Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Um modelo para o gerenciamento de informações de contexto baseado em ontologias

Ana Helena Ozaki Rivera Castillo

Brasília
2012

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Um modelo para o gerenciamento de informações de contexto baseado em ontologias

Ana Helena Ozaki Rivera Castillo

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Carla Denise Castanho

Coorientador
Prof. Dr. Ricardo Pezzuol Jacobi

Brasília
2012

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programa de Pós-Graduação em Informática

Coordenadora: Prof.ª Dr.ª Mylene Christine Queiroz de Farias

Banca examinadora composta por:

Prof.ª Dr.ª Carla Denise Castanho (Orientadora) — CIC/UnB
Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB
Prof.ª Dr.ª Célia Ghedini Ralha — CIC/UnB
Prof. Dr. Mario Dantas — INE/UFSC

**CIP — Catalogação Internacional na Publicação**

Castillo, Ana Helena Ozaki Rivera.

Um modelo para o gerenciamento de informações de contexto baseado em ontologias / Ana Helena Ozaki Rivera Castillo. Brasília : UnB, 2012. 74 p. : il. ; 29,5 cm.

Tese (Mestrado) — Universidade de Brasília, Brasília, 2012.

1. Ambientes inteligentes, 2. Ontologia, 3. Sensibilidade ao Contexto, 4. *Middleware*

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Um modelo para o gerenciamento de informações de contexto baseado em ontologias

Ana Helena Ozaki Rivera Castillo

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Carla Denise Castanho (Orientadora)
CIC/UnB

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
CIC/UnB

Prof. Dr. Mario Dantas
INE/UFSC

Prof.ª Dr.ª Mylene Christine Queiroz de Farias
Coordenadora do Programa de Pós-Graduação em Informática

Brasília, 04 de julho de 2012

# Dedicatória

Dedico este trabalho aos meus avós maternos, que lutaram muito para que seus filhos e netos pudessem estudar.

# Agradecimentos

Este trabalho contou com o apoio de várias pessoas para ser concluído. Agradeço aos meus pais e minhas irmãs, por terem me incentivado e por apoiarem a minha decisão de seguir uma carreira acadêmica. Agradeço também ao Bruno (futuro Mestre Jedi) por todo o amor e cuidados, tornando a minha vida mais feliz.

Agradeço à banca, composta pela professora Célia Ghedini Ralha e pelo professor Mario Dantas, e aos meus orientadores, professora Carla Castanho e professor Ricardo Jacobi, por estarem do meu lado nos momentos em que precisei. Em especial, agradeço ao Fabricio, por ter contribuído inúmeras vezes durante todo o projeto e por ter conduzido as reuniões do UnBiquitous, nosso grupo de pesquisa, para que pudéssemos fazer de cada trabalho uma obra muito maior, somando o esforço dos alunos.

Gostaria de agradecer também o apoio do professor Jacir Bordim e dos colegas do laboratório ComNet. Também não poderia deixar de mencionar o professor Homero Piccolo, por ter me dado a oportunidade de trabalhar durante o mestrado em projetos para a disciplina de Introdução à Ciência da Computação, formada por 15 turmas.

# Resumo

Um *smart space* pode ser descrito como um ambiente que integra dispositivos, sensores e redes de comunicação para prover um ambiente computacional coeso e adaptável às necessidades dos usuários. Aplicações sensíveis ao contexto determinam a forma como o ambiente atua em prol do usuário, utilizando *middlewares* como camada de abstração para interagir com o ambiente e coletar informações relevantes para o reconhecimento do contexto.

Dentre as formas de representar contexto, as ontologias têm sido utilizadas devido à possibilidade de compartilhar, reusar e modelar a semântica do contexto, independente de linguagem de programação, sistema operacional e/ou *middleware* utilizado. Deste modo, este trabalho trata do problema de evoluir ontologias em *smart spaces*, com foco no compartilhamento de informações entre dispositivos e suporte do *middleware* a aplicações sensíveis ao contexto. Partindo da definição de componentes básicos necessários para o gerenciamento do contexto em um *smart space*, é apresentado um modelo que suporta a evolução de ontologias contendo informações de contexto. Este modelo foi implementado e integrado ao *middleware uOS*, desenvolvido na Universidade de Brasília e voltado para a adaptabilidade de serviços em *smart spaces*.

Para fins de avaliação, foi criado um conjunto de casos de uso de acordo com os componentes do modelo para gerenciamento de contexto apresentado neste trabalho. Além disso, uma avaliação quantitativa foi realizada com o objetivo de estimar o *overhead* introduzido pela implementação do modelo no *middleware uOS*. Os experimentos mostraram que consultas à uma ontologia com aproximadamente 100 axiomas acrescentaram um *overhead* médio de 3 ms, o que representa 8% do tempo máximo em que o ser humano não percebe que houve atraso [43]. Com relação à inserção de instâncias na ontologia, foi observado um crescimento de tempo linear em relação ao número de instâncias inseridas.

**Palavras-chave:** Ambientes inteligentes, Ontologia, Sensibilidade ao Contexto, *Middleware*

# Abstract

A smart space can be described as an environment that integrates devices, sensors and networks in order to provide a cohesive and adaptable computational environment. Context aware applications determine how a smart space acts on behalf of users, using middlewares as an abstraction layer to interact with the environment and to collect relevant information for context recognition.

Among the ways of representing context, ontologies have been used due to the possibilities of sharing, reusing and modelling context semantics, independently of the programming language, operating system and/or middleware used. Thus, this work addresses the problem of evolving ontologies in smart spaces, with focus on information sharing and middleware support to ontologies changes. Starting from the definition of basic components needed for context management in a smart space, we propose a model that supports the evolution of ontologies containing context information. The present work was implemented and integrated into the uOS middleware, developed at the University of Brasília and conceived with the objective of supporting service adaptability in smart spaces.

For evaluation purposes, it was created a set of use cases according to the components of the model for context management presented in this work. In addition, a quantitative evaluation was performed in order to estimate the overhead introduced by the implementation of the model in the uOS middleware. The experiments showed that queries to an ontology with approximately 100 axioms increased an average overhead of 3 ms, which represents 8% of the time limit where human beings do not realize that there was a delay [43]. Regarding the insertion of instances, it was observed a linear increase over time related to the number of inserted instances.

**Keywords:** Smart Spaces, Ontology, Context Awareness, Middleware

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

A Computação Ubíqua (*UbiComp*) refere-se às tecnologias que se encontram misturadas ao cotidiano das pessoas de maneira natural e transparente, capazes de atuar de forma integrada e inteligente em prol do usuário. Sensores e dispositivos de controle estão cada vez mais imersos no ambiente físico que nos rodeia [51]. Existe atualmente uma profusão de dispositivos móveis e inteligentes com múltiplas utilidades, capazes de acessar serviços locais e remotos. Estão sendo fabricados aparelhos eletrônicos cada vez menores, mais baratos e que operam com maior confiabilidade e menos energia.

Neste cenário, um conceito fundamental é o de *smart space*, o qual tem como foco o uso de dispositivos, sensores e redes de comunicação para fornecer um ambiente inteligente e sensível ao contexto. Pode-se citar como exemplo uma sala de reuniões que identifica a dinâmica do ambiente e atua facilitando as tarefas dos usuários [60]. A variedade e a heterogeneidade de serviços providos por um *smart space* conferem maior complexidade ao desenvolvimento de aplicações que acessam esses serviços. Para prover segurança, escalabilidade, tolerância a falhas e comunicação entre os recursos disponíveis, faz-se necessário o uso de um *middleware* que forneça uma camada de abstração para os dispositivos distribuídos em uma rede [5].

*Middlewares* em *smart spaces* devem interagir com o ambiente, de modo a coletar informações relevantes para o reconhecimento de contexto e o monitoramento de sua evolução [52]. As aplicações podem, por exemplo, precisar saber quais recursos estão disponíveis no ambiente e que serviços esses recursos fornecem. Propagar as informações de contexto para o nível da aplicação favorece o seu desenvolvimento, uma vez que o *middleware* encarrega-se de desafios básicos relacionados à construção de um *smart space*. Em [16] são descritas algumas funcionalidades que podem ser providas por *middlewares* para dar suporte a aplicações sensíveis ao contexto, tais como: (1) entrega de informações de acordo com o modelo *Publish-Subscribe*; (2) consulta a informações de contexto; (3) processamento das informações, de baixo nível para o nível da aplicação; (4) síntese do contexto, que é a fusão de várias informações; e (5) descoberta e gerenciamento de serviços, que permite que as aplicações localizem e obtenham o conjunto de serviços disponíveis. Com base nessas funcionalidades, diversas implementações de *middlewares* foram propostas a fim de prover suporte a aplicações sensíveis ao contexto, como em [53, 4, 56, 59, 67, 9].

Além dos aspectos relacionados as tarefas de obtenção, processamento e compartilhamento das informações de contexto, alguns *middlewares* encarregam-se de armazenar

dados coletados do ambiente [53, 13, 73, 49]. Armazenar estes dados possibilita que o conhecimento adquirido por cada tipo de serviço seja compartilhado e reutilizado quando necessário pelas aplicações. Tais informações precisam ser organizadas de acordo com alguma forma de representação para que sirvam de bases de conhecimento. Existem diversas formas de representar o contexto, como por exemplo: valor-chave, esquema de marcação, forma gráfica (utilizando UML (*Unified Modelling Language*) por exemplo), baseada em ontologia, entre outras [62]. Dentre as formas de representação citadas, as ontologias destacam-se por permitirem a utilização de mecanismos de inferência, dado o formalismo lógico que existe nas ontologias [37].

Uma das razões para a utilização de ontologias em *smart spaces* é a possibilidade de que as informações sejam descritas de acordo com o nível de detalhe requerido pelas aplicações. Em um *smart space*, o contexto dos recursos disponíveis evolui dinamicamente e, assim, as alterações precisam ser refletidas na ontologia. Entretanto, o desenvolvimento de aplicações ocorre frequentemente de forma independente, tornando mais complexa a tarefa de compartilhamento e reúso das informações. Além disso, o fato das aplicações alterarem a ontologia possibilita o surgimento de inconsistências. Portanto, faz-se necessário que as alterações sejam gerenciadas de acordo com um processo evolutivo. Conforme relatado em [61], a evolução da ontologia significa a adaptação temporal de uma ontologia devido a mudanças e a propagação consistente dessas alterações para os artefatos relevantes.

Considerando a mobilidade dos usuários e seus dispositivos, temos que o uso de ontologias para representar o contexto e a criação de mecanismos para a sua evolução podem melhorar significativamente a habilidade de um *middleware* dar suporte às aplicações sensíveis ao contexto. No entanto, soluções encontradas na literatura que utilizam ontologias para a modelagem de contexto tratam o gerenciamento das informações de maneiras distintas. Alguns modelos possuem foco na representação de atributos das informações, consultas e modelagem de eventos [54, 34]. Outros trabalhos têm por objetivo estabelecer a melhor forma de organizar o contexto, visando encontrar um denominador comum para a criação de um padrão [11]. Em [22] os conceitos são divididos em camadas, dependendo do nível de generalidade (ou especificidade) da informações. Em [53] é definido um servidor de ontologias, que controla as alterações e evita que informações inconsistentes sejam adicionadas na base de conhecimento. Contudo, as regras para a verificação de consistência e a estratégia para controle das alterações no servidor não são especificadas no trabalho. Embora muitos dos trabalhos analisados adotem ontologias para representar o contexto, a definição de mecanismos para o gerenciamento e evolução de ontologias em ambientes inteligentes permanecem como foco de pesquisa.

## 1.1 Problema e Objetivos

Em um *smart space*, várias aplicações sensíveis ao contexto podem atuar de forma descentralizada. Se uma aplicação, por exemplo, tenta remover uma informação da base de conhecimento do dispositivo, é preciso verificar se esta informação não foi fornecida por outra aplicação, que, neste caso, estaria sendo prejudicada. Além disso, como cada dispositivo pode agregar um conjunto de informações, deve-se permitir que essas informações sejam consultadas e compartilhadas por dispositivos remotos, de modo a fornecer às aplicações uma visão acurada do ambiente.

Dessa forma, abordamos neste trabalho o problema da evolução de ontologias em *smart spaces*, com foco no compartilhamento de informações de contexto entre dispositivos do ambiente e controle sobre as alterações na ontologia. Mais especificamente, os objetivos deste trabalho são:

- prover suporte a aplicações sensíveis ao contexto;
- desenvolver uma solução que permita o reúso e o compartilhamento das informações de contexto entre as aplicações.

Para isto, é apresentado um modelo baseado em ontologias que visa gerenciar e prover suporte à evolução de informações de contexto. O modelo foi implementado e integrado ao *middleware uOS*, desenvolvido na Universidade de Brasília [8]. O *uOS* foi projetado para permitir a comunicação entre dispositivos computacionais em um *smart space*. Tem como foco a adaptabilidade de serviços, definida em [9] como a decomposição dos recursos presentes no ambiente em serviços, de tal maneira que as aplicações (ou o usuário) possam escolher de forma transparente qual recurso melhor atende às suas necessidades. Dessa forma, o trabalho apresentado complementa o *middleware* adotado no sentido de gerenciar informações de contexto obtidas por meio dos serviços providos, permitindo o compartilhamento de tais informações pelas aplicações sensíveis ao contexto.

Neste trabalho será apresentado:

1. *Um modelo para gerenciamento do contexto baseado em ontologias:* partindo da definição de componentes básicos necessários para o gerenciamento do contexto em um *smart space*, é apresentado um modelo que suporta a evolução de ontologias contendo informações de contexto. Para isso, o modelo: (i) provê uma API que permite às aplicações inserirem dados na ontologia; (ii) realiza um processo de verificação de consistência sobre as alterações solicitadas; e (iii) permite a consulta e compartilhamento de informações entre as aplicações e dispositivos de um *smart space*.
2. *Implementação do modelo no middleware uOS:* o modelo proposto foi implementado e integrado ao *middleware uOS*. A implementação permitiu uma avaliação prática de mecanismos para a evolução de ontologias com o objetivo de prover suporte à dinâmica das informações de contexto. Estes mecanismos incluem regras para verificação de consistência em alterações na ontologia local e a implementação de um *driver* da ontologia, desenvolvido para permitir consultas a ontologias presentes em dispositivos remotos.

Para fins de avaliação, um conjunto de casos de uso foi modelado e um protótipo foi implementado de acordo com os componentes deste trabalho. Além disso, uma avaliação quantitativa foi realizada com o objetivo de estimar o *overhead* introduzido pela implementação do modelo no *middleware uOS*. Os experimentos mostraram que o tempo de processamento de consultas à uma ontologia com aproximadamente 100 axiomas acrescentou um *overhead* médio de 3 ms, o que representa um impacto 8% em relação à 50 ms, considerado o limite de tempo em que o ser humano não percebe que houve atraso [43].

## 1.2 Organização do trabalho

O restante deste trabalho é organizado da seguinte forma:

- no Capítulo 2 são introduzidos conceitos básicos relativos a sensibilidade ao contexto, descreve o fluxo das informações de contexto e apresenta os principais desafios relacionados a modelagem de aplicações sensíveis ao contexto;
- no Capítulo 3 são apresentados conceitos relacionados a ontologias considerados relevantes para este trabalho; com alguns trabalhos relacionados que utilizam ontologias para representação e gerenciamento das informações de contexto;
- no Capítulo 4 são descritos alguns serviços implementados por *middlewares* para prover suporte às aplicações sensíveis ao contexto e é apresentado o *middleware uOS*, adotado na implementação deste trabalho;
- o modelo proposto e os principais pontos da implementação no *middleware uOS* são discutidos no Capítulo 5;
- no Capítulo 6 são apresentados casos de uso baseados em aplicações e cenários de utilização da proposta, e uma avaliação quantitativa com dados referentes ao tempo de processamento de operações na ontologia em um sistema *multi-thread*;
- as considerações finais são apresentadas no Capítulo 7.

# Capítulo 2

# Sensibilidade ao Contexto

A Computação Ubíqua pode ser descrita como um método para melhorar o uso de dispositivos computacionais, deixando-os disponíveis e dispersos no ambiente de forma eficientemente "invisível" aos usuários [68]. O conceito de Computação Ubíqua relaciona-se com a palavra "ubíqua", cujo significado diz respeito a estar presente em todo lugar ao mesmo tempo, denotando a idéia de onipresença. Conforme descrito em [69], as boas ferramentas são aquelas em que seu uso é tão natural que torna-se imperceptível no dia a dia, daí a noção de invisibilidade.

> *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."* Mark Weiser

Computação Ubíqua e Computação Pervasiva são termos muitas vezes utilizados de forma intercambiável [57]. A Computação Pervasiva tem como objetivo capturar o contexto do ambiente e construir modelos computacionais que alteram seu comportamento de acordo com o contexto, ajustando-se dinamicamente às necessidades do usuário e sem a interferência deste. Já a Computação Ubíqua combina a computação pervasiva com os avanços da computação móvel para prover ao usuário um contexto computacional global e integrado [72].

Alguns autores definem contexto como sendo o ambiente do usuário [7], enquanto outros o definem como o ambiente da aplicação [66]. Este trabalho utiliza a definição de contexto dada por Dey [17], por ser uma definição capaz de englobar tanto as aplicações quanto os usuários.

**Definição:** ***Contexto*** *é qualquer informação que possa ser usada para caracterizar a situação de uma entidade, sendo que essa entidade pode ser uma pessoa, lugar ou objeto considerado relevante para a interação entre o usuário e a aplicação, incluindo o próprio usuário e a aplicação* [17].

**Definição:** *Um sistema é* ***sensível ao contexto*** *(context aware) se utiliza o contexto para prover informações e serviços relevantes ao usuário, onde a "relevância" depende das tarefas dos usuários* [15].

Destas definições, entende-se que a sensibilidade ao contexto é a habilidade de um sistema ser sensível ao que está a sua volta, reagindo conforme o contexto evolui de um estado para outro. Também podemos observar que a "relevância" das informações e serviços depende das tarefas dos usuários, a qual é descoberta utilizando informações de contexto. Alguns tipos de informação são comumente usados para determinar o contexto, como localização, temperatura, velocidade, distância até os recursos disponíveis, entre vários outros [38].

## 2.1 Informações de Contexto

As informações de contexto constituem um conjunto de informações utilizado para a definição do contexto. Em um sistema contendo um *middleware* e várias aplicações, o processo de tratamento de informações basicamente segue um fluxo que pode ser descrito pelas seguintes etapas:

1. obtenção da informação;
2. representação da informação;
3. processamento da informação;
4. entrega da informação.

A *obtenção* da informação refere-se aos meios pelos quais a informação de contexto é capturada pelo sistema. Em seguida, a *representação* da informação busca organizar a informação de acordo com os requisitos das aplicações. O *processamento* da informação utiliza o conjunto de informações obtido e representado nas etapas anteriores para extrair informações não explícitas que possam ser úteis às aplicações. Por fim, a *entrega* da informação é responsável por fornecer as informações de contexto relativas a cada aplicação em execução no *middleware*. As próximas subseções detalham cada etapa envolvida.

### 2.1.1 Obtenção da informação

Dependendo dos requisitos da aplicação, diferentes métodos de obtenção de informação podem ser utilizados para a definição do contexto. Conforme a classificação de Mostefaoui et al., as informações de contexto podem ser [44]:

1. Sentidas - por meio de sensores instalados no ambiente que captam informações de temperatura, umidade, pressão atmosférica, localização, orientação, velocidade, etc;
2. Derivadas - de outras informações, como por exemplo a duração de uma reunião, a qual pode ser derivada dos horários de início e término da reunião;
3. Providas - pelo próprio usuário, por meio de cadastros com opções de preferência, etc.

**Exemplo:** Como um exemplo, suponha que uma determinada aplicação necessite saber se existem pessoas presentes no laboratório Y. Considere também que nesse laboratório Y existe um sensor capaz de detectar a presença de pessoas no ambiente. Então, nessa situação, a informação de que Y é, por exemplo, um laboratório de anatomia seria uma informação *provida*, dado que o sensor de presença por si só não consegue identificar que tipo de ambiente ele se encontra. A informação de que existem ou não pessoas no ambiente seria uma informação *sentida*. Por fim, a informação de que existem pessoas no laboratório de anatomia seria uma informação *derivada* das duas informações anteriores.

Algumas características consideradas importantes para a obtenção de informações em ambientes ubíquos são descritas em [18]. A primeira define que os sensores precisam ser capazes de se auto-calibrarem, adaptando-se à dinâmica do ambiente. Para atingir este objetivo, devem comunicar seus estados e área de cobertura, tornando necessária a utilização de uma infra-estrutura de rede. A terceira característica é chamada *distributed computing* e visa promover o melhor uso dos recursos de processamento disponíveis no ambiente. As demais características mencionadas em [18] estão especificamente relacionadas aos tipos de sensores, como por exemplo sensores móveis e embarcados, capturando dados de áudio, vídeo, etc. O uso de vários tipos de sensores visa coletar um volume de dados que indique de maneira confiável informações precisas e detalhadas às aplicações sensíveis ao contexto.

Independente da forma de obtenção da informação, cada informação de contexto possui um conjunto de atributos. Conforme descrito em [3], os atributos básicos são tipo e valor (e. g., Volume e 10 dB), em que o valor depende da unidade utilizada. Porém, existem outros atributos que podem ser utilizados para facilitar o gerenciamento das informações de contexto como *Timestamp*, origem da informação, confiabilidade, período de validade da informação, entre outros.

### 2.1.2 Representação da informação

Para organizar e armazenar as informações de contexto é preciso definir uma forma de representação. Segundo [62], são identificados na literatura basicamente seis tipos de modelos utilizados para representar informações de contexto:

1. Valor-chave: é o modelo mais simples para estruturação dos dados. Neste modelo cada informação é associada a uma chave (e.g., Temperatura) e um valor (e.g., 24 graus);
2. Esquema de marcação: utiliza uma estrutura de dados hierárquica composta com *tags* de marcação para atributos e conteúdo. Perfis tipicamente representam esse tipo de modelo, como em [65];
3. Gráfico: a modelagem de contexto gráfica pode ser implementada utilizando linguagens gráficas como a UML;
4. Orientação a objeto: nessa abordagem vários objetos são utilizados para representar o contexto. Ao utilizar a orientação a objeto, os detalhes de processamento são encapsulados e o acesso ao contexto é feito via interfaces bem definidas;

5. Baseado em lógica: este tipo de modelo possui um alto grau de formalismo. Fatos, expressões e regras são utilizados para modelar o contexto. O processo de inferência permite a derivação de novos fatos com base em regras existentes;

6. Baseado em ontologia: nessa abordagem os conceitos, os subconceitos e os fatos relevantes são descritos de maneira uniforme, geralmente seguindo a sintaxe de uma linguagem para ontologias.

Dentre os modelos citados acima, as ontologias destacam-se por oferecerem características como simplicidade, flexibilidade, generalidade e expressividade semântica [37]. A simplicidade é um requisito que define que as expressões e relacionamentos devem ser tão simples quanto possível, de modo a facilitar o trabalho dos desenvolvedores. A flexibilidade refere-se à capacidade da ontologia suportar a inclusão de novos elementos do contexto, assim como seus relacionamentos. A terceira característica, generalidade, indica que o uso de ontologias não é limitado a um certo domínio, podendo suportar diferentes tipos de contexto. Por fim, a expressividade diz respeito a capacidade de descrever contextos com níveis de detalhes arbitrários [3]. Devido a essas características vários trabalhos encontrados na literatura utilizam ontologias para representar as informações de contexto, como em [36, 22, 73, 67, 40].

### 2.1.3 Processamento da informação

Esta etapa tem como objetivo processar as informações de contexto de modo a torná-las úteis às aplicações, adaptando as informações para os níveis de abstração desejados. Além disso, visa inferir novas informações de contexto relevantes a partir das informações obtidas. Conforme relatado em [38], as aplicações são comumente modeladas a partir de regras *if-then*. Ao utilizar formalismos lógicos é possível também realizar inferências sobre as informações de contexto, derivando informações com maior nível de abstração. Porém, um sistema modelado a partir de regras determinísticas torna-se rígido, uma pequena diferença em relação a regra pode fazer com que ela não seja executada quando deveria. Uma outra forma de processar informações de contexto é por meio de redes neuronais, as quais permitem o aprendizado de padrões de comportamento. Redes neuronais artificiais podem ser aplicadas para encontrar relacionamentos probabilísticos entre uma situação e a ação a ser tomada. Contudo, encontrar tais relacionamentos pode requerer um grande volume de dados para treinamento e o processo de depuração é complexo.

As técnicas utilizadas para o processamento das informações de contexto têm por objetivo apoiar a tomada de decisão das aplicações e não são excludentes. Existem trabalhos que combinam essas técnicas para o desenvolvimento de aplicações robustas, que tomam decisões com base em vários fatores. Em [47], por exemplo, é apresentado um modelo de aplicação que utiliza sensores de luz e temperatura, um módulo de inferência sobre as informações coletadas pelos sensores e um módulo de aprendizado de máquina para conhecer as preferências musicais dos usuários. Com base em todas essas informações, a aplicação desenvolvida pelos autores escolhe músicas de acordo com os usuários presentes no ambiente.

### 2.1.4 Entrega da informação

O último passo do fluxo das informações de contexto é a entrega. Diante de contextos dinâmicos, é necessário um mecanismo de atualização das informações. Em muitos sistemas, a entrega da informação é feita por meio de eventos, conforme o modelo *Publish-Subscribe* [6]. Após ser processada, uma informação de contexto pode ou não gerar um evento. A geração de evento ocorre quando condições predefinidas pelas aplicações são satisfeitas.

Outra forma de realizar a entrega da informação é por meio de regras que limitam a propagação de informações de acordo com critérios como localização geográfica, perfis de usuário, etc [13]. Com relação ao modo de comunicação, a disseminação das informações de contexto pode ocorrer a partir de um nó central com maior capacidade de processamento ou entre nós de mesmo nível em uma hierarquia de dispositivos.

## 2.2 Desafios relacionados ao contexto

Existem alguns desafios inerentes ao gerenciamento do contexto a serem tratados pelos *middlewares* e pelas aplicações sensíveis ao contexto. Conforme descrito em [38], o contexto capturado nada mais é do que um *proxy* para a intenção do usuário. Sendo assim, as informações de contexto servem apenas como um indicador de como determinada aplicação deveria atuar, não sendo possível garantir que tais informações sejam suficientes para determinar a intenção do usuário. Nesta seção são apresentados alguns dos problemas encontrados na modelagem de contexto e suas possíveis soluções.

### 2.2.1 Contextos modelados de forma incompleta

Contexto é um conceito amplo que abrange vários fatores e, portanto, cabe aos projetistas definir quais parâmetros são relevantes para as aplicações [41]. Ao modelar uma aplicação, o projetista lista os possíveis contextos em que ele espera que os usuários se encontrem e constrói um modelo baseado nos contextos identificados. Entretanto, em um contexto fora do que foi modelado, a aplicação pode atuar de forma incorreta. A acurácia na detecção do contexto do usuário pode ser aumentada com o uso de composição de contextos. Além disso, ao encontrar ambiguidades, a aplicação pode oferecer ao usuário um meio interativo que permita a escolha da ação mais apropriada.

### 2.2.2 Grande volume de informações de contexto

O gerenciamento das informações de contexto pode se tornar muito complexo e oneroso devido a um grande volume de informações de contexto, especialmente quando se mantém um histórico destas informações. Para filtrar o grande volume de informações, técnicas de mineração de dados podem ser utilizadas. Adicionalmente, pode-se regular a periodicidade em que as informações de contexto são capturadas de acordo com a necessidade. Dependendo do volume de informações que alimentam a base de conhecimento da aplicação, convém que sejam adotadas políticas de exclusão das informações em *cache*.

### 2.2.3 Informações de contexto incorretas

A correta definição do contexto depende da confiabilidade das informações capturadas por sensores ou providas pelo usuário. Por exemplo, um sistema de localização que auxilia na refrigeração de um ambiente deve fornecer a localização precisa de um usuário que esteja próximo a parede e não na sala vizinha, de modo que o sistema refrigerador acionado seja o do ambiente em que o usuário se encontra. Como não se pode garantir que todas as informações de contexto tenham sido obtidas corretamente, uma solução seria admitir a existência desse tipo de falha e atribuir aos dados capturados por sensores atributos de confiabilidade. Ao ultrapassar um certo limiar a ação seria tomada pela aplicação automaticamente, caso contrário a possibilidade de escolha poderia ser dada ao usuário.

### 2.2.4 Riscos à privacidade

Para definir o contexto, o sistema pode abrigar informações sensíveis referentes aos usuários. A fim de proteger tais informações, técnicas de criptografia devem ser utilizadas. Além disso, durante a modelagem do sistema pode-se selecionar certos tipos de sensores em detrimento de outros, como por exemplo, o uso de sensores de presença com base em infravermelho no lugar de câmeras de vídeo. No sistema apresentado em [71] foram utilizados microfones ao invés de câmeras com o intuito de preservar a privacidade.

### 2.2.5 Compartilhamento

As informações de contexto podem ser obtidas por diversos tipos de sensores, cadastradas pelos usuários ou derivadas de outros dados. Tais informações podem ser utilizadas por aplicações executando de forma local ou remota, e portanto, precisam ser compartilhadas entre os dispositivos do ambiente. Devido a variedade das informações de contexto, o compartilhamento dos dados e a propagação de alterações representam um desafio para os sistemas ubíquos.

### 2.2.6 Forma de avaliação

Aplicações sensíveis ao contexto são difíceis de avaliar, pois dependem de informações geradas a partir do contexto do usuário, do ambiente ou da própria aplicação. Dependendo da modelagem da aplicação, seu comportamento pode estar baseado em diversos parâmetros, como preferências de usuário e outras informações específicas. Com tantos quesitos a serem analisados, as tarefas de simular contextos em laboratório, comparar os resultados obtidos com outros trabalhos e realizar testes quantitativamente tornam a forma de avaliação mais um desafio.

## 2.3 Considerações

Neste capítulo foram apresentadas algumas das principais características e formas de representar o contexto. Além disso, foi descrito o fluxo de informações de contexto, o qual inicia-se com a obtenção das informações, passa pela representação, processamento e, por fim, a entrega aos artefatos relevantes. Conforme visto, existem vários desafios a serem

observados durante a modelagem de uma aplicação sensível ao contexto. A aplicação deve ser capaz de prever múltiplas situações, tratar do volume de dados, atribuir grau de confiabilidade das informações, proteger os dados dos usuários, ser avaliada quanto à eficiência e satisfação dos usuários, entre outros. Deste modo, faz-se necessário o desenvolvimento de soluções que gerenciem as informações de contexto, encarregando-se de desafios comuns às aplicações.

# Capítulo 3

# Ontologias

O termo "ontologia" tem origem na Filosofia e está ligado ao estudo da natureza da existência. Nas Ciências da Computação e da Informação, esse termo refere-se à modelagem do conhecimento acerca de algum domínio, seja real ou virtual [39]. Este trabalho adota a seguinte definição de ontologia:

**Definição:** *Uma* ***ontologia*** *é uma especificação formal e explícita de uma conceitualização compartilhada acerca de um domínio de interesse* [21].

Entende-se o termo conceitualização como uma visão de algo que existe e que deseja-se representar. Quando existe um consenso acerca de determinada conceitualização, diz-se que ela é compartilhada. Também cabe ressaltar que o propósito de uma ontologia não é o de conceituar tudo o que existe, mas sim informações consideradas relevantes no contexto de um problema específico. Desse modo, uma ontologia pode ser vista como uma forma de representar o conhecimento referente a um domínio. Basicamente as ontologias descrevem:

- conceitos relevantes a um domínio;
- relacionamentos entre os conceitos;
- restrições sobre os elementos da ontologia;
- instâncias que correspondem a indivíduos em um domínio.

As ontologias podem abrigar qualquer tipo de conhecimento acerca de um domínio. Em um cenário ubíquo, dispositivos entram e saem de maneira dinâmica, conforme seus usuários deslocam-se de um ambiente para outro. Assim, dispositivos não expressamente projetados para trabalharem juntos precisam ser interoperáveis. Conforme relatado em [26], a interoperabilidade pode ser alcançada ao explicitar o conhecimento sobre as características, meios de acesso, e outras informações sobre os dispositivos em uma ontologia.

O uso de ontologias também permite o compartilhamento e o reúso do conhecimento em um ambiente aberto e distribuído [50]. Ao descrever o conhecimento acerca de um domínio em uma ontologia, obtém-se uma representação semântica das informações, as quais podem ser usadas para direcionar o comportamento de aplicações sensíveis ao contexto.

## 3.1 Evolução da Ontologia

Na medida em que novos conceitos, com suas relações e propriedades, são incorporados, alterados ou mesmo excluídos de um domínio, faz-se necessário que a ontologia associada seja adaptada a essas mudanças. Este trabalho adota a seguinte definição de evolução de ontologia:

**Definição:** *A **evolução da ontologia** é a adaptação temporal de uma ontologia para o surgimento de mudanças e a propagação dessas mudanças de forma consistente para os artefatos dependentes* [61].

Portanto, a evolução de uma ontologia envolve a manutenção da sua consistência após a mudança. Ao evoluir uma ontologia pode-se verificar a sua consistência **antes** ou **depois** da alteração [61]. Quando a verificação é feita **depois**, todas as mudanças podem ser verificadas de uma só vez. Porém, em caso de falha é preciso retornar a ontologia ao estado inicial. A vantagem de verificar a consistência **antes** da mudança é que não é preciso guardar o estado inicial da ontologia. No entanto, a verificação à priori requer a definição de precondições necessárias à manutenção da consistência, além das restrições de consistência, que devem ser definidas para ambos os casos.

Devido à complexidade estrutural das ontologias, suas alterações precisam ser gerenciadas a fim de manter a consistência após as mudanças. Conforme relatado em [35], o processo de evolução de uma ontologia pode ser dividido em cinco etapas:

1. requisição de alteração;
2. representação da alteração;
3. semântica da alteração;
4. verificação e implementação da alteração;
5. propagação da alteração.

A *requisição de alteração* especifica uma alteração a ser aplicada na ontologia. Esta requisição surge de mudanças de contexto que precisam ser refletidas na base de conhecimento. Na segunda etapa, *representação da alteração*, a requisição é representada de acordo com um formato predefinido, que serve para padronizar a forma como as alterações são implementadas na ontologia. Em seguida, os efeitos da alteração são avaliados quanto à *semântica*, possivelmente trazendo como consequência a dedução de novas mudanças. Na quarta etapa, *verificação e implementação da alteração*, a requisição é realizada na ontologia. Por fim, as alterações são *propagadas* aos artefatos que dependem destas informações.

### 3.1.1 Concisão x Consistência

Existem vários critérios para avaliar uma ontologia, dentre eles, pode-se citar a concisão e a consistência [64]. A concisão indica se a ontologia contém axiomas que não são úteis para modelar a semântica do contexto, podendo ser irrelevantes ou redundantes. Os axiomas são considerados irrelevantes quando não agregam informação ao domínio

modelado (e.g. uma ontologia de vinhos contendo informações sobre vida marinha). A concisão também serve para indicar se a ontologia contém informações redundantes [64], ou seja, se existem axiomas para informações que poderiam ser extraídas por meio de mecanismos de inferência. Os sistemas, em geral, limitam-se a controlar as informações quanto à redundância, uma vez que a decisão acerca da relevância de uma informação frequentemente necessita da análise de um especialista.

No âmbito da Lógica, diz-se que uma teoria é consistente quando não contém contradições [12]. Em [25] são definidas três noções de consistência:

1. Consistência **estrutural**: garante que a ontologia está de acordo com os construtores da linguagem de ontologia utilizada.
2. Consistência **lógica**: esta noção de consistência está relacionada à verificação de satisfatibilidade de cada conceito existente [2].
3. Consistência **definida pelo usuário**: esta definição abrange restrições que não estão relacionadas à estrutura ou ao formalismo lógico, mas sim ao contexto da aplicação.

Desse modo, entende-se que uma ontologia é consistente quando as restrições de consistência estabelecidas são respeitadas, sejam elas estruturais, lógicas ou definidas pelo usuário.

## 3.2 Linguagens para Ontologias

Uma das primeiras linguagens para ontologia utilizadas na *Web* foi a linguagem SHOE (*Simple HTML Ontology Extensions*) [31]. A SHOE parte do princípio de que as ontologias são altamente interligadas e sujeitas a alterações. Para dar suporte a essas necessidades, a linguagem oferece diretivas com o propósito de permitir a importação de outras ontologias, versionamento, compatibilidade de informações, entre outros [31].

Para suportar formalismos semânticos da lógica descritiva e mecanismos de inferência eficientes foram projetadas as linguagens OIL (*Ontology Interchange Language*) e DAML (*DARPA Agent Markup Language*), que combinadas formam a linguagem DAML+OIL [70]. Esta linguagem é precursora da OWL (*Web Ontology Language*), linguagem recomendada pela W3C (*World Wide Web Consortium*) para modelagem de ontologias desde 2004. A OWL baseia-se em conceitos de linguagens predecessoras, como a SHOE. Com relação a aspectos semânticos, a OWL é equivalente a uma lógica descritiva altamente expressiva, sendo bastante influenciada pela DAML+OIL. Sua sintaxe segue o formato de outras especificações da W3C, como RDF (*Resource Description Framework*) e RDFS (*Resource Description Framework Schema*). Sendo assim, a OWL é uma linguagem robusta, projetada para permitir a descrição semântica de um domínio segundo seus conceitos e propriedades [30].

A OWL API é uma API em Java para criação, manipulação e serialização de ontologias em OWL [29]. Esta API fornece uma camada de abstração na forma de acesso às ontologias, permitindo que os desenvolvedores foquem em aspectos de mais alto nível durante a implementação de suas aplicações.

### 3.2.1 Perfis OWL

Quanto maior o grau de expressividade da linguagem mais complexos tornam-se os mecanismos de inferência e as consultas. Para acomodar as necessidades das aplicações em termos de expressividade e performance foram criadas sublinguagens para a OWL. Segundo [42], a OWL inicialmente apresentava três sublinguagens:

- *OWL Lite*: voltada para aplicações que essencialmente precisam de uma classificação de hierarquia e restrições simples.
- *OWL DL (Description Logic)*: direcionada para aplicações que precisam do máximo de expressividade, porém, mantendo a decidibilidade e completude das implementações.
- *OWL Full*: voltada para o máximo de expressividade, com a liberdade de sintaxe da RDF, porém sem garantias computacionais.

Com a publicação da segunda versão da OWL foram apresentadas mais três sublinguagens [27]:

- *OWL 2 EL (Existential Language)*: voltada para aplicações com ontologias contendo um grande número de propriedades e/ou classes.
- *OWL 2 QL (Query Language)*: direcionada para aplicações que utilizam um grande número de instâncias, onde a tarefa mais importante é a resposta a consultas.
- *OWL 2 RL (Rule Language)*: voltada para aplicações que requerem consultas e mecanismos de inferência eficientes, sem perder muito poder de expressividade.

A OWL 2 trouxe novas funcionalidades, como restrições de cardinalidade qualificada e propriedades assimétricas, reflexivas e disjuntas. Além disso, manteve a compatibilidade com a versão anterior. Isto significa que as sublinguagens da OWL 1 continuam válidas na OWL 2 e que qualquer ontologia construída com base na primeira versão também continua válida na segunda versão. As sublinguagens da OWL 1 foram projetadas de forma incremental em termos de expressividade semântica. Ou seja, a OWL *Lite* está contida na OWL DL, que está contida na OWL *Full*. Apesar de ser mais restrita que a DL, a sublinguagem *Lite* ainda possui complexidade computacional exponencial, o que gerou a necessidade da criação de sublinguagens que possibilitassem implementações polinomiais. Assim, cada sublinguagem da OWL 2 forma um conjunto maximal de expressividade com a característica de manter a complexidade polinomial.

## 3.3 Lógica Descritiva

Basicamente, todo o formalismo lógico que existe nas ontologias é proveniente da Lógica Descritiva (*Description Logic - DL*). A Lógica Descritiva tem como objetivo representar o conhecimento sobre um domínio de modo estruturado e formal [2]. Caracteriza-se por permitir um alto grau de expressividade semântica, possibilitando a construção de conceitos e relacionamentos complexos. Utilizando Lógica Descritiva pode-se expressar operações de inverso, transitividade, inclusão de relacionamentos e restrições numéricas, como no exemplo abaixo.

**Exemplo:** Pode-se definir o conceito de "Uma mulher solteira e com mais de três filhos." em *DL* da seguinte forma:

$$Humano \sqcap Femea \sqcap \neg Casada \sqcap > 3(temFilho.Humano)$$

A modelagem de uma base de conhecimento envolve a especificação de dois componentes: *TBox* (*Terminological Axioms Box*) e *ABox* (*Assertions Box*) [2]. A *TBox* contém a terminologia associada ao domínio representado, com suas principais propriedades e conceitos. Enquanto que a *ABox* contém o conhecimento assertivo específico das instâncias do domínio modelado. O conhecimento assertivo de uma *ABox* pode ser de dois tipos: conceitual ou relacional. O conhecimento conceitual é o que define que um indivíduo $a$ é uma instância do conceito $C$. Por exemplo, *Garfield* é um *gato*. Já o relacional estabelece uma relação entre dois indivíduos $a$ e $b$. Por exemplo, *John* é dono do *Garfield*. Por ser mais específico, o conhecimento descrito pela *ABox* tende a estar mais sujeito a alterações do que o descrito na *TBox*.

Um dos relacionamentos mais básicos presentes em uma *TBox* é o que determina a especialização ou generalização de um conceito, mais conhecido como relacionamento "é um" (*is-a*). Por este relacionamento é possível estabelecer hierarquias de conceitos e propriedades.

**Exemplo:** Vamos supor que deseja-se inserir o seguinte axioma em uma determinada *TBox*: "gato é um felino", onde *felino* é um conceito mais amplo que *gato*. Em DL, tal axioma seria escrito da seguinte forma:

$$gato \sqsubseteq felino$$

Existindo o conceito de *gato*, pode-se incluir na *ABox* a assertiva que define *Garfield* como uma instância de *gato*. Assim, a partir da assertiva e do axioma pode-se inferir que *Garfield* é um *felino*.

## 3.4 Mecanismos de Inferência

A descrição do conhecimento a cerca de um domínio em uma ontologia com os formalismos da lógica descritiva permite a realização de certos tipos de inferência. Os mecanismos de inferência têm como objetivo obter informações que não estão explícitas na base de conhecimento.

Os principais mecanismos de inferência relacionados aos conceitos de uma *TBox* $T$ são [1]:

- Satisfatibilidade: Um conceito $C$ é satisfatível com respeito a $T$ se existe um modelo $I$ de $T$ tal que $C^I$ é não vazio, ou seja, se $I$ é um modelo de $C$.
- *Subsumption:* Um conceito $C$ é um subconjunto (*subsumes*) de um conceito $D$ com respeito a $T$ se $C^I \subseteq D^I$ para cada modelo $I$ de $T$.
- Equivalência: Dois conceitos $C$ e $D$ são equivalentes com respeito a $T$ se $C^I = D^I$ para cada modelo $I$ de $T$. Neste caso, escreve-se $C \equiv_T D$ ou $T \models C \equiv D$.

- Disjunção: Dois conceitos $C$ e $D$ são disjuntos com respeito a $T$ se $C^I \cap D^I = \emptyset$ para cada modelo $I$ de $T$.

Descobrir que o *TBox* é satisfatível significa dizer que não há contradições lógicas nesse conjunto. *Subsumption* refere-se ao mecanismo utilizado para saber se um dado conjunto é subconjunto de outro. A equivalência permite que conceitos semanticamente equivalentes possam ser tratados dessa forma. Por fim, a disjunção restringe a forma com as instâncias relacionam-se com os conceitos de uma *TBox*.

Com relação aos axiomas de uma *ABox* o principal mecanismo de inferência diz respeito à consistência das assertivas.

- Consistência das assertivas: Uma *ABox* $A$ é consistente com respeito a uma *TBox* $T$ se existe uma interpretação que é tanto um modelo para $A$ quanto para $T$.

Baseando-se neste mecanismo, nota-se que caso dois conceitos sejam disjuntos então não é consistente a assertiva que indica que um mesmo indivíduo seja uma instância de ambos os conceitos. Por exemplo, a combinação das assertivas "Joana é pai de Paulo" e "Joana é mãe de Paulo", resultaria em uma inconsistência lógica e semântica, considerando que o conjunto formado pelos pais seja disjunto do formado pelas mães.

### 3.4.1 *Reasoners*

Um *reasoner* é uma ferramenta capaz de realizar consultas e inferir informações de uma ontologia, de acordo com mecanismos de inferência. Os *reasoners* podem adotar metodologias distintas para inferência em ontologias. Dentre elas, destacam-se as baseadas em cálculo de *tableaux*[1] e as que se baseiam em técnicas de resolução lógica. A Tabela 3.1 apresenta algumas das principais implementações de *reasoners* para a linguagem OWL.

| ***Reasoner*** | **Interface** | **Código aberto** | **Técnica** |
|---|---|---|---|
| Pellet [48] | Java | Sim | *Tableau* |
| KAON2 [45] | Java | Não | *Resolution & Datalog* |
| Fact++ [63] | C++ | Sim | *Tableau* |
| HermiT [58] | Java | Sim | *Hypertableau* |
| RacerPro [24] | XML | Não | *Tableau* |

Tabela 3.1: *Reasoners* para a linguagem OWL.

Como pode ser visualizado na Tabela 3.1, existem atualmente poucos *reasoners* com interface para a linguagem Java, entre eles destacam-se Pellet[48] e HermiT[58], por implementarem a interface OWLReasoner, definida pela OWL API, e terem código aberto.

## 3.5 Ontologias para *smart spaces*

O uso de ontologias em ambientes sensíveis ao contexto tem como motivação a possibilidade de modelar a semântica do contexto de forma independente da linguagem de

[1]Procedimento de prova usado para determinar a satisfatibilidade de um conjunto de fórmulas lógicas.

programação, sistema operacional e/ou *middleware* utilizado. Neste capítulo são apresentadas algumas ontologias encontradas na literatura para modelagem do conhecimento em ambientes sensíveis ao contexto.

### 3.5.1 CoBrA

O projeto CoBrA (*Context Broker Architecture*) utiliza as ontologias COBRA-ONT [10] e SOUPA [11] para representar informações de contexto em ambientes pervasivos. As principais contribuições estão relacionadas aos casos de uso modelados, juntamente com a separação de conceitos chave como localização, lugares, atividade e agentes. Contudo, no modelo apresentado em [10] não há divisão entre os conceitos relacionados ao domínio, como *Campus*, e conceitos mais gerais e aplicáveis a vários domínios, como *Place*, pois ambas as classes são listadas em um mesmo bloco, conforme mostrado na Figura 3.1.

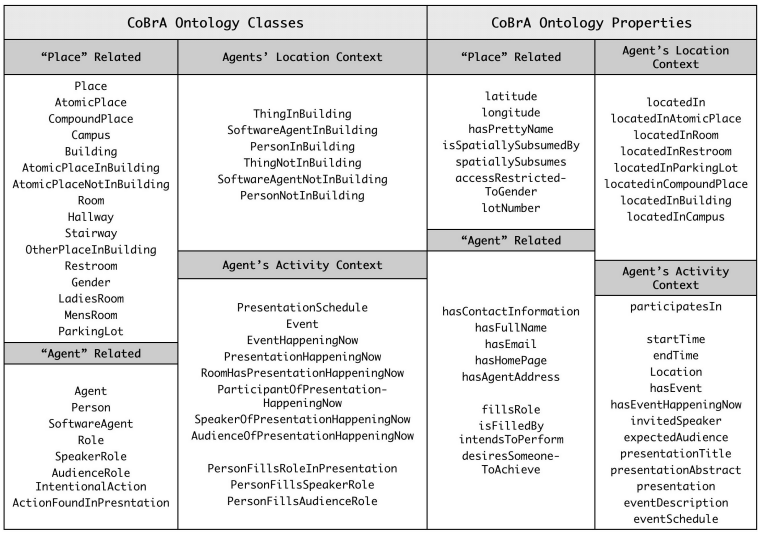



Figura 3.1: Classes e propriedades da ontologia de contexto do projeto CoBrA [10].

Em [11], é apresentado um conjunto de ontologias, desenvolvido a partir da combinação de outras ontologias, a fim de buscar um consenso sobre a forma de representar certos tipos de conceitos. Assim, desenvolvedores com pouco conhecimento sobre ontologia poderiam criar aplicações que utilizassem conceitos modelados de forma padrão. O projeto divide a ontologia SOUPA em duas partes: Soupa *Core* e Soupa *Extension*. A Soupa *Core* define conceitos considerados comuns a várias aplicações sensíveis ao contexto, como: pessoa, política/ação, agente, tempo, espaço e evento. Enquanto a Soupa *Extension* modela conceitos relacionados a domínios específicos, porém, comumente encontrados em várias aplicações sensíveis ao contexto, como: encontros envolvendo agendas de participantes, documentos, captura de imagens, relações de espaço e localização.

### 3.5.2 SOCAM

O SOCAM (*Service-Oriented Context Aware Middleware*) apresenta um modelo de duas camadas baseado em ontologia [22]. Nesse modelo, a camada superior possui con-

ceitos de alto nível e independentes de domínio. Já a camada inferior contém a ontologia específica de domínio, por exemplo o contexto do escritório, carro, etc. Essa divisão em camadas, ilustrada na Figura 3.2, permite que a camada inferior seja ligada a camada superior de forma dinâmica, diminuindo o escopo de análise do contexto. Considerando-se que os dispositivos móveis possuem recursos limitados, a abordagem mostra-se pertinente pois tem como objetivo reduzir o custo de processamento das inferências e consultas realizadas à ontologia.

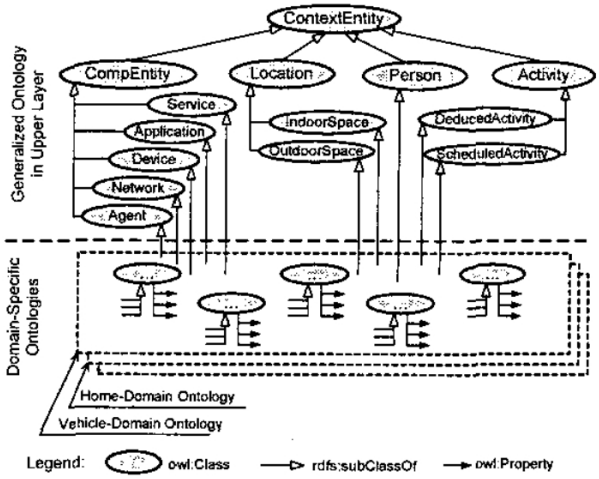



Figura 3.2: Divisão em duas camadas do projeto SOCAM [22].

Em [23] os autores do projeto SOCAM classificam as informações de contexto coletadas do ambiente em *diretas* e *indiretas*. As informações diretas são aquelas capturadas por sensores, podendo estes sensores serem físicos (ex. sensor de presença) ou virtuais (ex. *web services*). Também são classificadas como *diretas* as informações definidas pelos usuários ou pelas aplicações. Enquanto que o contexto *indireto* refere-se ao que pode ser inferido por meio de técnicas de *reasoning* e agregação de informações. Essa classificação é utilizada para resolução de conflitos, em que uma informação definida pelo usuário é considerada mais confiável do que uma informação sentida [23]. Uma outra propriedade definida pelo modelo é a de dependência. A relação de dependência indica quais informações são utilizadas para determinar a situação de uma entidade. Por exemplo, a informação de localização de uma pessoa mostrando que ela está na sala somada a informação de que a televisão está ligada indica que ela está assistindo a televisão.

### 3.5.3 ONTO-MoCA

MoCA (*Mobile Collaboration Architecture*) é um *middleware* desenvolvido para prover suporte a aplicações colaborativas e sensíveis ao contexto [56]. Em [54] é apresentada a ONTO-MoCA, uma extensão desse *middleware* que utiliza uma ontologia para representar informações de contexto. De acordo com os autores, essa ontologia descreve conceitos e propriedades do modelo de contexto do projeto MoCA, com exceção às propriedades relacionadas ao comportamento em tempo de execução [54]. A representação em ontologia tem como objetivo permitir o compartilhamento de informações com outras arquitetu-

ras, possibilitar a modelagem de atributos complexos e prover suporte à verificação de consistência e mecanismos de inferência.

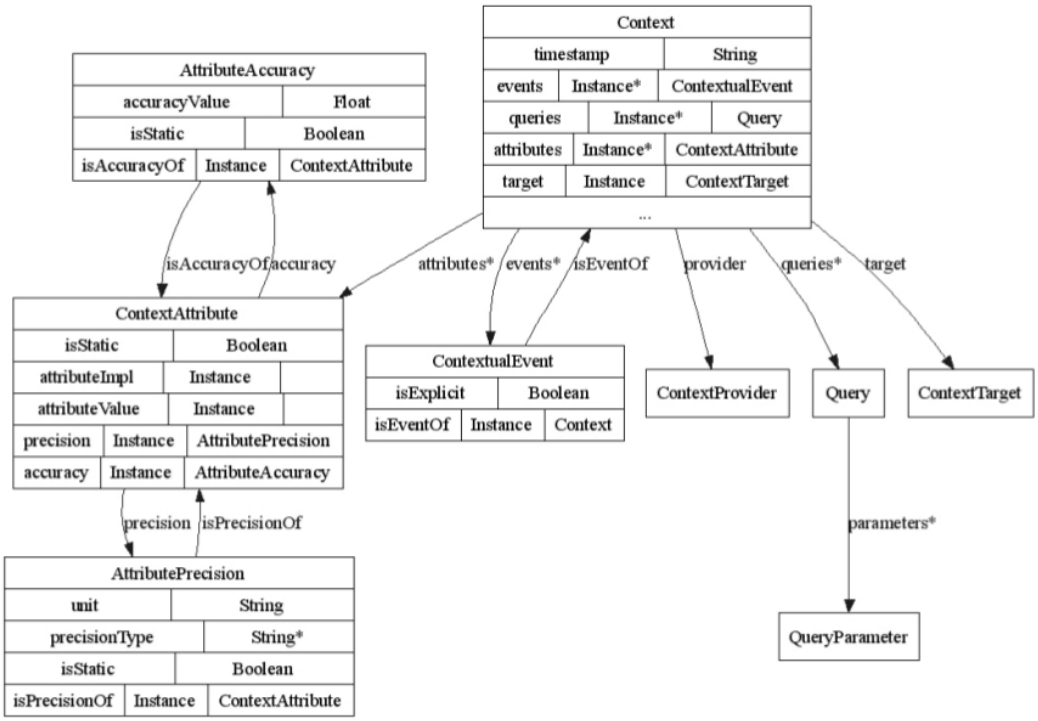


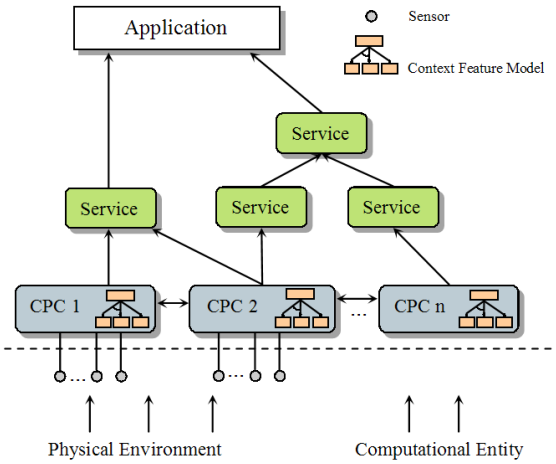

Figura 3.3: Estrutura do modelo de contexto do projeto ONTO-MoCA [54].

A Figura 3.3 apresenta a estrutura do modelo de contexto do projeto. A ONTO-MoCA contém conceitos de alto nível que possam interessar aos desenvolvedores de aplicações sensíveis ao contexto. As principais abstrações do modelo são: eventos, atributos de contexto e consultas. Adicionalmente, o modelo inclui os conceitos acurácia e precisão como atributos de qualidade.

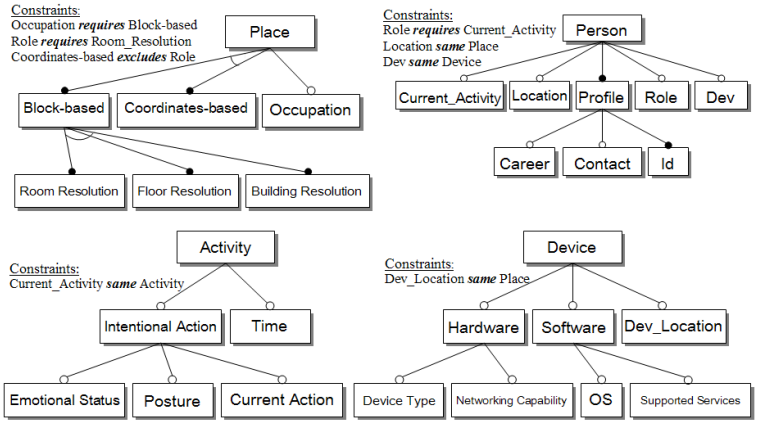
### 3.5.4 CANDEL

O projeto CANDEL (*Context As dyNamic proDuct Line*) apresenta uma forma de modelagem de contexto independente de domínio baseada em modelos de características [34]. O gerenciamento do contexto é realizado por componentes denominados CPCs (*Context Proxy Components*), que atuam como *proxies* das informações de contexto. Por exemplo, o CPC de lugar provê a localização com a característica de coordenadas (ex: latitude e longitude) ou blocos (ex: sala, andar, prédio, etc). Como podem haver várias características acerca de uma mesma informação de contexto, cabe aos serviços selecionar as características a serem utilizadas para prover informações adequadas às aplicações. Dessa forma, o modelo trata os diferentes níveis de abstração das informações de contexto. A Figura 3.4 ilustra a arquitetura do projeto, organizada em aplicações, serviços e os CPCs.

Adicionalmente, o trabalho identifica alguns requisitos relacionados a modelos para o gerenciamento do contexto [34]. Entre esses requisitos pode-se citar: (1) a necessidade de uma infra-estrutura para o desenvolvimento de aplicações; (2) a abordagem deve ser




(a) Relação entre componentes de contexto, os serviços e as aplicações.




(b) Exemplos de componentes descrevendo informações de contexto.

Figura 3.4: Arquitetura do projeto CANDEL para o gerenciamento de contexto [34].

genérica o suficiente para abranger aplicações de vários tipos de domínios; (3) a modelagem de contexto deve seguir padrões conhecidos; e (4) o modelo deve ser dinâmico e permitir que as alterações do ambiente sejam refletidas no sistema. Estes requisitos asseguram que a solução desenvolvida seja flexível, podendo suportar vários tipos de aplicações; compartilhe informações com outros sistemas e seja adaptável às alterações do ambiente.

### 3.5.5 Smart-M3 *Framework*

Em [40] é apresentado um *framework* voltado para o desenvolvimento de aplicações sensíveis ao contexto em *smart spaces*. O trabalho foi implementado de acordo com a arquitetura Smart-M3 da Nokia [28] e apresenta um modelo para representação de informações de contexto baseado em ontologia. A Figura 3.5 ilustra o modelo, que assim como o SOCAM [22] faz divisões entre conceitos independentes de domínio e relacionados ao domínio da aplicação.

De forma semelhante à classificação apresentada em [22], o trabalho faz uma distinção entre contexto *atômico*, captado diretamente de componentes provedores de contexto; e contexto *inferido*, que é composto de informações extraídas a partir do conjunto de dados preexistente na ontologia. O contexto é inferido por meio de regras, as quais são expressas em Python e processadas por uma *engine* do sistema.

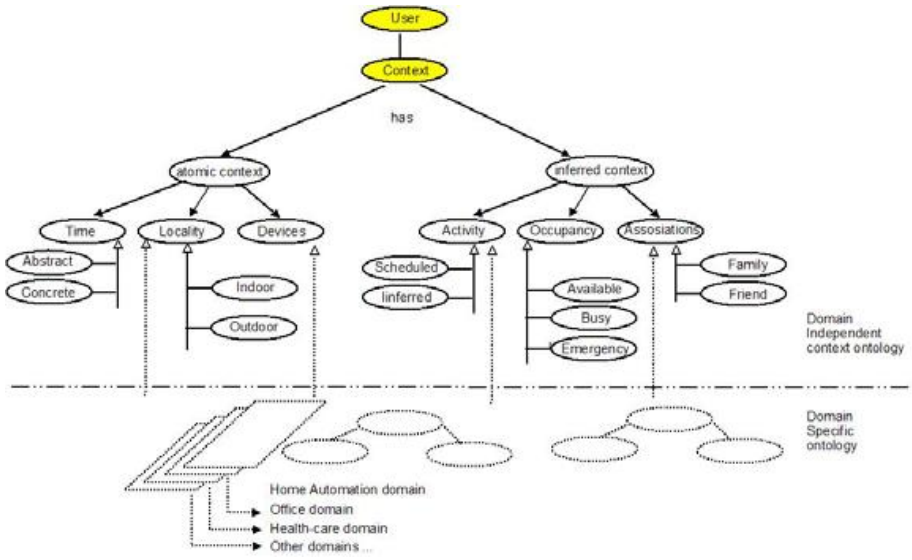



Figura 3.5: Modelo de infomações de contexto baseado em ontologia do projeto Smart-M3 *Framework* [40].

As regras de inferência são baseadas em três cláusulas: (i) *with*, (ii) *when*, (iii) *then*. As cláusulas *with* contém afirmações, enquanto que as do tipo *when* indicam os eventos em que a regra deve ser usada. Por fim, as cláusulas *then* contém as ações do sistema de acordo com informações obtidas a partir dos elementos *with* e *when*. Desse modo, em [40] é apresentado um modelo focado na implementação de uma *engine* de processamento de informações de contexto a partir de cláusulas definidas em Python. Essa abordagem traz como vantagem maior facilidade no desenvolvimento das aplicações, uma vez que o desenvolvedor não necessita de conhecimentos sobre ontologias para utilizar o modelo.

## 3.6 Considerações

Dentre as formas de representação de conhecimento encontradas na literatura, as ontologias têm sido amplamente utilizadas por possibilitarem alto grau de expressividade semântica, que permite descrições detalhadas dos domínios modelados. Desse modo, este capítulo apresenta tópicos considerados relevantes para a definição do modelo baseado em ontologias proposto, tais como conceitos básicos, processo evolutivo, linguagens e ferramentas. Adicionalmente, foram abordados alguns pontos referentes à lógica descritiva, a qual representa a base formal lógica das ontologias. Na Seção 3.2, foi apresentada a linguagem OWL, utilizada neste trabalho, com seus perfis e mecanismos de inferência, que permitem a extração de informações não explícitas da ontologia.

A Seção 3.5 apresenta modelos específicos para gerenciamento de informações de contexto em ambientes ubíquos. A partir da análise dos trabalhos encontrados na literatura, constata-se que a utilização de ontologias para modelagem de informações de contexto em ambientes inteligentes vem recebendo crescente atenção e interesse da comunidade científica. No entanto, foi observado que o uso específico de ontologias nestes trabalhos possui algumas diferenças. Alguns modelos utilizam ontologias com foco na representação de atributos das informações, consultas e modelagem de eventos [54, 34]. Outros trabalhos têm por objetivo estabelecer a melhor forma de organizar o contexto, visando encontrar um denominador comum para a criação de um padrão [11]. Em [22, 40] os conceitos são divididos em camadas, dependendo do nível de generalidade (ou especificidade) da informações. Embora muitos dos trabalhos analisados adotem ontologias para representar o contexto, mecanismos para o gerenciamento e evolução de ontologias em ambientes inteligentes permanecem como foco de pesquisa.

# Capítulo 4

# *Middlewares* em *smart spaces*

Um dos principais objetivos de um *middleware* em ambientes distribuídos é prover uma camada interoperável[1], capaz de encapsular a heterogeneidade dos dispositivos. Além de prover interoperabilidade, os *middlewares* em *smart spaces* devem coletar informações como localização de usuários e dispositivos, estado dos dispositivos, preferências dos usuários e características dos dispositivos. Na Seção 4.1, são apresentadas algumas estratégias utilizadas por *middlewares* para auxiliar as aplicações na obtenção, representação, processamento e entrega das informações de contexto. A Seção 4.2 descreve o *middleware uOS*, adotado na implementação deste trabalho.

## 4.1 Supporte de *middleware*

Conforme visto na Seção 2.2, existem diversos desafios relacionados ao contexto a serem tratados pelas aplicações. No entanto, certos desafios independem do domínio das aplicações e podem ser realizados por *middlewares*. Em [16] são listados alguns serviços que podem ser providos por *middlewares* para dar suporte ao desenvolvimento de aplicações sensíveis ao contexto:

- Serviço de inscrição e entrega de informações;
- Consulta e compartilhamento das informações de contexto;
- Serviço de processamento das informações;
- Serviço de verificação das informações de contexto;
- Descoberta e gerenciamento de serviços.

As próximas subseções apresentam como alguns *middlewares* gerenciam as informações de contexto de acordo com os serviços listados em [16].

[1] Uma camada interoperável é uma camada em que os dispositivos conseguem comunicar-se de forma transparente, sejam semelhantes ou não.

### 4.1.1 Serviço de inscrição e entrega de informações

O Serviço de inscrição e entrega de informações é um serviço em que as aplicações podem inscrever-se e receber informações no momento em que um evento esperado ocorrer. No projeto Gaia [55], o serviço de inscrição e entrega de informações é realizado por meio de um gerenciador de eventos, que possui consumidores, fornecedores e canais para controle de eventos. Em [59], o *middleware* captura informação em tempo real de sensores que enviam mensagens de eventos em uma rede *ad hoc* móvel. O projeto utiliza o modelo *Publish-Subscribe* para prover a comunicação entre os objetos e os eventos são roteados de quem publica a informação para os inscritos via protocolos *multicast*. Em CARMEN (*Context-Aware Middleware for Resource Management in the Wireless Internet*) [4] é apresentado um gerenciador de eventos utilizado para controlar um único tipo de informação de contexto: a mobilidade dos dispositivos. O gerenciador de eventos indica a movimentação de um determinado dispositivo no ambiente e, baseando-se nesses eventos, o *middleware* realiza o redirecionamento do serviço que o recurso provê de acordo com políticas pré-definidas.

### 4.1.2 Consulta e compartilhamento das informações

O Serviço de consulta e compartilhamento das informações permite às aplicações consultar o contexto atual do usuário, ambiente ou da própria aplicação. Adicionalmente, pode-se definir uma estratégia de compartilhamento das informações. O projeto MoCA (*MObile Collaboration Architecture*) [56] contém um serviço chamado *Monitor*, que coleta informação do dispositivo em que está sendo executado e envia para o CIS (*Context Information Service*). O CIS é um serviço distribuído, em que cada servidor recebe as informações de contexto do serviço Monitor dos dispositivos. Assim, as informações de contexto são distribuídas entre os nós da rede e podem ser consultadas pelas aplicações. No projeto SALES (*Scalable context-Aware middleware for mobiLe EnvironmentS*) [13] adota-se o princípio da localidade para proativamente disseminar informações de contexto. Os dispositivos fazem *cache* das informações nos vizinhos (lógicos ou físicos), de modo a reduzir a massa de informações a ser compartilhada.

### 4.1.3 Serviço de processamento das informações

O Serviço de processamento das informações transforma os dados primitivos em informações úteis às aplicações. Por exemplo, as coordenadas de localização de um usuário podem ser transformadas em uma localização baseada em blocos (ex. usuário X está na sala 1.) ou terem conotação de proximidade (ex. usuário X está próximo de uma impressora.). O projeto CAPNET (*Context-Aware Pervasive NETworking*) [14] utiliza *componentes de contexto* para agregar serviços que dão suporte a outros componentes e aplicações no ambiente distribuído. O componente recebe, processa e provê as informações de contexto, atuando como um moderador entre os sensores e as aplicações. A conexão pode ser trocada no meio da comunicação sem que os componentes percebam. Em [67] as informações de contexto chegam ao módulo de processamento de contexto e são mapeadas para um modelo baseado em ontologia. Este módulo de processamento fica em um dispositivo remoto, que transmite as informações processadas para dispositivos móveis.

### 4.1.4 Serviço de verificação das informações

O Serviço de verificação das informações do contexto é um serviço que verifica os dados que serão utilizados pelas aplicações para tomada de decisão. Em [53] é definido um servidor de ontologias, que controla as alterações e evita que informações inconsistentes sejam adicionadas na base de conhecimento. No projeto MIDSEN (*Middleware for Wireless Sensor Network*) [49] são definidos atributos para verificar a validade das informações de contexto capturadas por sensores. Ao armazenar atributos como o horário em que foi feita a leitura pelo sensor pode-se determinar o tempo de vida desse dado no *middleware*. Por exemplo, uma temperatura medida às 2 p.m. não será mais válida à noite.

### 4.1.5 Descoberta e gerenciamento de serviços

A descoberta e gerenciamento de serviços permite que as aplicações localizem e obtenham em tempo real o conjunto de serviços disponíveis no ambiente. Gaia apresenta um Serviço de Presença, que detecta as entidades presentes no *Active Space* [55]. As entidades podem ser físicas, isto é, pessoas e dispositivos (localização), ou digitais, como aplicações e serviços. De forma semelhante, o projeto MoCA [56] contém um serviço para descoberta de dispositivos que armazena informações como nome, propriedades, endereços, etc, sobre qualquer aplicação ou serviço registrado no *middleware*. Além disso, um serviço de configuração captura a configuração dos dispositivos móveis e grava os dados em uma tabela *hash* persistente para facilitar a descoberta de serviços. Em [49] é apresentado um algoritmo que realiza *matching* de serviços, retornando a similaridade entre o serviço requisitado e os disponíveis.

Conforme visto nesta seção, foram publicadas na literatura algumas soluções para tratar dos desafios relacionados ao gerenciamento das informações de contexto. A Seção 4.2, a seguir, descreve o *middleware uOS* [9] e como estes desafios são abordados pelos autores.

## 4.2 *Middleware uOS*

O *middleware uOS* foi concebido com o objetivo de promover a adaptabilidade de serviços em um ambiente de computação ubíqua [9], de modo que os serviços dos dispositivos presentes no ambiente possam ser oferecidos e compartilhados. Conforme apresentado na Figura 4.1, o *uOS* atua como uma camada entre as aplicações e os *drivers*. Um *driver* é uma especificação do conjunto de serviços providos por um dispositivo no ambiente. Para viabilizar a comunicação entre os dispositivos, existem os *plugins* de rede, que são implementações dedicadas a tecnologias de rede específicas, como 3G, *Bluetooth* e *Ethernet*.

O *uOS* adota a arquitetura DSOA (*Device Service Oriented Architecture*). Seguindo a arquitetura SOA (*Service Oriented Architecture*) a DSOA também possui os papéis de consumidor, provedor e registro; com a diferença de que na DSOA quem assume esses papéis são os dispositivos e não os *Web Services*. Assim como em [55, 59, 4], o gerenciamento de eventos segue o modelo *Publish-Subscribe*.

A Figura 4.2 mostra as três camadas do *middleware uOS*:

1. Camada de Rede: responsável por gerenciar as interfaces de rede do dispositivo;

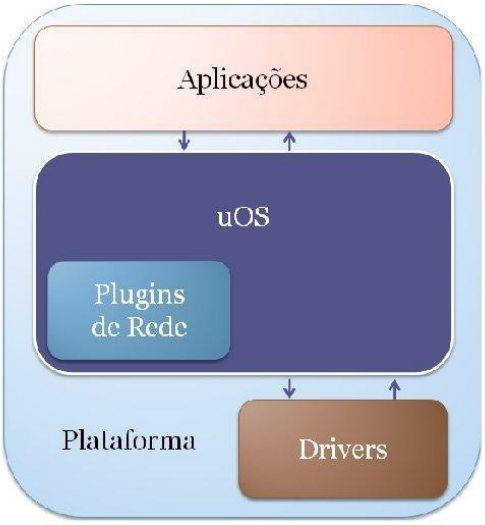



Figura 4.1: *Middleware uOS*

2. Camada de Conectividade: responsável por gerenciar conexões entre dispositivos que se comunicam por diferentes tecnologias, como *Bluetooth* e *Ethernet*;
3. Camada de Adaptabilidade: responsável por gerenciar os serviços disponíveis e o acesso aos mesmos.

A camada de Rede contém os módulos: RADAR e *Connection Manager*. O RADAR descobre os dispositivos do ambiente por meio de varreduras e o *Connection Manager* gerencia conexões entre dispositivos com a mesma tecnologia de comunicação. Na camada de Conectividade, a comunicação entre dispositivos com diferentes tecnologias é obtida com a utilização de um *Proxy*. Por fim, a camada de Adaptabilidade contém um módulo chamado *Adaptability Engine*, que faz a intermediação entre as aplicações e os *drivers* com o objetivo de selecionar o serviço mais apropriado para determinada aplicação. A camada de Adaptabilidade também contém o módulo *Application Deployer*, que permite a inclusão e exclusão de aplicações no *uOS* por meio das operações de *Deploy* e *Undeploy*.

## 4.3 Considerações

Neste capítulo foram apresentados alguns serviços que podem ser providos por *middlewares* para tratar desafios comuns às aplicações e controlar o fluxo das informações de contexto em *smart spaces*. Estes serviços listados em [16]: (i) determinam para quais dispositivos as informações precisam ser enviadas; (ii) possibilitam a comunicação dos dispositivos; (iii) auxiliam no processamento; (iv) síntese das informações de contexto; e ainda (v) descobrem dinamicamente os serviços disponíveis.

O *middleware uOS* suporta os serviços (i), (ii) e (v). O serviço (i) é provido por meio do gerenciamento de eventos, segundo o modelo *Publish-Subscribe*. O serviço (ii) é contemplado pelos *plugins* de rede e por *proxies* na camada de conectividade. A descoberta de dispositivos e serviços (v) é gerenciada na camada de adaptabilidade utilizando dos módulos *Device Manager* e *Driver Manager*.

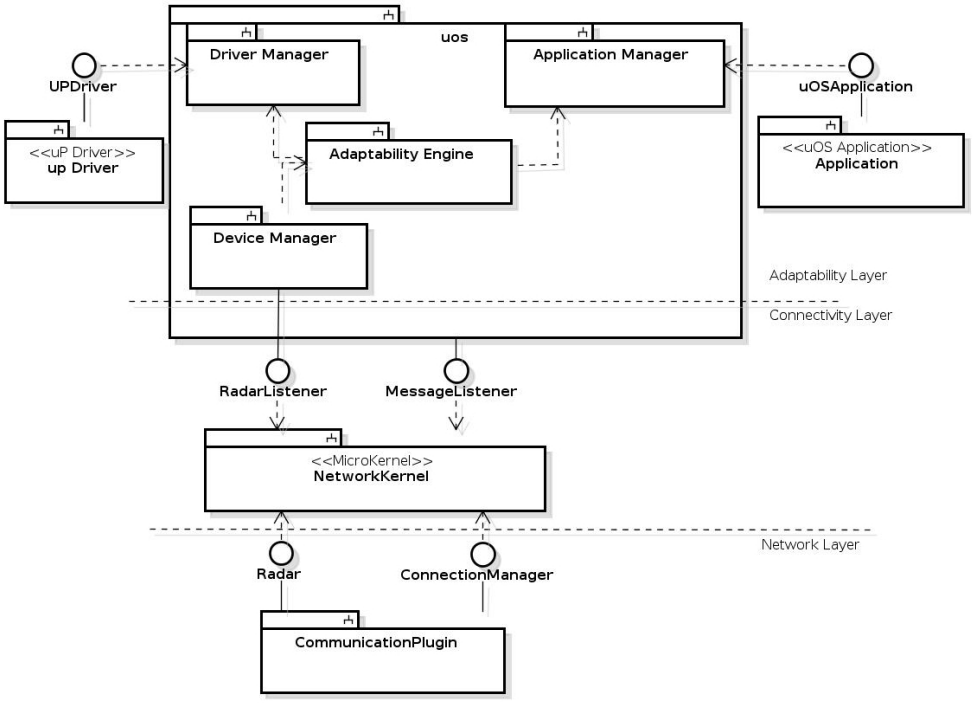



Figura 4.2: Camadas do *middleware uOS*.

A proposta deste trabalho suporta os serviços (iii) e (iv) e complementa o serviço (ii), disponibilizando consultas a ontologias locais e remotas para compartilhamento das informações de contexto. O serviço (iii) é obtido por meio da representação em forma de ontologia, que permite consultas utilizando mecanismos de inferência. O serviço (iv) é provido por meio de um processo de verificação que controla alterações na base de informações de contexto.

# Capítulo 5

# Proposta

Neste capítulo é apresentada a proposta do trabalho, que consiste em um modelo para o gerenciamento de informações de contexto e sua implementação no *middleware uOS*. A Seção 5.1 descreve os componentes do modelo proposto e a Seção 5.2 traz alguns detalhes de implementação.

## 5.1 Modelo para gerenciamento do contexto

O gerenciamento do contexto requer o controle do fluxo de informações que passam pelo sistema, incluindo a propagação dos dados aos dispositivos do ambiente. Para inserir e remover informações na ontologia, é desejável o uso de uma API, capaz de encapsular operações especificamente relacionadas à persistência. Adicionalmente, é preciso coordenar as alterações realizadas na ontologia, de modo a preservar a consistência dos dados.

Esta seção descreve o modelo para o gerenciamento de informações de contexto baseado em ontologias proposto neste trabalho. O modelo contém três componentes principais: (i) Gerenciador de Contexto, (ii) *Context API* e (iii) Gerenciador de Alterações. A Figura 5.1 ilustra os componentes responsáveis por gerenciar as alterações de contexto realizadas pelas aplicações. As operações de *Deploy* e *Undeploy* que aparecem na figura referem-se à inserção e remoção de aplicações no modelo (conforme apresentado na Seção 4.2).

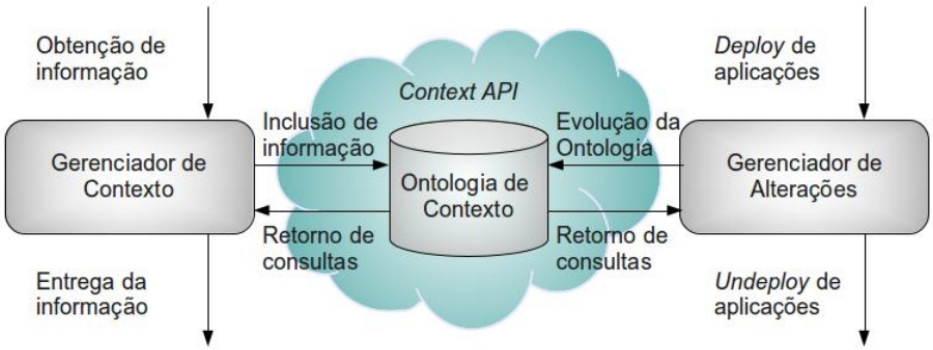



Figura 5.1: Modelo baseado em ontologia para o gerenciamento de contexto.

Basicamente, as informações chegam ao Gerenciador de Contexto, o qual é responsável por entregar informações de contexto atualizadas às aplicações que estão registradas para recebê-las. O mecanismo de registro segue o modelo *Publish-Subscribe* e a disseminação de informações ocorre por meio de eventos, com a entrada e saída de dispositivos no *smart space* de forma dinâmica. Usando a *Context API*, as aplicações podem alterar a ontologia e evoluir as informações de contexto. Antes de serem salvas, estas alterações passam por um processo de verificação realizado pelo Gerenciador de Alterações. A linguagem utilizada para a ontologia é a OWL (*Web Ontology Language*), por possibilitar um alto grau de expressividade semântica e ser recomendada pela W3C (*World Wide Web Consortium*). As Subseções 5.1.1, 5.1.2 e 5.1.3 a seguir detalham os componentes envolvidos no modelo proposto.

### 5.1.1 Gerenciador de Contexto

O Gerenciador de Contexto é o componente responsável por controlar o fluxo das informações de contexto, as quais são essencialmente obtidas por meio de sensores e representadas na ontologia. O Gerenciador de Contexto verifica as atualizações na ontologia e notifica as aplicações que estão registradas para recebê-las. Durante a execução das aplicações, consultas à ontologia podem feitas com a utilização de mecanismos de inferência.

Conforme ilustrado na Figura 5.2, o Gerenciador de Contexto contém, por padrão, uma ontologia de alto nível. As informações pré-definidas compõem um conjunto de conceitos independentes de domínio, formando uma camada superior. Esta camada serve de base para que as aplicações incorporem informações específicas aos domínios modelados. A adoção dessa ontologia serve para padronizar a forma como é tratado cada tipo de conceito relacionado ao contexto. As ontologias de domínio, as quais extendem a ontologia de alto nível, são construídas e evoluídas a partir das aplicações por meio da *Context API*, detalhada na Seção 5.1.2.

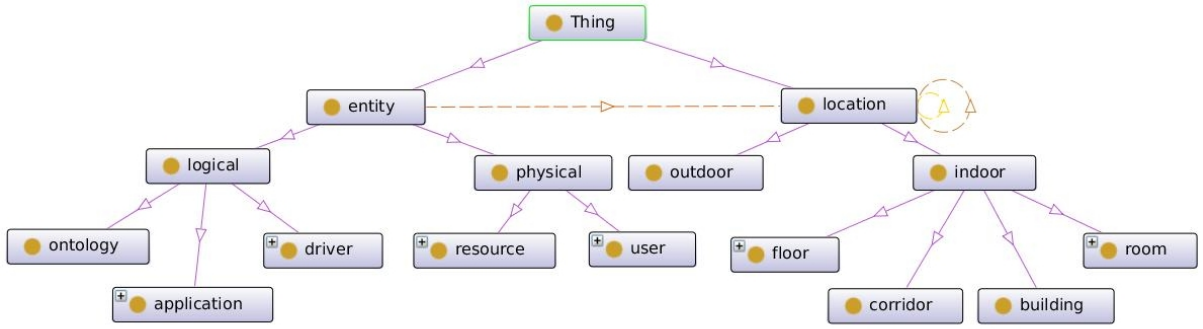



Figura 5.2: Ontologia de alto nível de abstração das informações de contexto. (Figura gerada pela ferramenta *Protégé* [46].)

A ontologia apresentada pela Figura 5.2 pode ser dividida em duas categorias: hierarquia para entidades e hierarquia para localização, as quais são descritas nas Subseções 5.1.1 e 5.1.1, a seguir.

### Hierarquia para entidades

A Figura 5.3 apresenta a hierarquia independente de domínio para entidades. O segundo nível divide a entidade em física e lógica. Uma entidade física pode ser um usuário ou um recurso do ambiente, como uma câmera ou um monitor, por exemplo. Por outro lado, a entidade lógica representa conceitos como *drivers* de dispositivos, aplicações e ontologias.

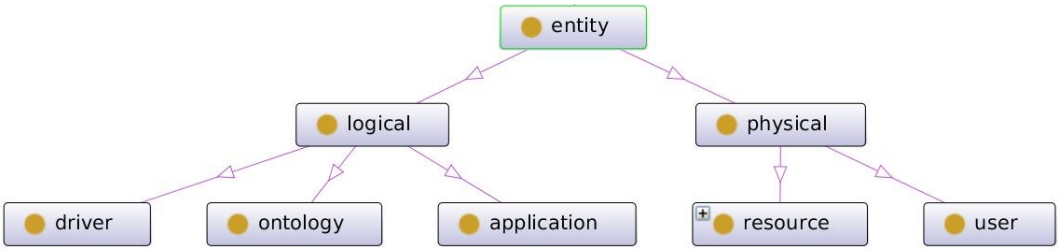



Figura 5.3: Hierarquia independente de domínio para entidades. (Figura gerada usando Protégé [46])

### Hierarquia para localização

A Figura 5.4 apresenta a hierarquia independente de domínio para localização. Como pode ser observado, as classes podem dividir-se em locais abertos ou fechados, como em [22, 40, 22]. Para locais fechados, a divisão é feita por salas, corredores, andares e prédios, conforme em [19].

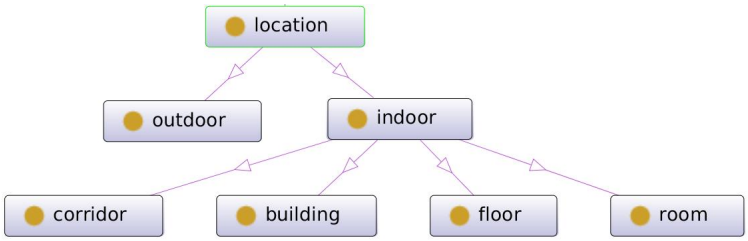



Figura 5.4: Hierarquia independente de domínio para localização. (Figura gerada usando Protégé [46])

Com base na hierarquia apresentada na Figura 5.4 as seguintes propriedades de objetos foram definidas:

- *isLocatedIn* (*Transitive*) - e.g., se a sala 1 está no primeiro andar do prédio X então a sala 1 está no prédio X.

  - *isEntityInLocation*
  - *isFloorInBuilding*
  - *isRoomInFloor*

- *isLocationOf* (*Inverse of isLocatedIn*)

A propriedade *isEntityInLocation* serve para estabelecer uma relação entre uma entidade da ontologia, pessoa, dispositivo, entre outros, e um local. Por exemplo, suponha que Lucas é uma instância da classe *user*, definida na hierarquia de entidades (Figura 5.3), e sala 1 é uma instância da classe *room*, contida na hierarquia de localização (Figura 5.4). Então, a informação de que "Lucas está na sala 1", poderia ser representada na ontologia pela inclusão de uma afirmação da forma "*isEntityInLocation*( Lucas, sala 1 )".

Outras questões poderiam ser extraídas da ontologia como por exemplo "Quem está na sala 1?", que neste caso receberia como resposta "Lucas". Esta informação pode ser derivada por meio da propriedade de objetos *isLocationOf*, definida como inversa da propriedade *isLocatedIn*. Ao definir a propriedade *isLocatedIn* como transitiva pode-se derivar que se A está localizado dentro da região de B, que por sua vez está dentro de C, então A está dentro de C. Por exemplo, se definirmos que a "sala 1" está no "piso 1" então também poderíamos inferir que "Lucas está no piso 1". Mais detalhes são apresentados no Apêndice A.3.

### 5.1.2 *Context API*

Dependendo do domínio, as aplicações sensíveis ao contexto podem requerer que diferentes conceitos, com suas propriedades e relacionamentos, estejam presentes na ontologia. Portanto, o modelo deve permitir que tais conceitos sejam refletidos na ontologia. A utilização de uma API tem como vantagem o desacoplamento das alterações realizadas em relação à forma de persistência da ontologia. Além disso permite que tais alterações ocorram de forma padronizada e controlada pelo Gerenciador de Alterações.

Construída sobre a OWL API [29], a *Context API* consiste em um conjunto de interfaces de programação que tem como objetivo viabilizar a manipulação estrutural e semântica da ontologia de contexto. Dessa forma, a construção da ontologia torna-se dinâmica e capaz de evoluir de acordo com as necessidades do ambiente.

Conforme descrito em [32], um conjunto de transformações é completo se para cada entidade existe uma transformação correspondente de adição e remoção. Desse modo, a Tabela 5.1 apresenta o conjunto completo de operações com respeito às entidades envolvidas em uma ontologia construída com a linguagem OWL 2. Cada operação representa uma alteração minimal na ontologia e pode ser realizada por meio da *Context API*.

A Tabela 5.2 mostra as principais operações de consulta e inferência que podem ser realizadas na ontologia. Dessa forma, a *Context API* disponibiliza tanto operações de inserção e remoção de axiomas quanto consultas à ontologia.

Analisando o conjunto apresentado na Tabela 5.1, pode-se perceber que para realizar algumas tarefas o desenvolvedor teria que dividí-la em vários passos.

**Exemplo:** Suponha que uma determinada aplicação tenha uma classe X com instâncias $\{x_1, x_2, x_3, ..., x_n\}$. Caso o desenvolvedor quisesse remover todas as instâncias da classe X teria que iterar $n$ vezes em seu programa até que as instâncias fossem removidas.

| Entidade | Operação | |
|---|---|---|
| *Class* | *addClass* | *removeClass* |
| | *addSubClass* | *removeSubClass* |
| | *addEquivalentClass* | *removeEquivalentClass* |
| | *addDisjointClass* | *removeDisjointClass* |
| *Instance* | *addInstanceOf* | *removeInstanceOf* |
| | *addObjectPropertyAssertion* | *removeObjectPropertyAssertion* |
| | *addDataPropertyAssertion* | *removeDataPropertyAssertion* |
| *Data Property* | *addDataProperty* | *removeDataProperty* |
| | *addDataSubProperty* | *removeDataSubProperty* |
| | *addDataPropertyDomain* | *removeDataPropertyDomain* |
| | *addDataPropertyRange* | *removeDataPropertyRange* |
| *Object Property* | *addObjectProperty* | *removeObjectProperty* |
| | *addObjectSubProperty* | *removeObjectSubProperty* |
| | *addObjectPropertyDomain* | *removeObjectPropertyDomain* |
| | *addObjectPropertyRange* | *removeObjectPropertyRange* |
| | *addTransitiveProperty* | *removeTransitiveProperty* |
| | *addSymmetricProperty* | *removeSymmetricProperty* |
| | *addInverseProperty* | *removeInverseProperty* |

Tabela 5.1: Entidades da ontologia e operações básicas de adição e remoção.

1. remova a instância $x_1$ da classe X;
2. remova a instância $x_2$ da classe X;
3. ...
4. remova a instância $x_n$ da classe X;

A fim de diminuir o número de operações necessárias para executar uma alteração na ontologia, foi incluído na API um conjunto de operações formado por composições das operações básicas. Este conjunto não agrega nenhuma funcionalidade em relação às operações mostradas anteriormente, sendo introduzido apenas com o intuito de diminuir o esforço de programação dos desenvolvedores que utilizarem a API. A Tabela 5.3 apresenta o conjunto de operações compostas da *Context API*.

### 5.1.3 Gerenciador de Alterações

O Gerenciador de alterações abrange a etapa de *verificação e implementação de alterações* do processo de evolução da ontologia, descrito na Seção 3.1. Ao permitir que as aplicações alterem a base de conhecimento representada por uma ontologia, facilita-se o compartilhamento de informações, uma vez que as aplicações podem consultar a base de forma conjunta. Contudo, traz como desvantagem o surgimento de possíveis inconsistências, como por exemplo, a inclusão de classe disjunta de si mesma. Para evitar que certas alterações tornem a base inconsistente, o Gerenciador de Alterações realiza um processo de verificação sobre cada alteração na ontologia. Assim, as alterações na ontologia

| Entidade | Operação |
|---|---|
| *Class* | *getInstancesFromClass*<br>*getSubClassesFromClass*<br>*getSuperClassesFromClass*<br>*isSubClassOf*<br>*areEquivalentClasses*<br>*areDisjointClasses* |
| *Instance* | *isInstanceOf*<br>*hasDataProperty*<br>*hasObjectProperty* |
| *Data Property* | *getDataPropertyDomains*<br>*getDataPropertyValues*<br>*getSubDataPropertiesFromDataProperty*<br>*getSuperDataPropertiesFromDataProperty* |
| *Object Property* | *getObjectPropertyDomains*<br>*getObjectPropertyRanges*<br>*getObjectPropertyValues*<br>*getSubObjectPropertiesFromObjectProperty*<br>*getSuperObjectPropertiesFromObjectProperty* |

Tabela 5.2: Operações de consulta e inferência na ontologia.

são controladas antes de serem salvas, de modo a não permitir que a base fique em um estado inconsistente.

### Processo de verificação

O processo de verificação do Gerenciador de Alterações utiliza um conjunto de regras que precisam ser válidas para que uma alteração seja aprovada. Em [61] [33], são apresentadas algumas regras para garantir a consistência e concisão da ontologia. O processo de verificação adota algumas destas regras (1-4) e inclui uma quinta regra de proteção (5), para garantir que axiomas inseridos por uma aplicação não sejam removidos por outra.

1. As classes, propriedades e instâncias possuem um identificador único[1].
2. As hierarquias de classes, propriedades de dados e objetos são grafos acíclicos direcionados.
3. Uma classe não pode ser declarada disjunta dela mesma.
4. As hierarquias de classes, propriedades de dados e objetos não são redundantes.
5. Axiomas inseridos por uma aplicação não podem ser removidos por outra.

[1]Em conformidade com a OWL 2, entidades distintas (ex: classe e instância) podem ter o mesmo nome. Contudo, essa possibilidade é apenas sintática, não acrescentando nenhuma relação semântica entre as entidades.

| Entidade | Operação |
|---|---|
| *Class* | *moveSubClass* |
| *Instance* | *moveInstanceOf*<br>*moveInstancesOfClass*<br>*removeInstancesOfClass* |
| *Data Property* | *moveSubDataProperty*<br>*changeDataPropertyDomain*<br>*changeDataPropertyRange* |
| *Object Property* | *moveSubObjectProperty*<br>*changeObjectPropertyDomain*<br>*changeObjectPropertyRange* |

Tabela 5.3: Entidades da ontologia e operações compostas.

A regra 1 garante que em um dado instante, a ontologia pode abrigar apenas uma classe, propriedade ou instância com o mesmo identificador. Ao incluir um axioma, uma anotação é gravada na ontologia indicando qual aplicação realizou a inclusão. A segunda regra diz respeito à estrutura da hierarquia. Apesar de não causar uma inconsistência lógica na ontologia, a formação de ciclos geralmente traz como consequência inconsistência semântica. Um ciclo indica que os conceitos envolvidos são equivalentes. No entanto, caso os conceitos fossem equivalentes eles deveriam ser declarados dessa forma. Uma classe disjunta de si mesma seria insatisfatível, portanto, a regra 3 evita uma inconsistência lógica. A regra 4 trata da concisão da ontologia, evitando redundâncias. Por fim a regra 5 impede que os axiomas sejam removidos arbitrariamente da ontologia, protegendo as alterações realizadas pelas aplicações.

## 5.2 Implementação

O modelo proposto foi implementado e integrado ao *middleware uOS* [9]. Esta seção descreve alguns detalhes da implementação do modelo e da integração com o *middleware* adotado. Os detalhes de implementação incluem validadores para as regras do Gerenciador de Alterações, o desenvolvimento de um *driver* para a ontologia e o controle de concorrência sobre o acesso à ontologia.

### 5.2.1 Integração com o *middleware uOS*

Esta seção descreve os principais pontos relacionados a incorporação do modelo ao *middleware uOS*. A Figura 5.5 apresenta o modelo proposto, denominado *Context Engine*, na camada de adaptabilidade do diagrama do *uOS*.

A *Context Engine* funciona como uma extensão do *uOS*, fornecendo suporte ao gerenciamento das informações de contexto representadas em ontologias. As funções do modelo podem ser desabilitadas do *middleware* sem nenhuma alteração em sua estrutura, conforme descrito no Apêndice A.1.3.

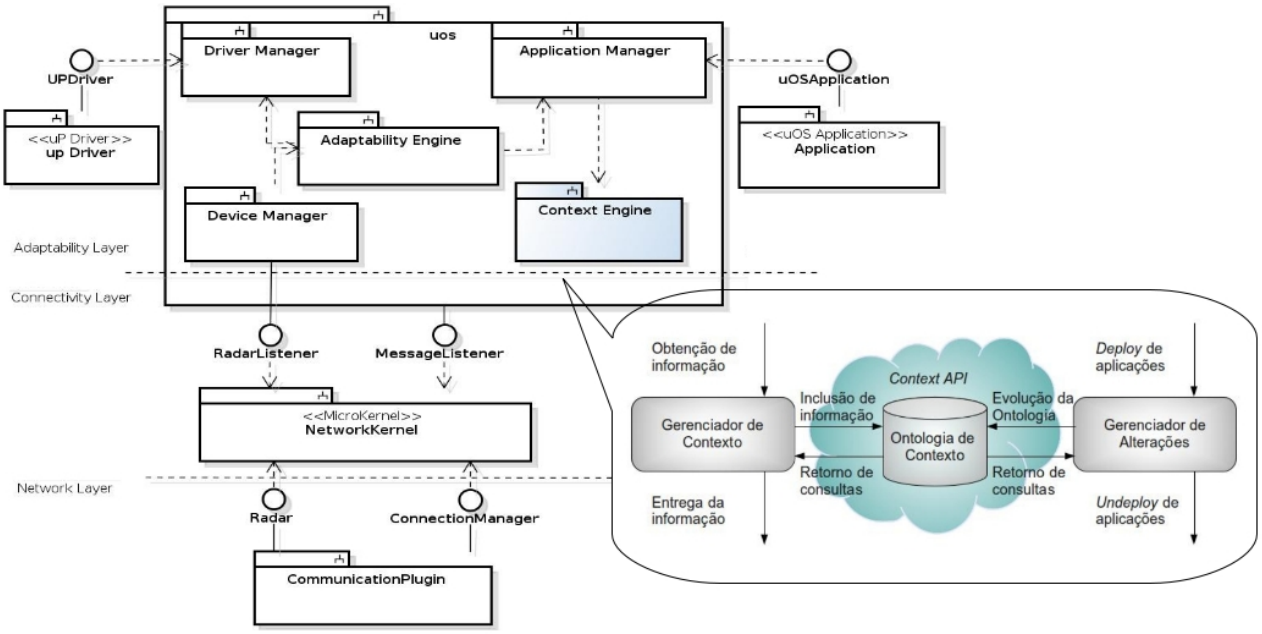



Figura 5.5: Integração do modelo no *middleware uOS*.

### Interface das aplicações com o *uOS*

Para serem integradas ao *uOS*, as aplicações precisam seguir uma interface chamada *UosApplication*. Esta interface, descrita em [8], define o ciclo de vida da aplicação em apenas início e término de execução. As aplicações não armazenam nenhuma informação para consultas posteriores e todos os dados produzidos são consumidos quase simultaneamente. Apesar de eficaz, esta definição não é compatível com a idéia da construção de uma base de conhecimento, capaz de agregar informações necessárias às aplicações sensíveis ao contexto. Assim propõe-se a adição dos seguintes métodos à interface:

- *void init(OntologyDeploy ontology)* : Método invocado para a inclusão de conceitos, propriedades e restrições na ontologia de contexto. A inclusão desses elementos é feita com o uso da *Context API*. O método é chamado apenas no momento de *Deploy* da aplicação e tem a capacidade de alterar estruturalmente a ontologia de contexto.

- *void tearDown(OntologyUndeploy ontology)*: Método invocado para a remoção de conceitos, propriedades e restrições incluídos no *Deploy* da aplicação.O método *tearDown* é chamado apenas no momento de *Undeploy* da aplicação e tem a capacidade de alterar estruturalmente a ontologia de contexto.

Também propõe-se a alteração do método *start* da interface *UosApplication* de modo enviar como parâmetro um objeto que permita consultas e a inserção e remoção de informações na ontologia.

- *void start(Gateway gateway, OntologyStart ontology)* : Método invocado para a iniciar a execução de uma aplicação no *middleware*. O parâmetro *ontology* permite a inclusão e remoção de instâncias na ontologia, assim como a realização de consultas.

### Ciclo de vida das aplicações no *uOS*

O ciclo de vida das aplicações abrange desde o momento em que a aplicação é inserida no sistema até o ponto em que é removida. A Figura 5.6 ilustra o ciclo de vida proposto para aplicações no *uOS*, o qual é composto por três fases: *deploy*, *execution* e *undeploy*.

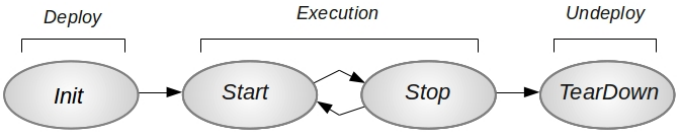



Figura 5.6: Ciclo de vida da aplicação no *middleware uOS*.

Durante a fase de *deploy*, a aplicação inicializa a ontologia com conceitos e propriedades predefinidos, os quais serão utilizados na próxima fase, *execution*. Na fase *execution*, as aplicações usualmente inserem instâncias na ontologia, iniciando (*start*) e interrompendo (*stop*) a execução sem a necessidade de inserirem novamente na ontologia os axiomas definidos na fase *deploy*. Por fim, na fase *undeploy*, a aplicação pode remover conceitos, propriedades e instâncias, chamando operações de remoção providas pela *Context API*.

### Concorrência

Cada instância de uma aplicação em execução no *uOS* corresponde a uma *Thread*. Conforme descrito em [29], a OWL API não é *Thread Safe*. Portanto faz-se necessária a utilização de mecanismos de sincronização entre as *Threads* que podem acessar a ontologia. A implementação contém *locks* de leitura e de escrita. Os *locks* de leitura são utilizados nas consultas. Enquanto os *locks* de escrita foram introduzidos nos pontos onde são gravadas as alterações na ontologia.

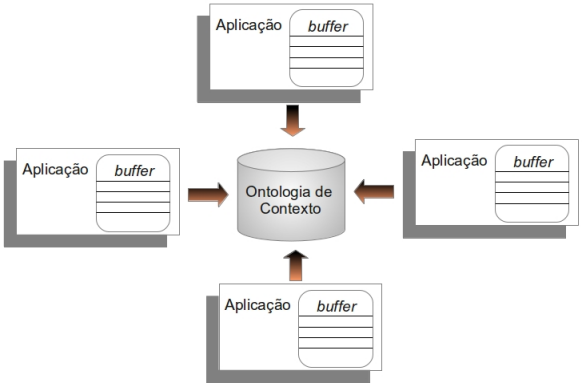



Figura 5.7: Aplicações executando no *uOS* e seus *buffers* de alterações na ontologia.

A OWL API permite que as alterações sejam armazenadas em um *buffer* ao invés de serem imediatamente executadas. Deste modo, as alterações podem ser gravadas na ontologia em um momento oportuno, determinado pela aplicação. A implementação da *Context API* segue este mesmo princípio, o que permite aumentar a concorrência das aplicações, uma vez que cada aplicação possui seu próprio *buffer* (Figura 5.7). Desta forma, as alterações podem ocorrer em paralelo, sendo necessária a exclusão mútua somente no momento de gravação.

### 5.2.2 Gerenciador de Contexto

O Gerenciador de Contexto controla o fluxo das informações de contexto, que podem ser obtidas por meio de serviços e acessadas pelas aplicações. Conforme a arquitetura DSOA [9], qualquer recurso no *smart space* possui um *driver*. Isso permite que aplicações executando em outros dispositivos acessem os serviços providos pelo recurso. O recurso pode ser tanto físico (ex: impressora, teclado, etc.) como lógico. Considerando a ontologia como um recurso lógico, um *driver* para a ontologia de contexto presente no *middleware uOS* foi implementado. Esse driver permite que uma aplicação consulte ontologias existentes em dispositivos remotos, possibilitando que as informações de contexto sejam compartilhadas entre os dispositivos (Figura 5.8).


Figura 5.8: Dispositivos compartilhando informações de contexto utilizando o *Ontology Driver*.

Conforme descrito em [8], a forma de interação entre os dispositivos pode ser síncrona ou assíncrona. Nos serviços providos de forma síncrona a requisição do serviço é seguida de uma resposta. Esta forma de interação é adequada para solicitações de informação que sirvam para apoiar alguma decisão imediata de uma aplicação. Por exemplo, uma aplicação que imprime documentos com base na localização do usuário pode requerer a informação de qual impressora está mais próxima do usuário em determinado no momento. Assim, a aplicação terá que aguardar, de forma síncrona, a resposta dos serviços de localização para decidir qual impressora será escolhida para o envio do documento. Por outro lado, a forma assíncrona é indicada para os casos em que a aplicação deseja aguardar a ocorrência de certos eventos para que a ação seja tomada.

No caso do *driver* para a ontologia os serviços síncronos são consultas à ontologia, representadas na Figura 5.9. A ferramenta utilizada para consultar e realizar inferências sobre a ontologia é a HermiT [58]. No entanto, uma outra ferramenta que implementasse

a interface de *Reasoner* da OWL API, como Pellet [48], por exemplo, também poderia ser utilizada.

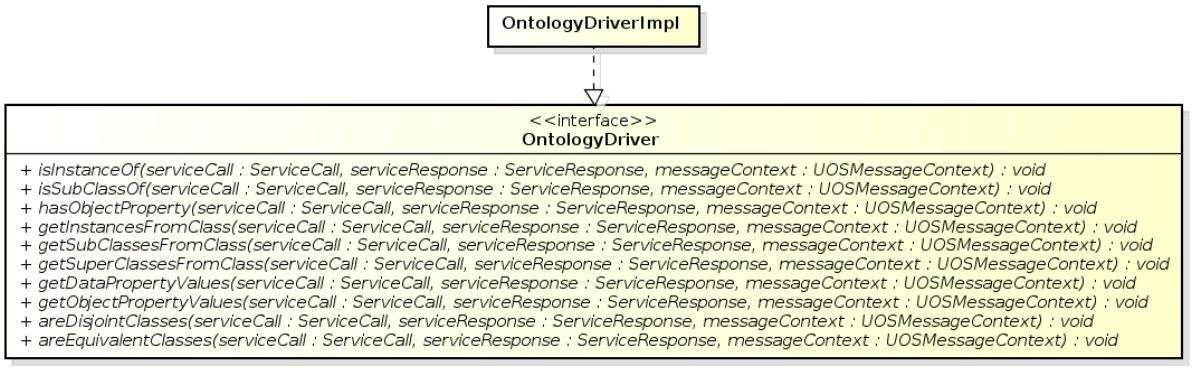



Figura 5.9: Serviços providos pelo *driver* da ontologia.

A forma de interação assíncrona segue o modelo *Publish-Subscribe*, descrito em [8]. Os eventos são divididos em três categorias:

- *InstanceOf*: o evento é lançado quando uma aplicação insere na ontologia uma nova instância.
- *DataProperty*: o evento é lançado quando uma aplicação insere na ontologia uma declaração de *data property*, ou seja, associa uma instância a uma propriedade de dados.
- *ObjectProperty*: o evento é lançado quando uma aplicação insere na ontologia uma declaração de *object property*, ou seja, associa uma instância a uma propriedade de objetos.

Desse modo, além do compartilhamento de informações de contexto entre as aplicações de um dispositivo, é possível consultar as bases de conhecimento de dispositivos remotos. Para maiores detalhes em relação a forma de utilizar os serviços providos pelo *driver* da ontologia consulte a Seção A.4.

### 5.2.3 *Context API*

A *Context API* contém métodos que realizam as operações de adição e remoção de axiomas na ontologia. A implementação das operações básicas e compostas foi dividida em quatro módulos, um para cada entidade representada: classe, instância, propriedade de dados e propriedade de objetos. Além dessas operações, foi implementado um módulo dedicado às consultas, que comunica-se com uma ferramenta de *reasoning* [2].

A Figura 5.10 ilustra a organização em módulos da *Context API*, mostrando também as operações *save* e *initialize*, que servem para salvar as alterações na ontologia e gravar informações definidas por padrão na ontologia. Os protótipos dos métodos implementados

[2]Experimentos foram feitos com a ferramenta HermiT [58]

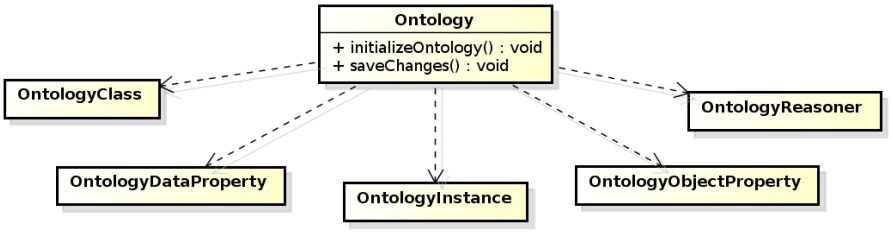



Figura 5.10: Organização dos módulos da *Context API*.

são apresentados nas Figuras 5.11, 5.12, 5.13 e 5.14. As entidades da ontologia foram divididas em módulos que implementam as interfaces utilizadas para *deploy* e *undeploy* de informações na ontologia. Os tipos de dados suportados para definição das propriedades de dados são: *String*, *Boolean*, *Float* e *Integer*.

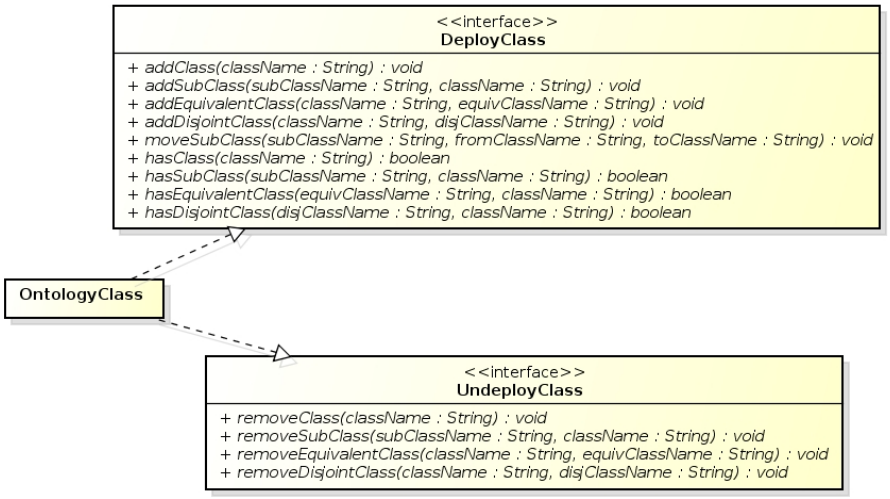



Figura 5.11: Métodos da API envolvendo classes.

No Apêndice A.3 foram apresentados exemplos de consultas utilizando o *reasoner*. A Figura 5.15 descreve métodos relacionados a consultas que podem ser realizadas na ontologia por meio desta implementação. Detalhes relacionados à utilização dos métodos providos pela *Context API* em aplicações estão no Apêndice A.2.

### 5.2.4 Gerenciador de Alterações

A proposta deste trabalho inclui um processo de verificação das alterações (Seção 5.1.3), que contém regras a serem validadas para a manutenção da consistência da ontologia. As subseções a seguir mostram a implementação das regras utilizadas na verificação.

#### Identificador Único

Quando uma aplicação tenta gravar uma entidade (classe, propriedade de dados, propriedade de objetos ou instância) idêntica na ontologia, o comportamento padrão da OWL

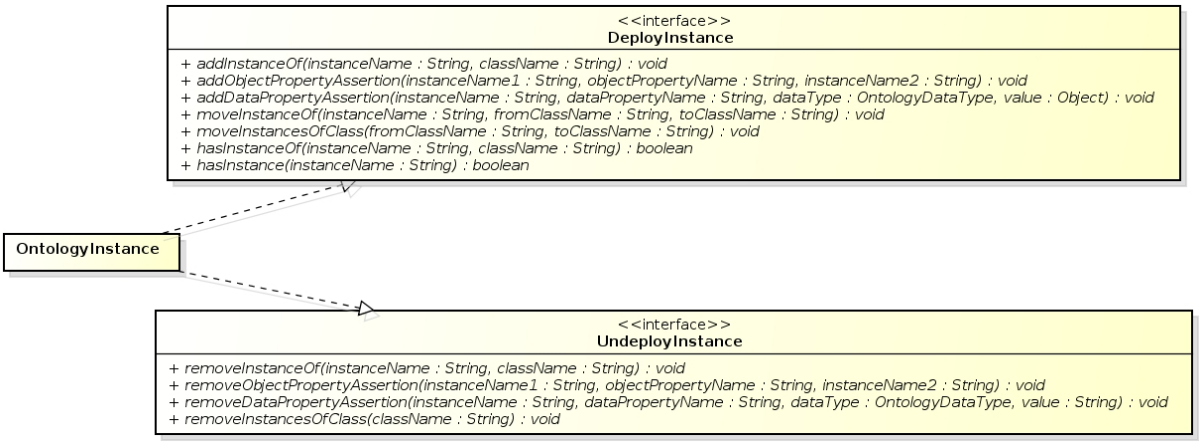



Figura 5.12: Métodos da API envolvendo instâncias.

API[3] é permitir a existência duplicada da entidade em arquivo. Isso pode levar a um desperdício do espaço em disco, uma vez que apenas uma declaração é necessária para incluir uma mesma entidade. Portanto, foi implementado um validador que é chamado a cada solicitação de inclusão de entidade, com o propósito de verificar se a entidade já não está presente na ontologia. Caso já esteja presente, o gerenciador de alterações não aceita essa inclusão e garante a regra 1, definida pelo modelo.

### Grafo Acíclico Direcionado

O resultado de um ciclo na hierarquia é a equivalência das entidades envolvidas. No entanto, a OWL possui um construtor específico para a definição de equivalência. Portanto, a formação de ciclos na hierarquia em geral decorre de anomalias na estrutura das classes. Para evitar a ocorrência de ciclos e garantir a regra 2 do modelo, foi implementado um validador que é chamado a cada solicitação de inclusão de subclasse, subpropriedade de dados e subpropriedade de objetos.

**Algoritmo 1**: Verifica se a inclusão do axioma que diz que $\gamma$ é subclasse de $\delta$ formará um ciclo.

**Entrada**: $\gamma$: subclasse; $\delta$: classe.
**Saída**: Valor booleano indicando se a inclusão do axioma forma um ciclo.

```
1 Π ← superclasses(δ);
2 Π inclui superclasses diretas e indiretas de δ;
3 forall α ∈ Π do
4     if α = γ then
5         return true ;
6     end
7 end
8 return false ;
```

[3]OWL API Versão 3.2.4 - lançada em 22 junho de 2011.

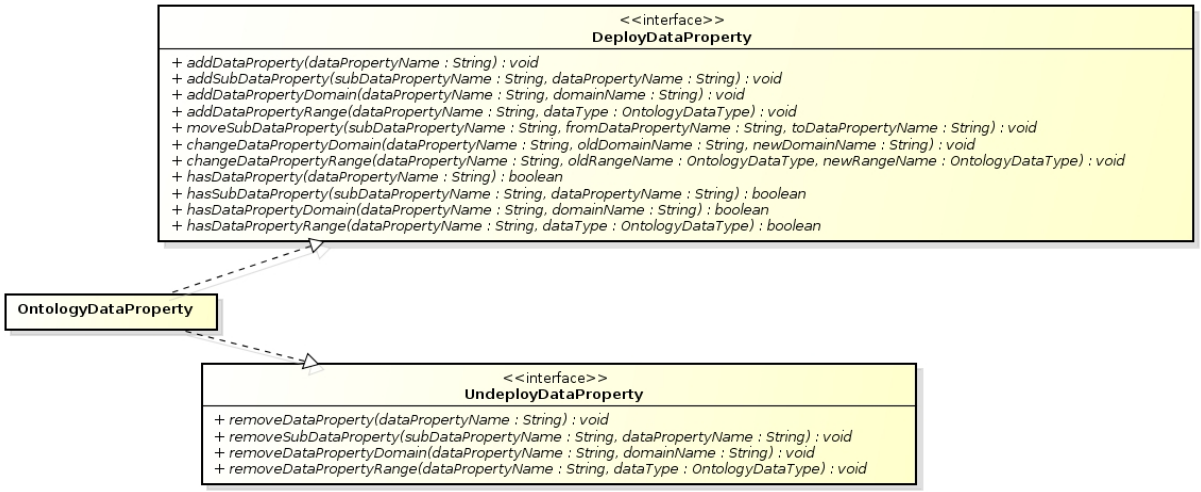



Figura 5.13: Métodos da API envolvendo propriedades de dados.

A Figura 5.16 ilustra a formação de um ciclo na hierarquia de classes. O validador implementa o Algoritmo 1, que recebe o nome da classe e da subclasse envolvidas na inclusão e detecta a formação de ciclo na inclusão de uma subclasse. O mesmo algoritmo também se aplica para a detecção de ciclo nas hierarquias de propriedades.

### Disjunção de classes

A disjunção de classes é um axioma que diz que se $\alpha$ e $\beta$ são classes disjuntas então não podem haver indivíduos que sejam instâncias de $\alpha$ e $\beta$ simultaneamente. Entretanto, a OWL API permite a inclusão do axioma que diz que uma classe seja disjunta dela mesma. Esse tipo de axioma levaria a uma inconsistência na ontologia, uma vez que, conforme a definição de disjunção, não seria possível criar instância para esse tipo de classe. A fim de evitar anomalias dessa natureza, foi implementado um validador que é chamado a cada inclusão do construtor $disjointWith$ da linguagem OWL. Desse modo, o validador verifica se as classes são distintas e garante a regra 3 do modelo. O mesmo procedimento vale para a disjunção de propriedades de dados e objetos.

### Redundância de subclasses

Ao incluir na ontologia os axiomas de subclasse, subpropriedade de dados e subpropriedade de objetos, não há garantias de que a informação já não esteja presente de forma indireta. A redundância não causa uma inconsistência na ontologia mas pode ser indesejável na medida em que não agrega nenhuma informação na base de conhecimento, ocupando espaço em memória de maneira desnecessária. Para evitar esse tipo de anomalia, foi implementado um validador que é chamado a cada inclusão de subclasse.

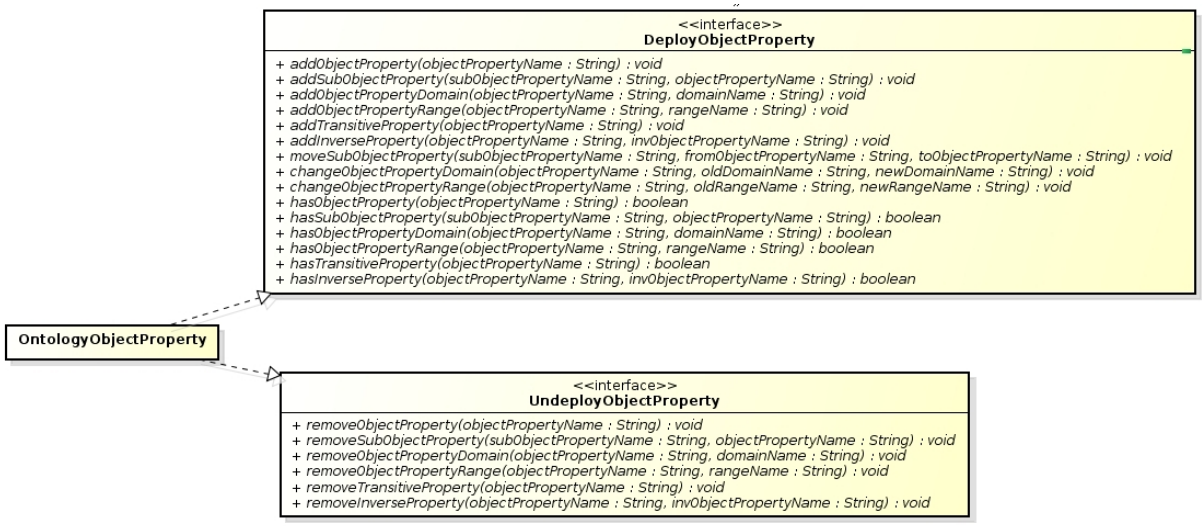



Figura 5.14: Métodos da API envolvendo propriedades de objetos.

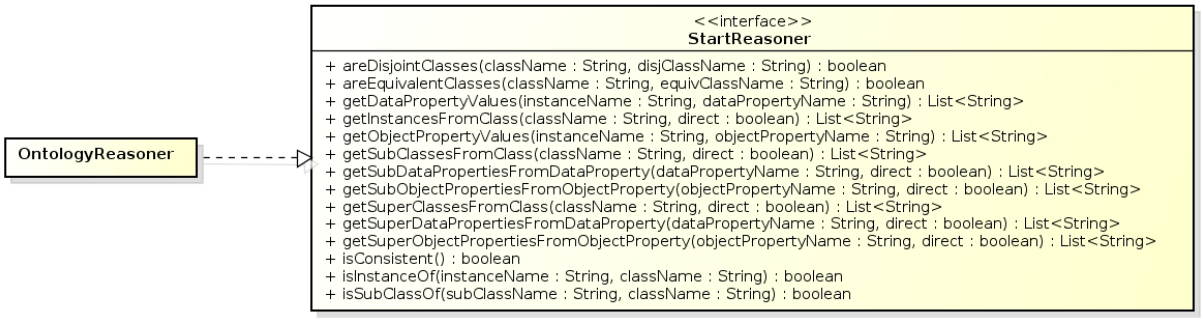



Figura 5.15: Métodos da API envolvendo consultas à ontologia.

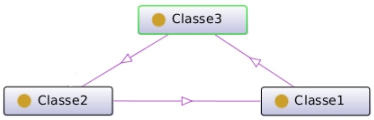



Figura 5.16: Exemplo de formação de ciclo na hierarquia de classes.

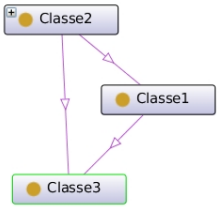



Figura 5.17: Exemplo de redundância na hierarquia de classes.

**Algoritmo 2**: Verifica se a inclusão do axioma que diz que $\gamma$ é subclasse de $\delta$ gera redundância.

**Entrada**: $\gamma$: subclasse; $\delta$: classe.
**Saída**: Valor booleano indicando se a inclusão do axioma gera redundância.

```
1  Π ← superclasses diretas e indiretas(δ);
2  Γ ← superclasses diretas e indiretas(γ);
3  Λ ← superclasses diretas(γ);
4  forall β ∈ Γ do
5      if β = δ then
6          return true ;
7      end
8      if β ∈ Λ then
9          forall α ∈ Π do
10             if β = α  &  β ≠ Thing then
11                 return true ;
12             end
13         end
14     end
15 end
16 return false ;
```

A Figura 5.17 ilustra a redundância na hierarquia de classes. Observe que existem dois caminhos indicando que a Classe3 é subconjunto da Classe2. O validador implementa o Algoritmo 2, que recebe o nome da classe e da subclasse envolvidas na inclusão e detecta a redundância na inclusão de uma subclasse. Como todas as classes são conceitualmente subclasses da classe *Thing* é preciso ajustar o algoritmo para não considerar esse caso. O mesmo algoritmo também se aplica para a detecção de redundância nas hierarquias de propriedades.

**Proteção de alterações realizadas pelas aplicações**

Foi criado um mecanismo de proteção para as alterações realizadas pelas aplicações. Esse mecanismo tem como objetivo impedir que os axiomas inseridos por uma aplicação possam ser removidos por outra. Para isso foi acionada à ontologia uma *Annotation* chamada *createdBy*. Essa *Annotation* é acrescentada a cada inserção de axioma na ontologia e serve para identificar a aplicação que criou o axioma. Por meio dessa *Annotation* pode-se controlar as remoções de axiomas e permitir que apenas a aplicação que inseriu o axioma, ou o próprio *middleware* (ex. em caso de redundância), possam removê-lo. Caso mais de uma aplicação tente inserir uma informação são adicionadas várias *Annotations*. Quando uma aplicação pede para remover o axioma, remove-se primeiro a sua respectiva *Annotation*. Caso não exista outra aplicação que esteja utilizando a informação então não haverá mais *Annotations* e o axioma poderá ser removido da ontologia.

## 5.3 Considerações

Neste capítulo foram detalhados os componentes envolvidos no modelo de gerenciamento de contexto. Foi apresentada uma ontologia independente de domínio contendo conceitos básicos para a determinação do contexto em ambientes ubíquos, como entidades envolvidas e suas localizações. O modelo busca satisfazer os requisitos de flexibilidade, simplicidade e compartilhamento de informações. Essas características são contempladas com (i) um Gerenciador de Contexto, que controla a propagação de informações; (ii) uma API, que encapsula a persistência das operações; e (iii) um Gerenciador de Alterações, que provê suporte à evolução de ontologias.

Na Seção 5.2 foram descritos os principais detalhes de implementação da proposta. Entre eles estão a forma de integração com o *middleware uOS* e os componentes do modelo. Com relação à forma de integração, foram propostos mais dois estados ao ciclo de vida das aplicações, *deploy* e *undeploy*. Os estados adicionais servem para que as aplicações possam incluir e remover dados que dificilmente seriam modificados durante a fase execução, como hierarquias de conceitos e propriedades. A interface do *middleware* com as aplicações também teve que ser adaptada para permitir a comunicação com a ontologia armazenada localmente no dispositivo.

A implementação do modelo definido na proposta foi apresentada em seguida. O Gerenciador de Contexto é o componente responsável pelo compartilhamento das informações de contexto. Sendo assim, comunica-se com o *driver* da ontologia para prover informações aos serviços prestados de forma síncrona e assíncrona, conforme a arquitetura DSOA [9]. Além disso, realiza consultas à ontologia local, que podem ser solicitadas diretamente pela aplicação. O segundo componente da proposta é a *Context API*, que contém métodos para as alterações nas classes, instâncias e propriedades de dados e objetos da ontologia. Por fim, foram especificadas as regras do processo de verificação, com indicações do momento de utilização e respectivos algoritmos quando necessário.

O *driver* da ontologia foi desenvolvido segundo os protocolos de comunicação uP [9]. Para que a *Context Engine* fosse adaptada a outras arquiteturas seria preciso alterar o *driver* da ontologia para a nova arquitetura, importar os módulos da *Context API* e o Gerenciador de Alterações.

# Capítulo 6

# Experimentos

Este trabalho envolve a definição de um modelo para o gerenciamento do contexto utilizando ontologias. Neste capítulo, a avaliação do modelo e sua implementação é realizada de forma quantitativa e por meio de casos de uso. Os casos de uso descrevem situações em que os componentes do modelo são utilizados, com aplicações e cenários envolvendo o modelo proposto. Na avaliação quantitativa, são mostrados os resultados obtidos em termos de tempo de processamento das operações na ontologia.

## 6.1 Aplicações para o modelo proposto

O uso de ontologias para o gerenciamento de informações de contexto em ambientes ubíquos permite que a criação que aplicações capazes de inferir quais informações e serviços são úteis para o usuário em determinado momento. Como forma de avaliação qualitativa, são apresentadas aplicações e cenários de utilização para o modelo proposto. Os cenários incluem inferência de localização e dos serviços disponíveis no ambiente inteligente.

### 6.1.1 Inferência na Localização

A localização é uma das informações mais utilizadas por aplicações sensíveis ao contexto [38]. Esta informação pode ser obtida de várias formas e com níveis de abstração distintos. Por exemplo, se uma pessoa estiver no Departamento de Ciência da Computação da Universidade de Brasília certamente estará na Universidade de Brasília. Do mesmo modo, se a pessoa estiver nas coordenadas (X, Y) de um laboratório contido no Departamento ela ainda estará no Departamento e na Universidade. Isto mostra que a localização pode ser apresentada de formas distintas para as aplicações.

O trabalho publicado em [20], apresenta uma aplicação chamada Nita (*Notes in the Air*). Esta aplicação posta mensagens para uma determinada região. Então qualquer usuário autorizado que "estiver" naquela região estará apto para receber as mensagens. Conforme descrito no trabalho a aplicação comunica-se com um sistema de localização quando é necessário enviar as mensagens aos usuários presentes em uma determinada região geográfica.

No modelo proposto, uma aplicação semelhante poderia ser implementada, com a diferença que uma vez obtida a região geográfica, a informação de localização poderia ser inferida de informações geográficas organizadas de forma ontológica.

Figura 6.1: Cenário para inferência da localização.

**Cenário**

1. Um sensor de localização *indoor* captura as coordenadas de usuário em um laboratório X contido no Departamento de Ciência da Computação da Universidade de Brasília.

2. Um outro sensor captura a localização de um segundo usuário na sala de reuniões Y também no Departamento de Ciência da Computação.

3. Um funcionário do Departamento deseja informar às pessoas presentes no local que haverá limpeza no Departamento e, portanto, as salas deverão ser desocupadas.

4. O funcionário envia a mensagem destinada a toda a região do Departamento.

5. Consultando uma ontologia, os dispositivos dos usuários autorizados recebem a mensagem e inferem que suas localizações (laboratório e sala de reuniões) encontram-se dentro da região de abrangência da mensagem.

6. Depois de realizar o processo de inferência, a aplicação entrega para o usuário a mensagem do funcionário.

Por meio do cenário descrito é possível observar que a utilização de uma ontologia permite que informações sejam organizadas de forma que uma aplicação abstraia a informação adequada no momento de tomar uma decisão. A Figura 6.1 ilustra o cenário apresentado.

Durante o processo de inferência da localização, classes que especificassem um determinado ambiente como sendo *indoor* ou *outdoor* poderiam ser definidas como disjuntas.

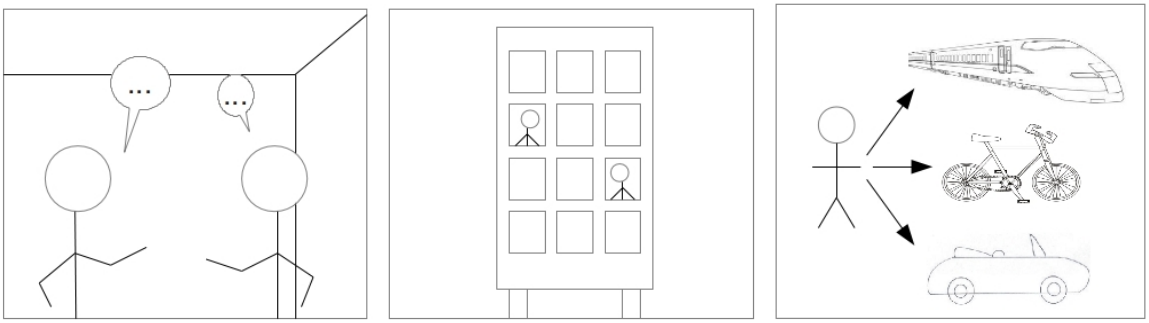
Utilizando a regra 3 do processo de verificação apresentado na Seção 5.1.3, seria possível identificar caso uma aplicação tentasse inserir um local como sendo instância de ambiente *indoor* e *outdoor* ao mesmo tempo, o que resultaria em uma inconsistência.

### 6.1.2 Inferência sobre a Proximidade

A análise das informações de contexto extraídas a partir de um *smart space* pode tornar-se difícil dependendo do contexto no qual essa informação é utilizada. Por exemplo, dados que descrevem a distância entre objetos podem ser insuficientes para que uma aplicação tome decisões. Goiânia, estando a 200 km de Brasília, pode ser considerada uma cidade próxima à capital. Por outro lado, essa mesma medida de distância não poderia ser utilizada para avaliar a proximidade de objetos em um ambiente fechado. Isto deve-se ao fato de que o conceito de proximidade é relativo ao contexto avaliado. Para evitar distorções entre a distância percebida por um sensor e a noção de proximidade, as seguintes regras poderiam ser adotadas para inferência da proximidade de usuários:

- *Talking distance*: uma distância em que os usuários estão próximos, tipicamente em uma mesma sala.
- *Walking distance*: os usuários estão separados por uma distância que pode ser percorrida à pé, por exemplo no mesmo prédio ou em edifícios próximos.
- *Travelling distance*: é o caso contrário, os usuários estão a uma distância que precisa de um meio de transporte para ser percorrida.

Essas regras, combinadas com informações relacionadas à localização do usuário, permitem que uma aplicação tome algumas decisões. A aplicação poderia, por exemplo, iniciar a projeção da pauta de uma reunião quando os participantes entrassem na sala de reunião (*talking distance*). A segunda regra poderia ser usada para apresentar dispositivos possivelmente fora do alcance visual do usuário, porém a uma *walking distance*, como impressoras, telas de projeção, entre outros. Por fim, a terceira regra poderia ser utilizada para sugerir um meio de transporte a um usuário que estivesse a uma *travelling distance* de seu destino. A Figura 6.2 ilustra possíveis cenários envolvendo as três regras.



(a) *Talking Distance.* (b) *Walking Distance.* (c) *Travelling Distance.*

Figura 6.2: Cenários com as regras de proximidade *Talking*, *Walking* e *Travelling*.

### 6.1.3 Inferência nos Serviços

Conforme descrito em 4.2, na arquitetura DSOA, os serviços providos pelos recursos do *smart space* são disponibilizados por meio de *drivers* e utilizados pelas aplicações. Para que uma aplicação utilize um serviço ela precisa informar o nome do *driver* que disponibiliza o serviço. O recurso que possuir o *driver* envia para a aplicação os serviços que ele oferece e a forma de utilização desses serviços.

Uma determinada aplicação pode, por exemplo, requerer a utilização de um *pointer*, interface que permite o usuário apontar objetos em um *display*. Suponha que existam vários *mouses* no ambiente, cada um com suas peculiaridades, como número de botões, *scroll*, etc. Além de tipos diferentes de *mouse*, o ambiente também dispõe de outros recursos que servem de *pointer* tais como canetas digitais e uma câmera conectada à um sistema de reconhecimento de gestos.

Apesar do conjunto de serviços oferecido por cada recurso apresentar diferenças (ex. a caneta digital pode não ter o equivalente ao *scroll* do mouse), existe em comum a funcionalidade *pointer*, que é a funcionalidade requerida pela aplicação.

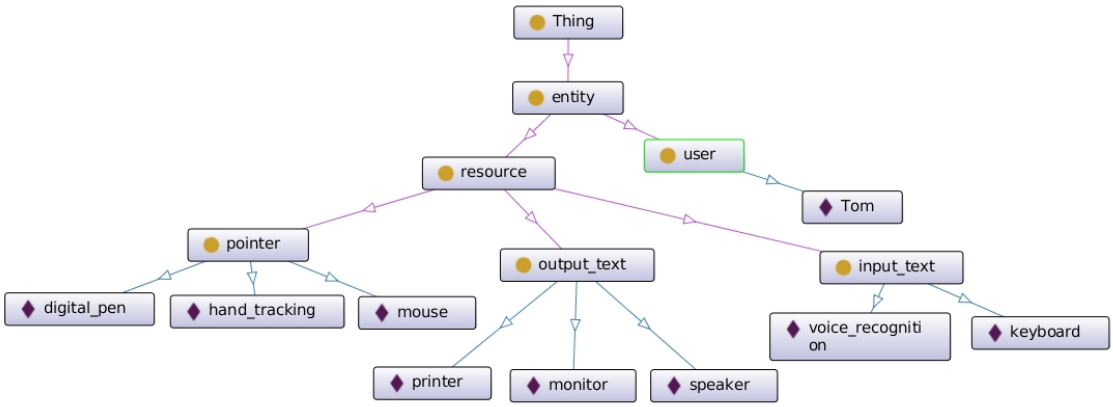



Figura 6.3: Exemplo de hierarquia de serviços baseada na funcionalidade do recurso. (Figura gerada usando Protégé [46])

No modelo proposto, uma hierarquia de dispositivos poderia ser representada em uma ontologia. A Figura 6.3 mostra um exemplo de uma modelagem desse tipo de hierarquia. A organização dos dispositivos em uma hierarquia possibilita inferir, por exemplo, que dada a necessidade de um *pointer*, um *mouse*, uma caneta digital ou uma câmera conectada à um sistema de reconhecimento de gestos poderiam ser utilizados. Assim a aplicação solicitaria um *pointer* e uma consulta na ontologia, realizada de forma conjunta a uma pesquisa dos serviços presentes no *smart space*, retornaria as possibilidades que a aplicação poderia utilizar no momento.

#### Cenário

1. Uma aplicação requer um *pointer* para que o usuário possa apontar dados contidos em uma apresentação em *slides*.

2. A aplicação consulta na ontologia possíveis dispositivos que ofereçam a funcionalidade *pointer*, depois faz um cruzamento entre essas informações e a lista de *drivers* que o *smart space* dispõe no momento.

3. Os possíveis *pointers* encontrados são caneta digital, reconhecimento de gestos (com o serviço de *hand tracking*) e *mouse*.

4. A aplicação consulta as preferências de interface do usuário e verifica que o reconhecimento de gestos é utilizado com frequência pelo usuário.

5. A aplicação seleciona o sistema de reconhecimento de gestos e apresenta um sinal aviso.

6. O usuário aponta os dados da apresentação em *slides* utilizando o serviço de *hand tracking*.

Por meio do cenário descrito é possível observar que a organização das informações em uma ontologia permite a existência de uma camada de abstração entre os recursos disponíveis e a funcionalidade (entrada de texto, *pointer*, etc.) que estes recursos representam para o usuário.

Considerando que os dispositivos entram e saem do *smart space* de forma dinâmica, uma hierarquia de recursos poderia ser criada em tempo real, com cada recurso informando qual/quais classes ele pertence e quais são suas superclasses. No momento em que o dispositivo (ex. *mouse*) fosse descoberto no ambiente, as suas respectivas classes (ex. *pointer*) e superclasses (ex. *resource*) seriam indicadas para inserção na ontologia. A regra 1 do processo de verificação apresentado na Seção 5.1.3 checaria se já não existe uma outra classe com o mesmo identificador. Caso a classe referente ao recurso já estivesse presente, então não seria necessário inserir novamente. A regra 2 verifica se a inclusão da *subclasse(recurso filho, recurso pai)* não gera ciclo e a regra 4 verifica a ocorrência de redundância. Por fim, uma instância para cada recurso seria inserida na ontologia. Utilizando a regra 5, as classes dessa hierarquia seriam removidas apenas quando não houvesse instância que as representassem, ou seja, quando o *smart space* não tivesse mais nenhum dispositivo que contendo a funcionalidade representada pela classe.

### 6.1.4 Um Protótipo de Aplicação

Foi implementado um protótipo de uma aplicação para mostrar o funcionamento da inferência dos serviços. Os conceitos da ontologia de serviços são os ilustrados pela Figura 6.3. Além desses conceitos, foram acrescentadas na ontologia duas propriedades de dados:

- *Link* da imagem: caminho até o arquivo de imagem.
- Nome do *driver*: nome do *driver* do recurso.

O nome do *driver* é usado para verificar se o recurso encontra-se disponível. Para obter esta informação a aplicação consulta a lista de *drivers* que o *middleware* dispõe no momento. Se estiver presente, marca o campo disponível como verdadeiro; falso caso contrário. A Figura 6.4 apresenta a interface de busca implementada. Observe que o usuário busca pela funcionalidade de apontamento (*pointer*) e a aplicação retorna, realizando uma consulta na ontologia, os possíveis recursos que atenderiam essa funcionalidade.

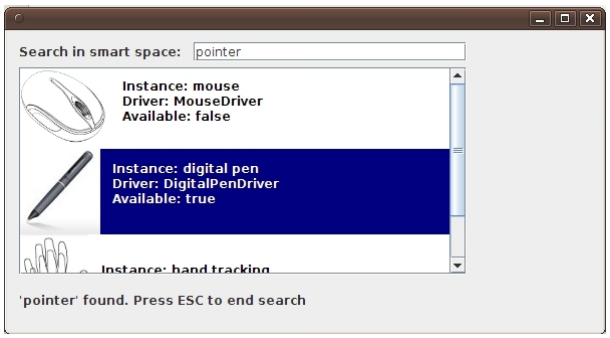



Figura 6.4: Busca por funcionalidade.

Os principais mecanismos de inferência que são utilizados nesta aplicação são:

- *Subsumption*: Dada uma ontologia $O$ e duas classes $A$ e $B$, verifica se $A$ é um subconjunto de $B$ em todos os modelos[1] de $O$.
- Checagem de instância: Dada uma ontologia, uma instância $a$ e uma classe $A$, verifica se $a$ é uma instância de $A$ em todos os modelos de $O$.

O mecanismo de *Subsumption* pode ser utilizado para identificar quais funcionalidades estão presentes no ambiente. O resultado de uma consulta à *resource* incluiria as funcionalidades *pointer*, *output text* e *input text*. Já a Checagem de Instância é utilizada, por exemplo, no caso em que o usuário informa a funcionalidade *pointer*, ilustrado na Figura 6.4, e obtém como retorno as instâncias desta classe na ontologia, *mouse*, *digital pen* e *hand tracking*.

## 6.2 Custo de processamento

Para avaliar o impacto da introdução de ontologias no *middleware uOS* foi conduzido um conjunto de experimentos envolvendo as seguintes operações: (1) consultas à ontologia e (2) inserção de instâncias. Esses experimentos forneceram a oportunidade de medir o *overhead* computacional que o processamento de ontologias causa ao *uOS*. Todos os experimentos foram conduzidos em um *notebook* Lenovo (3Gb RAM com um Dual-Core T4300 2.10GHz). O sistema operacional utilizado foi o Ubuntu 11.4 com Máquina Virtual Java versão 1.6.

[1]Um modelo de $O$ é uma atribuição de valores (interpretação) em $O$ que assume valor verdadeiro.

### 6.2.1 Consultas

As informações que, especificamente, irão determinar a ontologia de contexto são imprevisíveis, posto que dependem das aplicações sensíveis ao contexto. Desse modo, foi criado um cenário de contexto com o escopo de um *smart space*. O cenário é um laboratório contendo os seguintes dispositivos:

- 1 impressora;
- 1 monitor grande;
- 1 câmera;
- 1 ar condicionado;
- 1 controlador de luminosidade;
- 10 computadores *desktop*.

Adicionalmente,este laboratório possui 15 usuários. Organizando essas informações de contexto em uma ontologia semelhante à mostrada na Figura 6.5, é possível estimar o número de entidades que representariam o cenário e assim, medir o tempo de processamento das consultas à ontologia. Por simplicidade, apenas classes e instâncias serão consideradas.

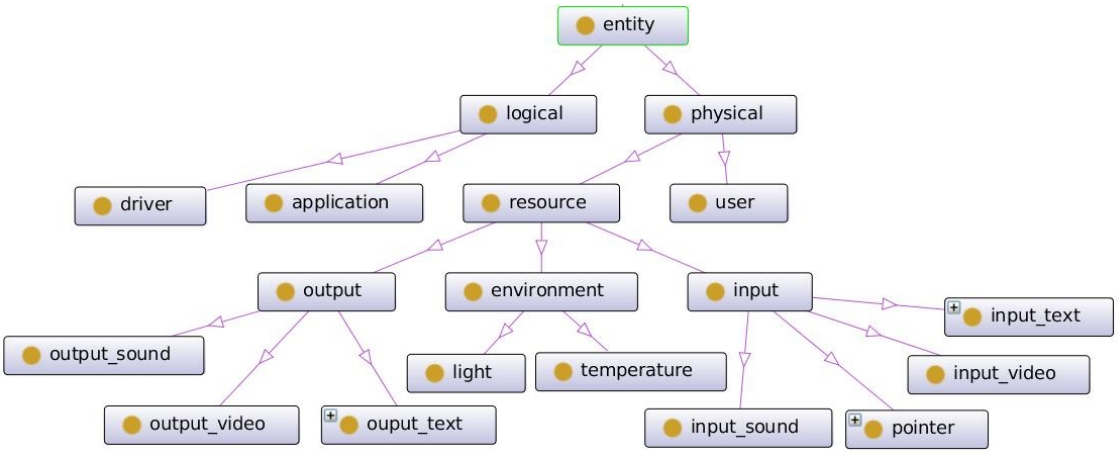



Figura 6.5: Hierarquia de classe organizada pela funcionalidade do recurso. (Figura gerada usando Protégé [46]).

Considerando que computadores *desktop* fornecem instâncias de *output video*, *input text* e *pointer*, os dispositivos presentes no laboratório resultariam em aproximadamente 50 instâncias. Além disso, a classe *driver* tem, pelo menos, uma instância para cada tipo de recurso, adicionando 8 instâncias. O tempo medido para inserir as 77 entidades mostradas na Tabela 6.1 usando a *Context API* foi de 0.26 segundos.

A Tabela 6.2 mostra o tempo médio de processamento de cinco tipos de consultas à ontologia utilizada no experimento. Nas 3 primeiras consultas de classe, a média calculada baseou-se no processamento das 19 classes da ontologia. Os tempos medidos foram: 3

| Entidade | Quantidade |
|---|---|
| *Class* | 19 |
| *Instance* | 58 |

Tabela 6.1: Número estimado de classes e instâncias de uma ontologia para um laboratório inteligente.

ms para trazer as instâncias de uma classe, 1.4 ms para consultar as subclasses e 1.8 ms para as superclasses. A quarta consulta verifica a relação de hierarquia entre as classes (*subsumption*). Sem perda de generalidade, foi selecionada uma classe para ser consultada em relação às demais, tendo 1.2 ms como resultado em média. Para a entidade instância, sem perda de generalidade, uma entidade foi selecionada e consultada sobre as outras 19 classes, resultando na média de 3.3 ms.

| Entidade | Operação | Tempo de processamento |
|---|---|---|
| *Classe* | *getInstancesFromClass* | 3.060 ms |
| | *getSubClassesFromClass* | 1.394 ms |
| | *getSuperClassesFromClass* | 1.809 ms |
| | *isSubClassOf* | 1.179 ms |
| *Instância* | *isInstanceOf* | 3.285 ms |

Tabela 6.2: Tempo médio de processamento das consultas na ontologia.

### 6.2.2 Inserção de instâncias

Para avaliar o impacto da inserção de instâncias na ontologia, primeiramente mediu-se o tempo que o *middleware* leva para iniciar uma aplicação, sem que houvesse qualquer operação na ontologia. Depois foi medido o tempo que a aplicação leva para inserir 100 instâncias em uma ontologia. A Tabela 6.3 mostra os resultados obtidos. $T_{middleware}$ representa o tempo que o *middleware* leva desde o início da sua execução até o inicio da execução de uma aplicação. $T_{total}$ representa o tempo gasto entre o início da execução do *middleware* até o término de 100 operações de inserção de instância na ontologia. Por fim, $T_{ontologia}$ é o tempo necessário para realizar as 100 operações de inserção de instância na ontologia. O tempo para a inserção de apenas 1 instância na ontologia foi de 19 ms.

| $T_{middleware}$**(ms)** | $T_{total}$**(ms)** | $T_{ontologia}$**(ms)** |
|---|---|---|
| 2020 | 2125 | 108 |

Tabela 6.3: Tempo de processamento de uma aplicação no *middleware uOS*.

A Figura 6.6 mostra o crescimento do tempo de processamento de acordo com o número de operações de adição de instância na ontologia. Os dados mostram o número de instâncias sendo incrementado de 10 em 10 até chegar a 100. Observou-se que o tempo médio para inserção de instâncias é minimizado quando são inseridas múltiplas

instâncias em uma mesma operação de gravação. O custo para inserção das 77 entidades mostradas na Tabela 6.1 foi de 0.26 s, bastante superior ao tempo de inserção de instâncias apresentado na Tabela 6.3. Essa diferença ocorre pois no experimento descrito em 6.2.1 existem relações de hierarquia entre as classes e as instâncias são atribuídas a classes distintas, conforme o ambiente modelado.

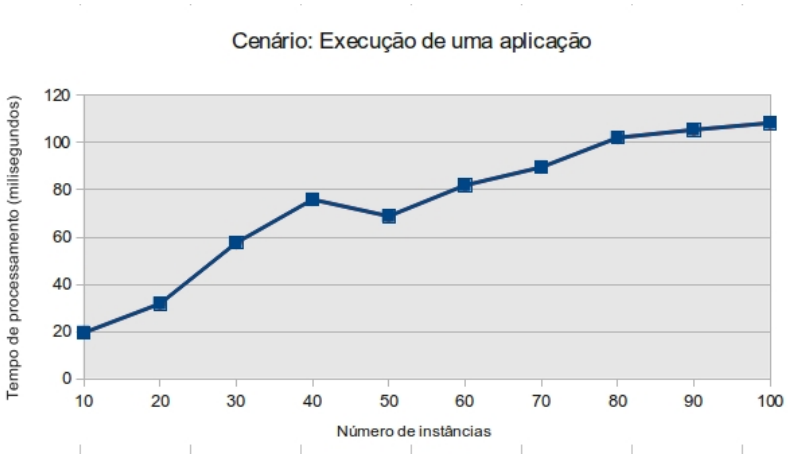



Figura 6.6: Custo de inserção de instâncias na ontologia.

## 6.3 Considerações

Neste capítulo foram apresentados casos de uso para a infra-estrutura de gerenciamento de informações de contexto apresentada neste trabalho. Os casos de uso foram construídos para prover uma avaliação empírica das vantagens de usar ontologias para representar contexto, como a realização de inferências sobre a localização, proximidade e serviços disponíveis no ambiente. O protótipo apresentado contém inferência de serviços e tem como objetivo a tradução da funcionalidade requerida pelo usuário em recursos disponíveis no ambiente. A implementação utiliza uma ontologia de serviços, que retorna informações para uma aplicação executando no *middleware uOS*.

Além de prover mecanismos de inferência, o modelo permite que as informações coletadas do ambiente possam ser compartilhadas pelas aplicações que estiverem executando em um mesmo dispositivo. Por exemplo, a localização utilizada para a inferência do destinatário de uma mensagem (cenário descrito em 6.1.1) também poderia ser usada para determinação da proximidade. Do mesmo modo, os serviços disponíveis no *smart space* poderiam ser filtrados de acordo com a localização e a proximidade. No modelo, os dados buscados do ambiente por uma aplicação podem ser reutilizados por outra por meio de consultas à ontologia providas pela *Context API*. Isso contribui para que as aplicações tenham acesso a um conjunto maior de informações de contexto, sem a necessidade de buscar os dados do ambiente novamente.

Para avaliação quantitativa da implementação do modelo foram realizados experimentos com o objetivo de estimar o impacto da introdução de ontologias no *middleware uOS*. Experimentos mediram o tempo de processamento médio de consultas à ontologia. Foi

construído um cenário para estimar o número de classes e instâncias envolvidas. Os resultados mostraram que as consultas levaram menos de 4 milisegundos. Isto representa um *overhead* de 8% em relação à 50 ms, que é considerado o limite de tempo em que o ser humano não percebe que houve atraso [43]. Os resultados relacionados à inserção de instância mostraram um crescimento linear do custo de inserção de instâncias na ontologia. Considerando um cenário em que a inclusão de informações ocorre de maneira esparsa, a medida que novos eventos acontecem no *smart space*, temos que a utilização de uma ontologia para armazenamento dessas informações mostra-se como uma solução viável para aplicações de tempo real.

# Capítulo 7

# Conclusões

Um ambiente ubíquo é formado por várias aplicações sensíveis ao contexto, as quais necessitam de informações para a tomada de decisão. A fim de possibilitar o compartilhamento e reúso dessas informações, é necessário armazená-las, para que possam ser recuperadas quando necessário. Dentre as formas de armazenamento de contexto, destacam-se as ontologias, por permitirem o uso de técnicas de inferência e a verificação da consistência, o que contribui para a diminuição da ocorrência de erros nas decisões tomadas pelas aplicações.

O desenvolvimento de aplicações sensíveis ao contexto traz grandes desafios, como os relatados na Seção 2.2. Conforme visto nos trabalhos correlatos, algumas iniciativas de utilização de ontologias em ambientes sensíveis ao contexto foram realizadas. Contudo, a maneira como os *middlewares* em ambientes ubíquos gerenciam essas ontologias ainda apresenta-se como potencial foco de investigação. Neste trabalho foi apresentado um modelo para o gerenciamento das informações de contexto, abrangendo todo o fluxo pelo qual as informações atravessam até serem propagadas ao nível da aplicação.

Conforme visto no Capítulo 5, o trabalho propõe um modelo contendo três componentes básicos: (i) um Gerenciador de Contexto; (ii) a definição da *Context API* e (iii) um Gerenciador de Alterações. O Gerenciador de Contexto controla o fluxo das informações de contexto durante a execução das aplicações. Além disso, permite o compartilhamento de informações entre aplicações sensíveis ao contexto. A *Context API* foi definida para viabilizar as alterações e padronizar o formato das operações realizadas. Deixar a cargo dos desenvolvedores a responsabilidade de lidar com a persistência da ontologia, assim como aspectos relacionados a dependência de informações e consistência, tornaria muito complexa a tarefa de evoluir uma ontologia. O desenvolvedor da aplicação não deve precisar conhecer o nome da ontologia nem onde está armazenada. Por fim, o Gerenciador de Alterações é o componente responsável por verificar a consistência da ontologia, utilizando o processo de verificação definido na Seção 5.1.3. Essa verificação deve ser feita pelo *middleware*, de modo a evitar que a inclusão de alterações na ontologia afete outras aplicações. Portanto, faz-se necessário agregar funções de gerenciamento ao *middleware*, o qual encarrega-se de controlar alterações na ontologia.

A implementação do modelo, apresentada no Capítulo 5.2, incorpora gerenciamento de contexto ao *middleware uOS* por meio da utilização de ontologias. A Tabela 7.1 apresenta os serviços para suporte às aplicações sensíveis ao contexto oferecidos pelo *middleware* uOS e o que foi incluído neste trabalho com a implementação da *Context Engine*.

| Serviços de suporte às aplicações | uOS | *Context Engine* |
|---|---|---|
| Serviço de inscrição e entrega de informações (SIE) | x | |
| Consulta e compartilhamento das informações (CCI) | x | x |
| Serviço de processamento das informações (SPI) | | x |
| Serviço de verificação das informações de contexto (SVI) | | x |
| Descoberta e gerenciamento de serviços (DGS) | x | |

Tabela 7.1: Serviços de suporte às aplicações.

| Modelos | SIE | CCI | SPI | SVI | DGS |
|---|---|---|---|---|---|
| CoBrA [10] | - | - | x | x | - |
| SOCAM [23, 22] | x | x | x | - | x |
| ONTO-MoCA [56, 54] | x | - | x | x | x |
| CANDEL [34] | - | x | x | - | x |
| *Smart* M3 [40] | - | x | x | - | x |
| uOS [9] com *Context Engine* | x | x | x | x | x |

Tabela 7.2: Modelos para o gerenciamento de informações de contexto e serviços providos às aplicações.

O *middleware uOS* suporta o gerenciamento de eventos, realiza a descoberta de dispositivos e serviços e provê *plugins* de rede e por *proxies* para a comunicação entre os nós. Este trabalho complementa o serviço de consulta e compartilhamento das informações, disponibilizando consultas a ontologias locais e remotas. O serviço de processamento das informações é obtido por meio da representação em forma de ontologia, que permite consultas utilizando mecanismos de inferência. Por fim, o serviço de verificação das informações de contexto é provido pelo Gerenciador de Alterações, que verifica as solicitações de mudança na ontologia.

A Tabela 7.2 mostra o comparativo entre os modelos para o gerenciamento de informações de contexto citados e os serviços que eles oferecem às aplicações. Nesta tabela estão marcados com 'x' as características encontradas na referência de cada modelo de gerenciamento de contexto, significando apenas que os valores em branco tratam-se de características não tratadas nestas referências. Conforme visto na Tabela 7.2, o *middleware uOS* juntamente com o modelo apresentado, contempla os serviços listados em [16].

Este trabalho aborda dois desafios relacionados ao contexto: heterogeneidade das informações e compartilhamento. A utilização de ontologias permite que informações heterogêneas sejam representadas nas hierarquias de classes e propriedades. O compartilhamento é obtido de acordo com a arquitetura DSOA [9], que provê a forma assíncrona, por meio de eventos, e a forma síncrona, utilizando chamadas de serviço. A avaliação foi realizada de duas formas: (1) utilizando casos de uso envolvendo os componentes da proposta e (2) quantitativamente, com estimativas do tempo médio de processamento de consultas e inserção de instâncias.

Baseando-se no modelo apresentado e em sua implementação temos que este trabalho possui as seguintes características:

1. ontologias são consideradas recursos do *smart space*;
2. dispositivos com baixa capacidade de memória e processamento podem realizar consultas em dispositivos remotos, capazes de armazenar bases de conhecimento representadas em ontologias;
3. a comunicação entre dispositivos para compartilhamento de informações de contexto pode ser síncrona ou assíncrona;
4. alterações na ontologia são controladas pelo *middleware*, o qual provê suporte e coordena as aplicações sensíveis ao contexto;
5. informações de contexto (como localização) armazenadas na ontologia podem ser utilizadas por múltiplas aplicações executando no mesmo dispositivo, sem a necessidade de replicar a informação entre as aplicações.

Em [34] cada CPC (*Context Proxy Component*) contém uma pequena ontologia, que descreve uma informação de contexto. Para consultar uma informação descrita na ontologia, a aplicação utiliza um serviço, que por sua vez chama um ou mais CPCs. Conforme descrito no item 1, neste trabalho a ontologia é um recurso do ambiente e pode ser consultada pelas aplicações diretamente. Esta simplificação da arquitetura flexibiliza a forma como as aplicações acessam as informações de contexto representadas na ontologia e permite que sejam realizadas inferências abrangendo todo o conjunto de informações de contexto contido em um dispositivo, ao invés de limitar-se à ontologia contida em um CPC.

Em relação aos itens 2 e 3, o trabalho difere-se da estratégia apresentada em [67] no sentido de que as informações em [67] chegam a dispositivos remotos apenas por meio de eventos, enquanto esta solução permite consultas síncronas e assíncronas, utilizando o *driver* da ontologia. Conforme o item 4, a solução apresentada controla as alterações na ontologia e inclui uma regra que guarda na ontologia qual aplicação inseriu determinada informação. Isto evita que uma aplicação remova dados previamente inseridos por outra aplicação. Não foi encontrado nos trabalhos correlatos nenhuma abordagem neste sentido.

Conforme exposto no item 5, nesta solução as aplicações contidas em um mesmo dispositivo podem inserir e remover dados na ontologia de forma conjunta. A principal vantagem é que os dados inseridos por uma aplicação (ex. localização, preferências, etc) são compartilhados por todas as aplicações do dispositivo. Porém, isso traz como desvantagem a necessidade da criação de mecanismos para controle das alterações realizadas pelas aplicações. Além disso, o compartilhamento de informações entre as aplicações pode não ser desejável. Caso as aplicações não tenham informações em comum, consultar uma base conjunta pode aumentar o tempo de processamento das consultas de forma desnecessária, uma vez que a consulta seria feita sobre toda a base de informações.

## 7.1 Limitações do trabalho

A implementação do modelo proposto para o gerenciamento de contexto baseado em ontologias possui algumas limitações. Embora haja mecanismos para consultar dispositivos remotos na comunicação síncrona, a forma de interação assíncrona é limitada a três tipos de eventos (*InstanceOf*, *DataProperty* e *ObjectProperty*), conforme descrito na Seção

5.2.2. Os eventos que podem ser monitorados pelas aplicações são aqueles relacionados a instâncias, pois, em geral, as instâncias são mais suscetíveis a alterações de contexto do que as hierarquias de conceitos e propriedades.

Outra limitação é que neste trabalho adotamos uma abordagem procedural para a fase de alteração do processo de evolução da ontologia. Conforme discutido em [61], nesta abordagem um desenvolvedor de aplicação precisa especificar a maioria dos passos relacionados à estratégia de evolução, isto é, operações como *addSubClass* e *removeObjectProperty* devem ser definidas explicitamente. Por outro lado, uma abordagem declarativa permite que as solicitações de alteração na ontologia sejam especificadas em um maior nível de abstração.

## 7.2 Trabalhos futuros

O trabalho apresentado pode ser estendido em vários aspectos, os quais incluem o desenvolvimento de aplicações que utilizem a representação do contexto em ontologias e a construção de mecanismos que permitam maior controle do *middleware* sobre o gerenciamento da ontologia. As aplicações podem utilizar os mecanismos de inferência da ontologia, realizando inferências sobre a localização de usuários e dispositivos, serviços disponíveis, proximidade de objetos, entre outros; conforme visto nos casos de uso apresentados na Seção 6.1. Com relação ao gerenciamento da ontologia, os seguintes pontos poderiam ser explorados em trabalhos futuros:

- versionamento das alterações na ontologia;
- importação de ontologias já padronizadas, como a desenvolvida no projeto SOUPA [11];
- inclusão de novos métodos na API, que permitam operações envolvendo diversas alterações na ontologia, de forma conjunta;
- análise de novas regras de consistência para as solicitações de alteração na ontologia.

A definição de formas de mapeamento e *merge* de ontologias e a implementação de eventos que indiquem com maior detalhe as condições para que a informação seja transmitida para outro dispositivo, também são tópicos que poderiam ser considerados como formas de extensão do trabalho apresentado.

Além disso, este trabalho não trata aspectos relacionados a segurança e privacidade das informações de contexto. Conforme relatado na Seção 2.2, as aplicações sensíveis ao contexto necessitam de diversos tipos de informações, que podem incluir preferências dos usuários, localização e outros dados que precisam ser protegidos.

## 7.3 Publicações

Parte deste trabalho foi publicada no seguinte artigo:

- "Um modelo para o gerenciamento de ontologias em ambientes ubíquos" - publicado no SBCUP (Simpósio Brasileiro de Computação Ubíqua), evento do CSBC (Congresso da Sociedade Brasileira de Computação) 2011 - Natal (RN).

# Apêndice A

# *Context Engine*

## A.1 Propriedades da *Context Engine*

Esta seção apresenta as propriedades do arquivo "ubiquitous.properties" relacionadas à implementação do modelo proposto.

### A.1.1 Caminho para o arquivo de ontologias

O caminho para o arquivo de ontologias pode ser definido pelo arquivo de propriedade do *middleware uOS*. A propriedade utilizada para definição desse caminho é chamada "ubiquitos.ontology.path". Observe na Listagem A.1 um exemplo de como esta propriedade pode ser definida.

```
ubiquitos.ontology.path=resources/owl/uoscontext.owl
```

Listagem A.1: Exemplo de definição do caminho para o arquivo da ontologia.

### A.1.2 Instânciando a ferramenta *Reasoner*

A implementação da *Context Engine* permite que as ferramentas de inferência, também conhecidas como *reasoners*, possam ser utilizadas de forma customizável. Ou seja, pode-se utilizar qualquer implementação de *reasoner* que implemente a interface especificada pela OWL API. Para isso, foi definida a propriedade "ubiquitos.ontology.reasonerFactory", usada para definir qual classe Java contém o método *ReasonerFactory*, que cria uma instância do *reasoner*. Observe na Listagem A.2 um exemplo de como esta propriedade pode ser definida.

```
ubiquitos.ontology.reasonerFactory=
        br.unb.unbiquitous.ubiquitos.ontology.OntologyReasoner
```

Listagem A.2: Exemplo de definição do *reasoner* da ontologia.

A Listagem A.3 apresenta um exemplo de como o método pode ser implementado, sendo que a classe *OntologyReasoner*, especificada no arquivo de propriedades, deve ser o nome da classe que contém o método *ReasonerFactory()*.

```
public static OWLReasonerFactory ReasonerFactory(){
    return new org.semanticweb.HermiT.Reasoner.ReasonerFactory();
}
```

Listagem A.3: Exemplo de definição do método *ReasonerFactory* que especifica a ferramenta de inferência utilizada, a Hermit.

### A.1.3 Desabilitando a *Context Engine*

Devido a limitações de *hardware*, é possível perceber que nem todos os dispositivos são capazes de suportar o armazenamento de informações de contexto. A fim que manter a flexibilidade do *middleware uOS*, a *Context Engine* foi implementada de modo a tornar possível que suas funcionalidades fossem desabilitadas. Para isso, basta que o arquivo de propriedades "ubiquitous.properties" não contenha a propriedade "ubiquitos.ontology.reasonerFactory", que indica o método a ser chamado para instanciar o *reasoner*.

## A.2 Criando e removendo entidades na ontologia

A criação e remoção de classes, propriedades e instâncias na ontologia guardada pelo *uOS* é realizada por meio de chamadas aos métodos definidos pela *Context API*. O código mostrado na Listagem A.4 gera o OWL ilustrado pela Listagem A.5.

```
public void init(OntologyDeploy ontology) {

        ontology.getOntologyDeployClass().
                        addClass(``Gato'');
        ontology.getOntologyDeployClass().
                        addSubClass(``Gato'', ``Felino'');
        ontology.getOntologyDeployInstance().
                        addInstanceOf(``Garfield'', ``Gato'');
        ontology.getOntologyDeployObjectProperty().
                        addObjectProperty(``hasOwner'');
        ontology.getOntologyDeployDataProperty().
                        addDataProperty(``hasAge'');

    }
```

Listagem A.4: Exemplo de criação de classes, propriedades e instâncias no método *init()*.

```
                        ...
    <Declaration>
        <Class IRI=``#Gato''/>
    </Declaration>
```

```
<Declaration>
    <ObjectProperty IRI="#hasOwner"/>
</Declaration>
<Declaration>
    <DataProperty IRI="#hasAge"/>
</Declaration>
<SubClassOf>
    <Class IRI="#Gato"/>
    <Class IRI="#Felino"/>
</SubClassOf>
<ClassAssertion>
    <Class IRI="#Gato"/>
    <NamedIndividual IRI="#Garfield"/>
</ClassAssertion>
                        ...
```

Listagem A.5: OWL gerado pelo método init() da Listagem.

A Listagem A.6 remove da ontologia as entidades criadas no método *init()* mostrado na Listagem A.4.

```
public void tearDown(OntologyUndeploy ontology)
                throws Exception {

        ontology.getOntologyUndeployClass().
                        removeClass("Gato");
        ontology.getOntologyUndeployClass().
                        removeSubClass("Gato", "Felino");
        ontology.getOntologyUndeployInstance().
                        removeInstanceOf("Garfield", "Gato");
        ontology.getOntologyUndeployObjectProperty().
                        removeObjectProperty("hasOwner");
        ontology.getOntologyUndeployDataProperty().
                        removeDataProperty("hasAge");

    }
```

Listagem A.6: Exemplo da remoção de classes, propriedades e instâncias no método *tearDown()*.

## A.3 Realizando consultas com o *Reasoner*

Esta seção mostra alguns exemplos de consultas realizadas sobre a ontologia pelo *reasoner*. A Listagem A.7 mostra algumas informações sendo salvas na ontologia.

```
ontology.getOntologyInstance()
        .addInstanceOf(``sala_1'', ``room'');
ontology.getOntologyInstance()
        .addInstanceOf(``Lucas'', ``user'');
ontology.getOntologyInstance()
        .addObjectPropertyAssertion(``Lucas'',
        ``isEntityInLocation'', ``sala_1'');
ontology.getOntologyInstance()
        .addInstanceOf(``piso_1'', ``floor'');
ontology.getOntologyInstance()
        .addObjectPropertyAssertion(``sala_1'',
        ``isRoomInFloor'', ``piso_1'');
ontology.getOntologyInstance()
        .addObjectPropertyAssertion(``piso_1'',
        ``isFloorInBuilding'', ``predio_1'');
ontology.saveChanges();
```

Listagem A.7: Salvando informações na ontologia.

As Listagens A.8, A.9, A.10, A.11, A.12, A.13, A.14, apresentam alguns exemplos de como certas informações poderiam ser extraídas da ontologia com base no que foi informado pela Listagem A.7.

```
Consulta no sistema: ontology.getOntologyReasoner()
        .hasObjectProperty(``Lucas'',``isLocatedIn'',``sala_1'');
Console: true
```

Listagem A.8: Pergunta: "Lucas está na sala 1?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .hasObjectProperty(``sala_1'',``isRoomInFloor'',
        ``piso_1'');
Console: true
```

Listagem A.9: Pergunta: "Sala 1 está no piso 1?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .hasObjectProperty(``sala_1'',``isLocatedIn'',
        ``predio_1'');
Console: true
```

Listagem A.10: Pergunta: "Sala 1 está no prédio 1?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .hasObjectProperty(``Lucas'',``isLocatedIn'',
        ``piso_1'');
Console: true
```

Listagem A.11: Pergunta: "Lucas está no piso 1?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .hasObjectProperty(``Lucas'',``isLocatedIn'',
        ``predio_1'');
Console: true
```

Listagem A.12: Pergunta: "Lucas está no predio 1?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .getObjectPropertyValues(``Lucas'', ``isLocatedIn'');
Console: [sala_1, piso_1, predio_1]
```

Listagem A.13: Pergunta: "Onde está Lucas?" .

```
Consulta no sistema: ontology.getOntologyReasoner()
        .getObjectPropertyValues(``sala_1'', ``isLocationOf'');
Console: Lucas
```

Listagem A.14: Pergunta: "Quem está na sala 1?" .

## A.4 *Driver* da ontologia

Esta seção apresenta exemplos de utilização do *driver* da ontologia. A Listagem A.15 ilustra um trecho de código que chama um dos serviços disponíveis pelo *driver* da ontologia de forma síncrona.

```
public void start(Gateway gateway, OntologyStart ontology) {
        try {
                Map<String,String> parameters =
                        new HashMap<String,String>();
                parameters.put(``instanceName'', ``Garfield'');
                parameters.put(``className'', ``Gato'');
                ServiceResponse response =
                gateway.callService(null, ``isInstanceOf'',
                ``br.unb.unbiquitous.uos.driver.OntologyDriver'',
                ``My_ontology_driver'', null, parameters);
                JSONObject isInstanceOf =
                        new JSONObject(response.getResponseData()
```

```
                                        .get("isInstanceOf"));
            System.out.println(``Garfield is instance of Gato:''
                               + isInstanceOf.get(``queryResult''));
            } catch (Exception e) {
                    e.printStackTrace();
            }
}
```

Listagem A.15: Exemplo de utilização de serviços síncronos disponíveis pelo *driver* da ontologia.

# Referências


[1] The description logic handbook: Theory, implementation, and applications. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003. 16

[2] F. Baader, I. Horrocks, and U. Sattler. Description logics. *Foundations of Artificial Intelligence*, 3:135–179, 2008. 14, 15, 16

[3] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007. 7, 8

[4] P. Bellavista, A. Corradi, R. Montanari, and C. Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Transactions on Software Engineering*, 29:1086–1099, 2003. 1, 23, 25

[5] P.A. Bernstein. Middleware: a model for distributed system services. *Communications of the ACM*, 39(2):86–98, 1996. 1

[6] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *ACM SIGOPS Operating Systems Review*, 21(5):123–138, 1987. 9

[7] M. Brown. Supporting user mobility. In *International Federation for Information Processing*. Citeseer, 1996. 5

[8] F. Buzeto. Um conjunto de soluções para a construção de aplicativos de computação ubíqua. Master's thesis, Departamento de Ciência da Computação, Universidade de Brasília, 2010. 3, 32, 34

[9] F. Buzeto, C. de P. Filho, C. Castanho, and R. Jacobi. Dsoa: A service oriented architecture for ubiquitous applications. In P. Bellavista, R. Chang, H. Chao, S. Lin, and P. Sloot, editors, *Advances in Grid and Pervasive Computing*, volume 6104 of *Lecture Notes in Computer Science*, pages 183–192. Springer Berlin / Heidelberg, 2010. 1, 3, 24, 31, 34, 39, 49

[10] H. Chen, T. Finin, and A. Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(03):197–207, 2003. xi, 18, 49

[11] H. Chen, F. Perich, T. Finin, and A. Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004. the First Annual International Conference on*, pages 258–267. Ieee, 2004. 2, 18, 21, 51

[12] A. Church. Introduction to logic and to the methodology of deductive sciences, 1941. 14

[13] A. Corradi, M. Fanelli, and L. Foschini. Implementing a scalable context-aware middleware. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 868–874. IEEE, 2009. 2, 9, 23

[14] O. Davidyuk, J. Riekki, V.M. Rautio, and J. Sun. Context-aware middleware for mobile multimedia applications. In *Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 213–220. ACM, 2004. 23

[15] A. K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7, 2001. 5

[16] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum.-Comput. Interact.*, 16(2):97–166, December 2001. 1, 22, 25, 49

[17] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In *CHI 2000 workshop on the what, who, where, when, and how of context-awareness*, volume 4, pages 1–6. Citeseer, 2000. 5

[18] I.A. Essa. Ubiquitous sensing for smart and aware environments. *Personal Communications, IEEE*, 7(5):47–49, 2000. 7

[19] T. Flury, G. Privat, and F. Ramparany. Owl-based location ontology for context-aware services. *Proceedings of the Artificial Intelligence in Mobile Systems (AIMS 2004)*, pages 52–57, 2004. 28

[20] K. Gonçalves, H.K. Rubinsztejn, M. Endler, B. Silva, and S.D.J. Barbosa. Um aplicativo para comunicação baseada em localização. In *Proceedings of VI Workshop de Comunicação sem Fio e Computação Móvel*, pages 224–231, 2004. 40

[21] T.R. Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5:199–199, 1993. 12

[22] T. Gu and H.K. Pung. A middleware for building context-aware mobile services. In *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, volume 5, pages 2656–2660. IEEE, 2005. xi, 2, 8, 18, 20, 21, 28, 49

[23] T. Gu, X.H. Wang, H.K. Pung, and D.Q. Zhang. An ontology-based context model in intelligent environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference*, volume 2004, pages 270–275. Citeseer, 2004. 19, 49

[24] V. Haarslev, R. Möller, and M. Wessel. Racerpro user's guide and reference manual version 1.9. 0, 2007. 17

[25] P. Haase and L. Stojanovic. Consistent evolution of owl ontologies. *The Semantic Web: Research and Applications*, pages 182–197, 2005. 14

[26] J. Heflin. OWL Web Ontology Language-Use Cases and Requirements. *W3C Recommendation*, 10, 2004. 12

[27] P. Hitzler, M. Krotzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph. OWL 2 web ontology language primer. *W3C Recommendation*, 27, 2009. 15

[28] J. Honkola, H. Laine, R. Brown, and O. Tyrkko. Smart-m3 information sharing platform. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 1041–1046. IEEE, 2010. 20

[29] M. Horridge and S. Bechhofer. The owl api: a java api for working with owl 2 ontologies. In *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2009), CEUR Workshop Proceedings, Chantilly, VA, United States, October*, pages 23–24. Citeseer, 2009. 14, 28, 33

[30] I. Horrocks. DAML+ OIL: a description logic for the semantic web. *Bulletin of the Technical Committee on*, 51(4), 2002. 14

[31] I. Horrocks, P.F. Patel-Schneider, and F. Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003. 14

[32] W.L. Hürsch, E. Landau, S. Maclane, K. Lieberherr, E. Zachos, F. Haskel, I. Silvalepe, L. Keszenheimer, C. Videira, W. Hrsch, et al. Maintaining consistency and behavior of object-oriented systems during evolution. 1995. 29

[33] N. I. Qazi and M. A. Qadir. Algorithms to evaluate ontologies based on extended error taxonomy. In *Information and Emerging Technologies (ICIET), 2010 International Conference on*, pages 1–6. IEEE, 2010. 30

[34] Z. Jaroucheh, X. Liu, and S. Smith. CANDEL: Product Line Based Dynamic Context Management for Pervasive Applications. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 209–216. IEEE, 2010. xi, 2, 19, 20, 21, 49, 50

[35] A.M. Khattak, K. Latif, S. Lee, and Y.K. Lee. Ontology evolution: A survey and future challenges. *U-and E-Service, Science and Technology*, pages 68–75, 2009. 13

[36] H. Kim, Y.J. Cho, and S.R. Oh. CAMUS: A middleware supporting context-aware services for network-based robots. In *Advanced Robotics and its Social Impacts, 2005. IEEE Workshop on*, pages 237–242. IEEE, 2005. 8

[37] P. Korpipaa and J. Mantyjarvi. An ontology for mobile device sensor-based context awareness. *Modeling and Using Context*, pages 451–458, 2003. 2, 8

[38] J. Krumm. *Ubiquitous Computing Fundamentals*. Chapman & Hall/CRC, 2009. 6, 8, 9, 40

[39] L. Liu and M.T. Zsu. *Encyclopedia of database systems*. Springer Publishing Company, Incorporated, 2009. 12

[40] M. M. Saleemi, N. D. Rodríguez, J. Lilius, and I. Porres. A framework for context-aware applications for smart spaces. *Smart Spaces and Next Generation Wired/Wireless Networking*, pages 14–25, 2011. xi, 8, 20, 21, 28, 49

[41] N. Malik, U. Mahmud, and Y. Javed. Future challenges in context-aware computing. In *proceedings of the IADIS International Conference WWW/Internet*, pages 306–310, 2007. 9

[42] D.L. McGuinness, F. Van Harmelen, et al. OWL web ontology language overview. *W3C recommendation*, 10:2004–03, 2004. 15

[43] A. Michotte. *The perception of causality*. Basic Books, 1963. vi, vii, 3, 47

[44] G.K. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon. Context-aware computing: a guide for the pervasive computing community. In *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on*, pages 39–48. IEEE, 2004. 6

[45] B. Motik and R. Studer. Kaon2–a scalable reasoning tool for the semantic web. In *Proceedings of the 2nd European Semantic Web Conference (ESWC'05), Heraklion, Greece*, 2005. 17

[46] N.F. Noy, M. Crubézy, R.W. Fergerson, H. Knublauch, S.W. Tu, J. Vendetti, and M.A. Musen. Protégé-2000: An open-source ontology-development and knowledge-acquisition environment: Amia 2003 open source expo. In *AMIA Annual Symposium Proceedings*, volume 2003, page 953. American Medical Informatics Association, 2003. xi, 27, 28, 43, 45

[47] H. Park and J. Lee. A framework of context-awareness for ubiquitous computing middlewares. In *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, pages 369–374. IEEE, 2005. 8

[48] B. Parsia and E. Sirin. Pellet: An owl dl reasoner. In *Third International Semantic Web Conference-Poster*. Citeseer, 2004. 17, 34

[49] P. Patel, S. Jardosh, S. Chaudhary, and P. Ranjan. Context Aware Middleware Architecture for Wireless Sensor Network. In *2009 IEEE International Conference on Services Computing*, pages 532–535. IEEE, 2009. 2, 24

[50] S. Peters and H.E. Shrobe. Using semantic networks for knowledge representation in an intelligent environment. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 323–329. IEEE, 2003. 12

[51] S. Poslad. *Ubiquitous computing: smart devices, environments and interactions*. John Wiley & Sons Inc, 2009. 1

[52] W. Qin, Y. Shi, and Y. Suo. Ontology-based context-aware middleware for smart spaces. *Tsinghua Science & Technology*, 12(6):707–713, 2007. 1

[53] A. Ranganathan and R.H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 143–161. Springer-Verlag New York, Inc., 2003. 1, 2, 24

[54] R.C.A. Rocha, M.A. Casanova, and M. Endler. Promoting efficiency and separation of concerns through a hybrid model based on ontologies for context-aware computing. In *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*, pages 9–13. IEEE, 2007. xi, 2, 19, 21, 49

[55] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):65–67, 2002. 23, 24, 25

[56] V. Sacramento, M. Endler, H.K. Rubinsztejn, L.S. Lima, et al. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, page 2, 2004. 1, 19, 23, 24, 49

[57] D. Saha and A. Mukherjee. Pervasive computing: a paradigm for the 21st century. *Computer*, 36(3):25–31, 2003. 5

[58] R. Shearer, B. Motik, and I. Horrocks. Hermit: A highly-efficient owl reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*. Citeseer, 2006. 17, 34, 35

[59] C.F. Sørensen, M. Wu, T. Sivaharan, G.S. Blair, P. Okanda, A. Friday, and H. Duran-Limon. A context-aware middleware for applications in mobile Ad Hoc environments. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 107–110. ACM, 2004. 1, 23, 25

[60] V. Stanford, J. Garofolo, O. Galibert, M. Michel, and C. Laprun. The nist smart space and meeting room projects: signals, acquisition annotation, and metrics. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, volume 4, pages IV–736. IEEE, 2003. 1

[61] L. Stojanovic. *Methods and tools for ontology evolution*. PhD thesis, University of Karlsruhe, 2004. 2, 13, 30, 51

[62] T. Strang and C. Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*. Citeseer, 2004. 2, 7

[63] D. Tsarkov and I. Horrocks. Fact++ description logic reasoner: System description. *Automated Reasoning*, pages 292–297, 2006. 17

[64] Denny Vrandečić. Ontology evaluation. In Steffen Staab and Dr. Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 293–313. Springer Berlin Heidelberg, 2009. 10.1007/978-3-540-92673-3$_1$3.13, 14

[65] W3C. Composite capability/ preference profiles (cc/pp). Disponível em http://www.w3.org/TR/CCPP-struct-vocab/. Acessado em: janeiro de 2012, 2004. 7

[66] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *Personal Communications, IEEE*, 4(5):42–47, 2002. 5

[67] T. Weerasinghe and I. Warren. Odin: Context-Aware Middleware for Mobile Services. In *2010 6th World Congress on Services*, pages 661–666. IEEE, 2010. 1, 8, 23, 50

[68] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993. 5

[69] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1995. 5

[70] J. Ye, L. Coyle, S. Dobson, and P. Nixon. Ontology-based models in pervasive computing systems. *The Knowledge Engineering Review*, 22(4):315–347, 2007. 14

[71] T. Yonezawa, N. Okamoto, H. Yamazoe, S. Abe, F. Hattori, and N. Hagita. Privacy protected life-context-aware alert by simplified sound spectrogram from microphone sensor. In *Proceedings of the 5th ACM International Workshop on Context-Awareness for Self-Managing Systems*, CASEMANS '11, pages 4–9, New York, NY, USA, 2011. ACM. 10

[72] R. Zhao and J. Wang. Visualizing the research on pervasive and ubiquitous computing. *Scientometrics*, pages 1–20, 2011. 5

[73] X. Zhou, X. Tang, X. Yuan, and D. Chen. SPBCA: Semantic Pattern-Based Context-Aware Middleware. In *2009 15th International Conference on Parallel and Distributed Systems*, pages 891–895. IEEE, 2009. 2, 8