Um modelo para o gerenciamento de ontologias em ambientes ubíquos

Ana Helena O. R. Castillo, Fabrício N. Buzeto, Carla D. Castanho, Ricardo P. Jacobi

[1]Departamento de Ciência da Computação – Universidade de Brasília (UnB)
Caixa Postal 4466 – 70910-900 – Distrito Federal – DF – Brazil

anaozaki@unb.br,fabricio@intacto.com.br,{carlacastanho,rjacobi}@cic.unb.br

Abstract. *Besides providing interoperability, context aware middlewares must interact with the environment in order to collect relevant information for context determination. The aim of this work is use ontologies to model context informations, allowing, at the same time, that developers build applications capable of structurally and semantically modify the underlying context ontology by making use of an API (Application Programming Interface). Even more, it also includes the support for the task of consistency verification in the uOS middleware before each change, to guarantee that the inclusion of a new application will not interfere the others.*

Resumo. *Além de prover interoperabilidade, os middlewares sensíveis ao contexto devem interagir com o ambiente de modo a coletar informações relevantes para a determinação do contexto. A proposta desse trabalho é utilizar ontologias para modelagem de informações de contexto, permitindo, ao mesmo tempo, que desenvolvedores construam aplicações capazes de modificar estruturalmente e semanticamente essas ontologias de contexto através de uma API (Application Programming Interface). Além disso, a proposta inclui o suporte do middleware uOS na tarefa de verificar a consistência antes de cada mudança, para garantir que a inclusão de uma nova aplicação no middleware não afete as demais.*

1. Introdução

Computação ubíqua e computação pervasiva são termos muitas vezes utilizados de forma intercambiável [Saha and Mukherjee 2003]. A computação pervasiva tem como objetivo capturar o contexto do ambiente e construir modelos computacionais que alteram seu comportamento de acordo com 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 [Zhao and Wang 2011]. Outro conceito importante é o de sensibilidade ao contexto (*Context Awareness*), que é definido como a habilidade de um dispositivo móvel ser sensível ao que está em volta do usuário, ao estado do ambiente físico [Muthukrishnan et al. 2005]. Um sistema é sensível ao contexto se utiliza contexto para prover informações e serviços relevantes ao usuário [Dey 2001]. Nessa definição a "relevância" depende das tarefas dos usuários e é descoberta utilizando informações de contexto.

Um dos principais objetivos de um *middleware* em ambientes distribuídos é prover uma camada interoperável [1], capaz de encapsular a heterogeneidade dos dispositivos. Os *middlewares* sensíveis ao contexto devem, além de prover interoperabilidade, interagir com o ambiente de modo a coletar informações relevantes para a determinação do contexto. Por exemplo: localização dos usuários e dispositivos; estado dos dispositivos; preferências dos usuários e características dos dispositivos. Essas informações precisam ser processadas para identificação do contexto e sua evolução, propagando o contexto para o nível de serviço.

Este trabalho aborda a utilização de ontologias para modelagem e gerenciamento das informações de contexto em ambientes ubíquos. Devido à diversidade de domínios nos quais as aplicações podem estar inseridas, propõe-se que ontologias de contexto não sejam modeladas de forma estática. Assim faz-se necessário o desenvolvimento de mecanismos que suportem a evolução de ontologias de acordo com a dinamicidade do ambiente ubíquo. Para atingir esses objetivos, este trabalho estrutura uma solução baseada em três componentes básicos: (i) um gerenciador de informações de contexto; (ii) um gerenciador de alterações das ontologias e (iii) a definição de uma interface para permitir que aplicações criem e alterem ontologias de contexto. O modelo proposto agrega sensibilidade ao contexto ao *uOS* [Buzeto et al. 2010], *middleware* voltado para a adaptabilidade de serviços em ambientes inteligentes.

O restante deste trabalho é organizado da seguinte forma. A Seção 2 introduz conceitos básicos relativos a ontologias. Trabalhos correlatos são apresentados na Seção 3. Na Seção 4 descreve-se brevemente o *middleware uOS*. O modelo proposto é discutido na Seção 5 e algumas considerações finais são apresentadas na Seção 6.

2. 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 [Liu and Zsu 2009]. Este trabalho adota a seguinte definição de ontologia:

Definição: *Uma **ontologia** é uma especificação formal e explícita de uma conceitualização compartilhada acerca um domínio de interesse* [Gruber et al. 1993].

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;

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

- 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 [Heflin 2004], 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 [Peters and Shrobe 2003]. Ao descrever o conhecimento acerca de um domínio em uma ontologia, obtém-se uma representação semântica das informações, as quais podem usadas para direcionar o comportamento de aplicações sensíveis ao contexto.

2.1. 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* [Stojanovic 2004].

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 mudança [Stojanovic 2004]. Quando a verificação é feita **depois**, todas as mudanças podem ser checadas de uma só vez. Porém, se a checagem falhar 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 checagem 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.

2.2. Consistência

No âmbito da Lógica, diz-se que uma teoria é consistente quando não contém contradições [Tarski and Helmer-Hirschberg 1941]. Este trabalho adota a definição dada por [Haase and Stojanovic 2005]:

Definição: *Uma **ontologia** O é consistente com respeito a um conjunto de condições de consistência K se, e somente se, para todo $k \in K$, O satisfaz a condição de consistência $k(O)$.*

Desse modo, entende-se que uma ontologia é consistente quando todas as restrições de consistência são respeitadas.

Exemplo: Suponha que o Departamento de Ciência da Computação da Universidade X ofereça cursos de graduação e mestrado. Considere que a ontologia utilizada para representar as entidades contenha a estrutura de conceitos apresentada na Figura 1.

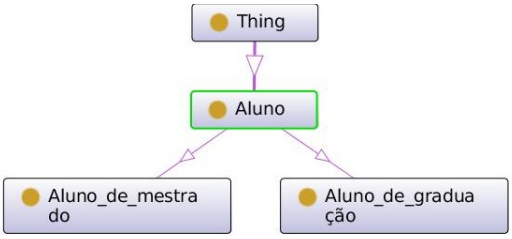

Figura 1. Exemplo de uma ontologia em OWL (*Web Ontology Language*). (Figura gerada pela ferramenta Protégé.)

Alguns anos após a utilização dessa estrutura, o referido Departamento adquire capacidade para oferecer um curso de doutorado. Para atender a essa demanda, um dos conceitos que poderiam ser refletidos para o sistema seria o de aluno de doutorado. Porém, o curso de doutorado participa do mesmo programa do mestrado, surgindo a figura do aluno de pós-graduação, um conceito que engloba os alunos de mestrado e doutorado. Dessa forma, a nova estrutura é ilustrada na Figura 2.

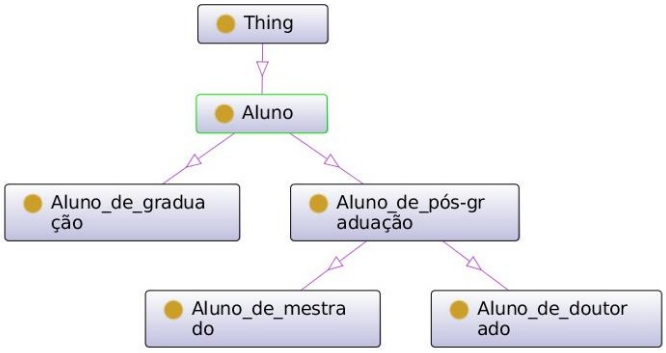

Figura 2. Exemplo de uma ontologia em OWL. (Figura gerada pela ferramenta Protégé.)

Observe que a inclusão do conceito de aluno de doutorado não necessariamente faz com que apenas este novo conceito seja incorporado na ontologia. Além disso, caso o conceito de aluno de doutorado fosse inserido na raiz *Aluno*, teríamos uma inconsistência pois *Aluno de pós-graduação* é um conceito que engloba o de aluno de doutorado. Pode-se dizer que de acordo com as necessidades do usuário, novos conceitos são vislumbrados e o conhecimento acerca de um domínio é aprimorado. Para que a ontologia não fique estagnada é preciso criar um mecanismo que permita a evolução da ontologia.

Deste exemplo, percebe-se que a evolução de uma ontologia não é um processo trivial, uma vez que a simples inclusão de um conceito pode causar inconsistências. Mesmo que a alteração de uma ontologia seja feita por um especialista, não há como garantir que todos os efeitos colaterais serão analisados. Faz-se necessário que a detecção de inconsistências seja um processo automatizado incorporado ao modelo de representação de contexto do *middleware*.

3. Trabalhos Relacionados

O uso de ontologias em ambientes ubíquos tem como motivação a possibilidade de modelar a semântica do contexto de forma independente da linguagem de programação, sistema operacional e/ou *middleware* utilizado.

O projeto CoBrA (*Context Broker Architecture*) propõe uma ontologia para ambientes pervasivos sensíveis ao contexto [Chen et al. 2003]. As principais contribuições estão relacionadas aos casos de uso modelados, juntamente com a separação de conceitos chave como localização, lugares, atividade e agentes. Porém, não há divisão entre os conceitos relacionados a domínio, como *Campus*, e conceitos mais genéricos e aplicáveis a vários domínios, como *Place*, pois ambas as classes são listadas em um mesmo bloco. Além disso, as classes e propriedades da ontologia são definidas de forma estática, não sendo tratado o aspecto evolutivo das ontologias a medida que novos conceitos são incorporados nos domínios.

Já o projeto SOCAM (*Service-Oriented Context Aware Middleware*) apresenta um modelo de contexto de duas camadas baseado em ontologia [Gu and Pung 2005]. Nesse modelo, a camada superior possui conceitos 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 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.

O DSM (*Dynamic Service Management*) [Min and Min 2010] apresenta um modelo de contexto baseado no projeto SOCAM. A diferença é que no DSM há uma terceira camada de ontologia, a qual é dedicada para os serviços. Esta camada gerencia a inclusão e exclusão de serviços. Quando um serviço é inferido da ontologia de domínio, ocorre uma busca pelo serviço inferido e, por fim, o serviço é inserido na ontologia de serviço. Caso haja alteração no domínio do contexto, a informação inferida anteriormente pode não ser mais necessária, nesse caso ocorre uma remoção na ontologia de serviço.

O projeto CANDEL (*Context As dyNamic proDuct Line*) apresenta uma forma de modelagem de contexto independente de domínio baseada em modelos de características [Jaroucheh et al. 2010]. O gerenciamento do contexto é realizado por componentes denominados CPCs (*Context Proxy Components*), que atuam como *proxies* das informações de contexto. Como podem haver várias características acerca de uma mesma informação de contexto, cabe aos serviços selecionar as características a serem utilizadas para prover as informações de contexto adequadas às aplicações. Dessa forma, o modelo trata os diferentes níveis de abstração das informações de contexto. Assim como em [Liu et al. 2010], a sublinguagem OWL DL (*Web Ontology Language Description Logic*) foi utilizada na implementação.

A partir da análise de 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. Dentre os trabalhos analisados, entretanto, não foi encontrado suporte à evolução de ontologias, característica que, conforme discutido na seção 2, contribui para aumentar a flexibilidade

do *middleware* no suporte a aplicações sensíveis ao contexto.

4. *Middleware uOS*

O *middleware uOS* foi concebido com o objetivo de promover a adaptabilidade de serviços em um ambiente de computação ubíqua [Buzeto et al. 2010]. 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 (*Device Service Oriented Architecture*) quem assume esses papéis são os dispositivos e não os *Web Services*. Assim como em outros projetos, o gerenciamento de eventos segue o modelo *Publish-Subscribe*. Basicamente, o *middleware uOS* é composto por três camadas:

1. Camada de Rede: responsável por gerenciar as interfaces de rede do dispositivo;
2. Camada de Conectividade: responsável por coordenar a comunicação entre dispositivos;
3. Camada de Adaptabilidade: responsável por gerenciar as aplicações, 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*, um dispositivo capaz de intermediar a comunicação. Por fim, a camada de Adaptabilidade contém um módulo chamado *Adaptability Engine*, que faz a intermediação entre as aplicações 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 por meio das operações de *Deploy* e *Undeploy*.

5. Modelo para gerenciamento de ontologias em ambientes ubíquos

A Figura 3 ilustra os componentes responsáveis por gerenciar as informações de contexto e as mudanças que ocorrem na ontologia devido ao *Deploy* e *Undeploy* de aplicações no *middleware uOS*. As informações chegam ao gerenciador de contexto, o qual gera instâncias de conceitos preexistentes e inclui essas informações na ontologia. O gerenciador de contexto também é responsável por entregar informações de contexto atualizadas às aplicações que estão registradas para recebê-las. O mecanismo de registro segue o modelo definido pela arquitetura DSOA [Buzeto et al. 2010]. Utilizando a *uOS-Context API* (*Application Programming Interface*) (Seção 5.2), as aplicações podem alterar a ontologia e evoluir as informações de contexto existentes no *middleware*. Contudo, estas alterações são efetuadas somente após passarem por uma verificação de consistência, realizada pelo gerenciador de alterações.

As subseções a seguir descrevem os três componentes propostos para o gerenciamento de uma ontologia de contexto do *middleware uOS*.

5.1. Gerenciador de Contexto

O Gerenciador de Contexto é o componente responsável por controlar o fluxo das informações de contexto. As informações são essencialmente obtidas por meio de sensores e representadas no *middleware* por uma ontologia. A linguagem utilizada para a

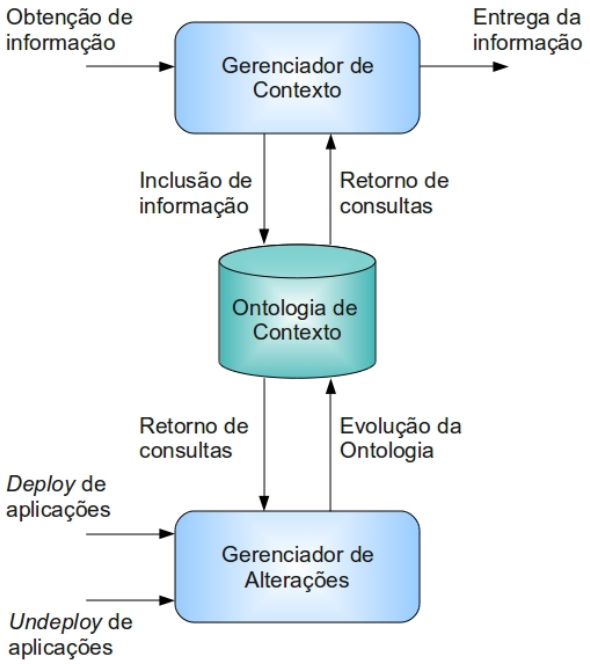

Figura 3. Gerenciamento do contexto do *middleware uOS* baseado em ontologia.

ontologia é a OWL (*Web Ontology Language*), por possibilitar um alto grau de expressividade semântica e ser recomendada pela W3C (*World Wide Web Consortium*). O nome da raiz de uma ontologia em OWL é *Thing*.

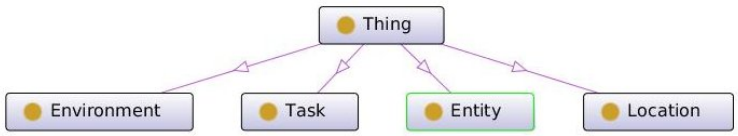

Figura 4. Ontologia de alto nível de abstração das informações de contexto. (Figura gerada pela ferramenta Protégé.)

Conforme ilustrado na Figura 4, o gerenciador de contexto do uOS contém uma ontologia de alto nível, a qual possui os conceitos considerados necessários para descrever o contexto de forma independente do domínio. A adoção dessa ontologia serve para padronizar a forma como é tratado cada tipo de conceito relacionado ao contexto. Existem também as ontologias de domínio, as quais extendem a ontologia de alto nível e são construídas e evoluídas a partir das aplicações por meio da *uOS-Context API*, detalhada a seguir.

5.2. *uOS-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, faz-se necessário que a inserção de aplicações no *middleware* permita que tais conceitos sejam refletidos na ontologia. A utilização de uma API tem como vantagem a separação das alterações realizadas em relação à forma de persistência da ontologia

no *middleware*. Além disso permite que tais alterações ocorram de forma padronizada e controlada pelo gerenciador de alterações (Seção 5.3), o qual garante a consistência da ontologia após a mudança.

O formato de cada mudança segue o modelo *sujeito-predicado-objeto*. O *sujeito* refere-se ao tipo de elemento em que será realizada a mudança: classe, subclasse, propriedade, subpropriedade, *range*, domínio... O *predicado* corresponde ao tipo de mudança: adição, remoção, edição... E o *objeto* contém a definição do sujeito, por exemplo, a classe que está sendo inserida. O número de parâmetros do objeto varia de acordo com o sujeito. Para incluir uma propriedade, o domínio e o *range* relacionados a propriedade também podem ser passados por parâmetro.

A *uOS-Context API* consiste em um conjunto de interfaces de programação que tem como objetivo viabilizar a manipulação estrutural e semântica da ontologia de contexto contida no *uOS*. A API oferece aos desenvolvedores a possibilidade de que suas aplicações alterem a ontologia existente no *middleware*. Cada instância do *uOS* possui uma ontologia ditada pelos conceitos inerentes as aplicações utilizadas pelo usuário. Desse modo, a construção da ontologia torna-se dinâmica e capaz de evoluir de acordo com as necessidades do ambiente.

5.3. Gerenciador de Alterações

Uma das razões para a utilização de ontologias em ambientes ubíquos é permitir que as informações de contexto sejam compartilhadas não somente entre vários dispositivos mas também entre várias aplicações. Entretanto, o desenvolvimento de aplicações frequentemente ocorre de forma independente, tornando mais complexa a tarefa de compartilhamento das informações. Além disso, o fato das aplicações alterarem a ontologia de contexto existente no *middleware* possibilita o surgimento de inconsistências. O gerenciador de alterações tem como objetivo controlar as mudanças realizadas na ontologia. A verificação da consistência é feita antes das alterações serem efetuadas, de modo a não permitir que a ontologia fique em um estado inconsistente.

O modelo de consistência adotado utiliza regras semelhantes às definidas em [Stojanovic 2004], com algumas alterações relacionadas a especificidades da linguagem OWL. A seguir estão listadas regras estruturais para manutenção da consistência:

1. Todas as entidades (classes, propriedades e instâncias) possuem um identificador único.
2. A hierarquia de classes é um grafo acíclico direcionado. O mesmo vale para a hierarquia de propriedades.
3. Existe uma classe que é superclasse de todas as classes, que na linguagem OWL é a classe *Thing*.
4. Existe uma classe que é subclasse de todas as classes, que na linguagem OWL é a classe *Nothing*.
5. Todas as classes existentes na ontologia estão presentes na hierarquia de classes. O mesmo vale para as propriedades.
6. Uma propriedade pode estabelecer uma relação entre indivíduos ou relacionar um indivíduo a um valor.
7. Um domínio relaciona o sujeito de uma propriedade a uma classe.
8. Um *range* relaciona o objeto de uma propriedade a uma classe.

9. Todo indivíduo (instância) está associado a pelo menos uma classe.
10. Um indivíduo deve apenas ter propriedades da(s) classe(s) associada(s) ou de uma de suas superclasses.
11. A cardinalidade mínima deve ser menor que a cardinalidade máxima.
12. A cardinalidade é definida para uma propriedade de uma classe.

Além das regras estruturais, o gerenciador de alterações inclui regras de compatibilidade relacionadas à dependência entre as aplicações:

1. Caso uma ou mais aplicações sejam excluídas, a exclusão de uma entidade só pode ser realizada se não estiver relacionada a nenhuma outra aplicação incluída no *middleware*.
2. Caso uma aplicação tente inserir uma nova entidade no *middleware*, a aplicação só será aceita se a inclusão não infringir nenhuma das regras estruturais citadas acima.
3. Caso uma aplicação tente inserir uma entidade já existente no *middleware*, a aplicação só será aceita se as entidades forem idênticas.

6. Conclusões

Neste trabalho foram propostos os componentes responsáveis pelo gerenciamento de uma ontologia de informações de contexto em um ambiente ubíquo. Conforme visto nos trabalhos correlatos, algumas iniciativas de utilização de ontologias para ambientes sensíveis ao contexto foram realizadas. Contudo, a maneira como os *middlewares* para ambientes ubíquos gerenciam essas ontologias ainda se apresenta como potencial foco de investigação.

Mais precisamente, a proposta deste trabalho é permitir que os desenvolvedores construam aplicações capazes de modificar a ontologia utilizando uma API. 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. A verificação de consistência deve ser feita pelo *middleware*, de modo a evitar que a inclusão de alterações na ontologia afete outras aplicações. Portanto, faz-se necessário incorporar funções de gerenciamento no *middleware*, o qual será encarregado de lidar com o gerenciamento da ontologia e das informações de contexto. Trabalhos futuros incluem análise de novas regras de consistência que permitirão maior controle do *middleware* sobre mudanças realizadas na ontologia.

Agradecimentos Os autores agradecem o apoio do DPP (Decanato de Pesquisa e Pós-Graduação) da Universidade de Brasília para a realização deste projeto.

Referências

Buzeto, F., de Paula Filho, C., Castanho, C., and Jacobi, R. (2010). DSOA: A Service Oriented Architecture for Ubiquitous Applications. In Bellavista, P., Chang, R.-S., Chao, H.-C., Lin, S.-F., and Sloot, P., editors, *Advances in Grid and Pervasive Computing*, volume 6104 of *Lecture Notes in Computer Science*, pages 183–192. Springer Berlin / Heidelberg.

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

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing*, 5:4–7.

Gruber, T. et al. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5:199–199.

Gu, T. and Pung, H. (2005). 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.

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

Heflin, J. (2004). OWL Web Ontology Language-Use Cases and Requirements. *W3C Recommendation*, 10.

Jaroucheh, Z., Liu, X., and Smith, S. (2010). 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.

Liu, C., Chang, K., Jason, J., and Hung, S. (2010). Ontology-Based Context Representation and Reasoning Using OWL and SWRL. In *8th Annual Communication Networks and Services Research Conference*, pages 215–220. IEEE.

Liu, L. and Zsu, M. (2009). *Encyclopedia of database systems*. Springer Publishing Company, Incorporated.

Min, D. and Min, H. (2010). Dynamic service management technique for various context information. In *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, volume 2, page V2. IEEE.

Muthukrishnan, K., Lijding, M., and Havinga, P. (2005). Towards smart surroundings: Enabling techniques and technologies for localization. *Location-and Context-Awareness*, pages 350–362.

Peters, S. and Shrobe, H. (2003). 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.

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

Stojanovic, L. (2004). *Methods and tools for ontology evolution*. PhD thesis, University of Karlsruhe.

Tarski, A. and Helmer-Hirschberg, O. (1941). *Introduction to Logic and to the Methodology of Deductive Sciences*. Oxford university press.

Zhao, R. and Wang, J. (2011). Visualizing the research on pervasive and ubiquitous computing. *Scientometrics*, pages 1–20.