identificado em todos os testes onde o percurso foi iniciado já com mudança de direção. Este percurso foi realizado caminhando com o dispositivo móvel segurado na mão e em posição aproximadamente horizontal, além de realizar as mudanças de direção de forma mais bruscas. Este foi um dos testes que os sensores de aceleração e orientação traçaram um percurso próximo ao identificado pelo GPS.

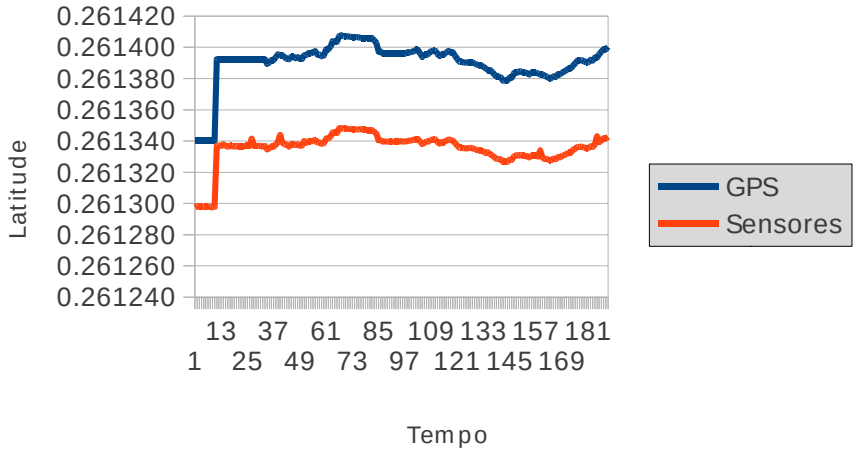

Figura 33. Gráfico comparativo entre pontos encontrados na latitude com GPS e com sensores.

A verificação da localização mostrou que o agente de localização atende de forma satisfatória os objetivos deste trabalho para ambientes externos, porém para ambientes internos foi atendido parcialmente os objetivos deste trabalho, pois seus resultados dependiam de uma utilização de forma controlada, pois caso contrário não seria registrado deslocamento.

5.2 Verificação da recomendação

A verificação da recomendação é a prova final para saber se todas as informações do contexto de uso de recursos foram coletadas e armazenadas corretamente na ontologia.

Para esta verificação e apenas durante o período de testes, foram armazenados na memória do dispositivo móvel, todas as recomendações realizadas: a localização geográfica, o instante e o recurso. Desta forma é possível traçar um gráfico com o local e instante onde cada recurso foi recomendado com o local e instante onde o respectivo recurso foi utilizado.

Também durante os testes, foram incluídas manualmente na ontologia registros de uso de *software* em diferentes locais e diferentes momentos.

Nos testes do agente de recomendação não estava instalado no dispositivo móvel o Desktop Semântico, portanto, para o usuário do protótipo visualizar uma recomendação foi criado uma pequena aplicação que ficava ouvindo as recomendações e gerasse uma mensagem de notificação. A figura 34 demonstra uma notificação gerada durante a verificação do sistema.

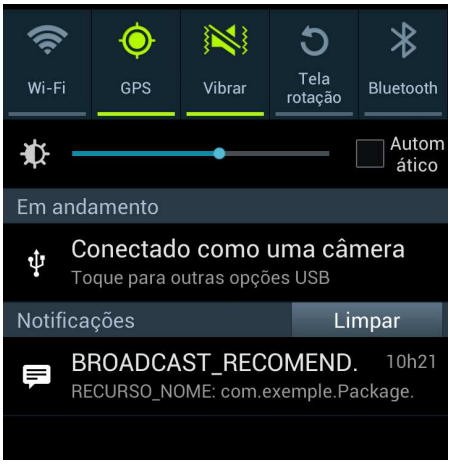

Figura 34. Notificação gerada ao receber recomendação de um recurso.

A figura 35 demonstra um percurso realizado (linha em cor vermelha) próximo a três locais que havia registro de uso de um *software* (incluídos manualmente na ontologia e no mapa). Aproximadamente dentro dos círculos em cor verde foram os locais em que ocorreram as recomendações.

As recomendações, mostradas na figura 35, ocorreram sequencialmente enquanto percorria a área próxima, sendo muitas destas recomendações contendo o mesmo percentual de confiabilidade já apresentado a poucos instantes. Uma melhoria futura para um algoritmo de recomendação seria a verificação e não exibição de uma recomendação contendo o mesmo percentual de confiabilidade a ser apresentado.

Figura 35. Visualização das áreas que ocorreu recomendação durante um percurso.

A verificação da recomendação mostrou-se coerente com os objetivos do trabalho e sendo a prova final para identificar se ocorreram problemas em todas as partes anteriores que coletam e armazenam as informações de contexto lógico estabelecidas no início deste trabalho.

6 CONCLUSÕES

Este capítulo apresenta o fechamento do presente trabalho com uma análise do estudo realizado e resultados alcançados. Além disso, com base no problema e na solução apresentada, evidenciam-se potenciais novos trabalhos futuros também aqui colocados.

6.1 Resultados alcançados

A arquitetura propunha-se a coletar e gerenciar contextos, que pela pesquisa bibliográfica realizada, ainda não havia sido realizada. Sua forma em particular em coletar contexto lógico de localização em ambientes internos através de sensores de aceleração e orientação, diferem-se dos demais trabalhos relacionados que utilizam a intensidade de rádio-frequência de antenas conhecidas, sejam elas de comunicação celular via GSM ou Wi-Fi. Quanto ao contexto de recomendação, ele também se difere de demais trabalhos relacionados, pois realiza inferência sobre dados armazenados em uma estrutura *Web Semântica*.

Durante as verificações do protótipo construído a partir a arquitetura apresentada, identificou-se a grande dificuldade de trabalhar com a sensibilidade dos sensores e a forma como a plataforma Android os gerencia, como por exemplo:

- Cada eixo do sensor de orientação e aceleração trabalha diferente, resultando em três equações diferentes para retirar a aceleração da gravidade em cada eixo de orientação do dispositivo móvel;
- Valor impreciso da aceleração apresentada quando realizado por um curto período de tempo.

Como consequência a estes fatores, o resultado para do agente de localização para a localização utilizando sensores é bem limitada e funciona apenas em ambiente controlado.

O uso de ontologias em dispositivo móvel mostrou-se satisfatório para registrar informações, porém, a inferência destes dados armazenados mostrou-se lenta, indicando limitações de *hardware* existentes e concluindo que deve ser usado com cautela quando necessita de respostas aparentemente imediatas para o usuário.

A maior contribuição deste trabalho foi criar e provar o funcionamento de uma arquitetura para gerenciar contextos utilizando tecnologias recentes para o universo de dispositivos móveis, como os sensores e a *Web* Semântica, até o momento pouco utilizado para identificar a utilização de programas e recursos pelos usuários destes dispositivos móveis.

Quem utilizará deste trabalho usufruirá de uma estrutura base para identificar contextos de localização e uso de recursos utilizados por um dispositivo móvel. Além de desfrutar de uma estrutura *Web* Semântica já montada e extraindo contexto de recomendação de uso de recursos a partir do histórico armazenado.

A arquitetura montada neste trabalho demonstra a coesão de cada agente desenvolvido, isso possibilitará que em trabalhos futuros, cada agente seja aperfeiçoado de forma independente. O funcionamento de uma arquitetura que integrasse estas tecnologias em um dispositivo móvel foi demonstrado com o protótipo desenvolvido.

6.2 Trabalhos futuros

Este trabalho armazena informações de contexto de localização do uso dos recursos instalados no dispositivo móvel, as quais já estão sendo utilizadas como base para o trabalho do Servidor de Contextos, que possibilita a integração com outras informações de contexto que podem ser coletadas, mas ainda podem servir como base para demais trabalhos futuros como:

- Desenvolver outros algoritmos de recomendação de uso de recursos;
- Aperfeiçoado do agente de localização utilizando sensores;
- Uma iteração entre dispositivos próximos compartilhando informações de contexto a fim de encontrar outros dispositivos móveis nas proximidades os quais tem um perfil parecido.
- Desenvolver uma ontologia dinâmica que possibilite armazenar, não apenas “o que” e “onde” foi utilizado, mas também a forma ou conteúdo informado pelo usuário em cada recurso.
- a criação de diversos outros sistemas, que possam tirar proveito da localização em ambientes internos disponibilizada pelo agente de localização: como registro de presença em sala de aula, cobrança de estacionamento, etc.

A arquitetura deste trabalho elaborada em componentes possibilita a utilização de cada um dos componentes para serem utilizados em outros trabalhos ou podem ser substituídos por outro componente de mesmo objetivo, porém mais evoluído. Por exemplo: seria possível substituir o agente de localização e agente de recomendação por outros agentes que aprimorem o atual algoritmo existente nestes agentes, respeitando apenas os contratos de uso estabelecidos, isto é, respeitando as informações que necessitam e as que fornecem, para assim manter em funcionamento todos os agentes que se utilizem deles, e como consequência o funcionamento de todo o protótipo desenvolvido.

REFERÊNCIAS

AQUIN, M.; NIKOLOV, A.; MOTTA, E. *Building SPARQL-Enabled Applications with Android Devices*. The 10th International Semantic *Web* Conference in Koblenz, Germany. 2011.

ABOWD, G.D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. *Towards a Better Understanding of Context and Context-Awareness*. Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. Springer-Verlag: Karlsruhe, Germany. 1999.

ASHBROOK, D; STARNER, T . *Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users* . Journal Personal and Ubiquitous Computing, Springer-Verlag, London. 2003.

BELLIFEMINE, F.; CAIER, G.; GREENWOOD, D. *Developing Multi-Agent Sytems with JADE*. ISBN 978-0-470-05747-6. Jonh Wiley & Sons, Ltd. 2007.

BERNERS-LEE, J. H. T.; LASSILA, O. *The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities*. Scientific American, 2001.

BRENNER, W; ZARNEKOW, R; WITTIG, H. *Inteligent Software Agents*. ISBN 3540634118. Springer. 1998.

CHU, D.; KANSAL, A.; LIU, J.; ZHAO, F. *Mobile Apps: It's Time to Move Up to CondOS*. Microsoft Research Redmond, Microsoft Research Asia. 2012.

CHEESMAN, J.; DANIELS, J . *UML Components : A Simple Process for Specifying Component-Based Software* . ISBN 0-201-70851-5 . Addison-Wesley . 2001.

DELOACH, S. A.; Wood, M. *Developing Muiltiagent Systems with agentTool*. In: Proceedings of Lecture Notes in Artificial Intelligence. Springer – Verlag. Berling, 2001.

DICKINSON, I. *Jena Ontology API*. 2009. Disponível em: http://jena.sourceforge.net/ontology/. Acesso em: mar. 2012.

FONOU, J. D.; HUISMAN, M. *Combining Ontology Development Methodologies and Semantic Web Platforms for E-government Domain Ontology Development*. International Journal of *Web* & Semantic Technology (IJWesT) Vol.2, 2011.

GÓMEZ, A. P.; FERNÁNDEZ, M. L.; CORCHO, O. *Ontological Engeneering*. ISBN 1-85233-551-3. Springer-Verlag London Limited, 2004.

GONZÁLEZ, D. S. *Un análisis sobre aplicaciones distribuidas dependientes del contexto* . Universidad Del País Vasco , Euskal Herriko Unibertsitatea . San Sebastián. 2009.

GRUBER, Tom. *Ontology*. Disponível em: <http://tomgruber.org/writing/ontology- definition-2007.htm>. Acesso em: fev. 2012.

GUENDA, L.; BRÁS, L.; OLIVEIRA, M.; CARVALHO, N. B.. *Indoor/Outdoor Management System Compliant with Google Maps and Android® OS.* IEEE EUROCON International

Conference on Computer as a Tool. 2011.

HAUSTEIN, S.; SEIGEL, J. *KSOAP 2 API*. Disponível em: http://ksoap2.sourceforge.net/doc/api/. Acesso em: jan. 2012.

JAYARAMAN, Prem; ZASLAVSKY, Arkady; DELSING, Jerker. *Cost Efficient Data Collection of Sensory Originated Data using Context-Aware Mobile Devices*. Monash University; Lulea University of Technology. 2008.

KOMATINENI, S; MACLEAN, D. *Pro Android 4*. ISBN 81430239307. Apress. 2012.

KLYNE, G.; CARROLL, J.J. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. 2004. Disponível em: http://www.w3.org/TR/rdf-concepts/. Acesso em jan. 2012.

LIMA, JÚNIO CÉSAR; CARVALHO, CEDRIC L. *Ontologias - OWL (Web Ontology Language)*. Relatório técnico RT-INF_004-05, Instituto de Informática, Universidade Federal de Goiás. 2005.

MCGUINNESS, D.L.; HARMELEN, F.V.. *OWL Web Ontology Language Overview*. 2004. Disponível em: http://www.w3.org/TR/owl-features/. Acesso em jan. 2012.

MIGUENS, ALTINEU PIRES. *Navegação: A ciência e a arte*. v. 1 – Navegação costeira, estimada e em águas restritas. 1993. Disponível em: <https://www.mar.mil.br/dhn/bhmn /download/cap1.pdf>. Acesso em: dez. 2012.

MOREIRA, A.; SANTOS, M. Y. *From GPS tracks to context: Inference of high-level context information through spatial clustering*. International Conference And Exhibition On Geographic Information. Estoril. ISBN 972-97367-5-8. 2005.

PADGHAM, L.; THANGARAJAH, J.; PARESH, P. *Prometheus Design Tool Version 2.5 - User Manual*. RMIT University. Disponível em: http://www.cs.rmit.edu.au/agents/pdt/docs/PDT-Manual.pdf. Acesso em: fev. 2012.

PRUD'HOMMEAUX, E.; SEABORNE, A. *SPARQL Query Language for RDF*, W3C, 2006. Disponível em: <http://www.w3.org/TR/2006/WD-rdf-sparql-query-20060220/>. Acesso em fev. 2012.

RAUTENBERG, S.; TODESCO, J. L.; STEIL, A. V.; GAUTHIER, F. A. O.; *Uma Metodologia para o Desenvolvimento de Ontologias*. Revista Ciências Exatas e Naturais, Vol.10 n 2. 2008.

SANTOS, P. L. V. A. da C.; ALVES, R. C. V. *Metadados e Web Semântica para estruturação da web 2.0 e web 3.0*. Revista de Ciência da Informação, v. 10, n. 6, 2009.

SHADBOLD, N.; HALL, W.; BERNERS-LEE, T. *The Semantic Web Revisited*. In: IEEE Intelligent Systems, 2006.

SILVA, EDNA LÚCIA DA ; MENEZES, ESTERA MUSZKAT. Metodologia da pesquisa e elaboração de dissertação. 4. ed. rev. atual. Florianópolis: Universidade Federal de Santa Catarina, 2005.

TECUCI, G. *Building Intelligent Agents*. ISBN 0126851255. Academic Press. 1998.

VITERBO, José; SACRAMENTO, Vagner; ROCHA, Ricardo; ENDLER, Markus. *MoCA: Uma Arquitetura para o Desenvolvimento de Aplicações Sensíveis ao Contexto para Dispositivos Móveis*. Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ). 2006.

WONGTHONGTHA, P; CHANG, E; DILLON, T; SOMMERVILLE, I. *Development of a Software Engineering Ontology for Multi-site Software Development.* IEEE Transactions on Knowledge and Data Engineering, Manuscript id. 2009. Disponível em: <http://www.cs.st-andrews.ac.uk/~ifs/Research/Publications/Papers-PDF/2005-09/TKDE-Ponpit-2009.pdf>. Acesso em: jan. 2012.

LONGLEY, PAUL A.; GOODCHILD, MICHAEL F.; MAGUIRE, DAVID J.; RHIND, DAVID W. *Geographic Information Systems and Science*. Brithish Library. ISBN 0-470-87000-1 (HB). 2005.