U-MAS: Um Meta-modelo para o Desenvolvimento de Aplicações Multiagentes Ubíquas

Mauricio da Silva Escobar, Marcelo Blois Ribeiro

Faculdade de Informática – PUCRS – Porto Alegre – RS – Brasil

{mauricio.escobar,marcelo.blois}@pucrs.br

Abstract. *This paper presents a meta-model independent from methodology to define ubiquitous concepts at the same abstraction level of multiagent systems concepts. It is based on FAML, a generic meta-model for MAS development, and serves for modelling MAS for ubiquitous environments. Using a standard model-driven development approach, the meta-model can be used as the basis for code generation to MAS platforms in order to implement ubiquitous multi-agent environments.*

Resumo. *Este artigo apresenta um meta-modelo independente de metodologia, para a definição de conceitos de computação ubíqua no mesmo nível de abstração dos conceitos de sistemas multiagentes. Ele é baseado no FAML, um meta-modelo genérico para o desenvolvimento de sistemas multiagentes (SMAs), e serve para a modelagem de SMAs para ambientes ubíquos. Utilizando uma abordagem de desenvolvimento orientada a modelos (model-driven development - MDD), o meta-modelo pode ser utilizado como base para a geração de código para plataformas de SMAs a fim de implementar ambientes multiagentes ubíquos.*

1. Introdução

A computação ubíqua é uma visão de futuro onde sistemas computacionais estarão integrados em nosso dia-a-dia, provendo serviços e informações em qualquer lugar e a qualquer momento [Weiser 1999, Ark & Selker 1999]. Ela propõe um novo paradigma computacional capaz de prover acesso computacional a usuários de forma invisível, isto é, o usuário não precisa perceber a tecnologia para aproveitar seus benefícios.

A tecnologia de agentes e sistemas multiagentes (SMAs) desempenham um papel importante em computação ubíqua, e, tem sido utilizadas como uma abordagem atrativa para a criação de aplicações para ambientes ubíquos [Symonds 2010, Gunasekera *et al*. 2008]. Agentes, inseridos em ambientes ubíquos, serão capazes de responder às mudanças ocorridas no ambiente, seja ele virtual ou físico, melhorando seu desempenho e seu comportamento, adicionando inteligência a esse ambiente.

A computação ubíqua tem evoluído da área de pesquisa acadêmica para a realidade comercial através do desenvolvimento e disponibilidade de infra-estruturas, ferramentas e ambientes que suportam o desenvolvimento de sistemas ubíquos. Muitas pesquisas em computação apontam características que devem existir em qualquer sistema ubíquo, tais como: contexto, localização, sensores e entidades capazes de sentir e responder às alterações no ambiente. Um dos problemas atuais é que existem muitas abordagens para o desenvolvimento de sistemas ubíquos que utilizam ou propõem soluções *ad-hoc*. Este tipo de desenvolvimento força os desenvolvedores a trabalharem com abstrações de baixo-nível e, muitas vezes, criando e implementando sistemas sem terem uma visão clara do contexto com o qual o sistema deve interagir e quais entidades fazem parte do ambiente.

Em engenharia de software, a abordagem *Model-Driven Development* (MDD) permite a especificação de sistemas utilizando modelos de abstrações que são independentes de metodologia e tecnologia, e que podem ser transformados em código-fonte, através de geradores automatizados. Os avanços em MDD têm motivado a aplicação de técnicas de modelagem para diversos domínios, motivando os pesquisadores a formalizarem o conhecimento de um domínio em particular sob a forma de meta-modelos. Visando reduzir o esforço e o retrabalho no desenvolvimento de aplicações ubíquas, são necessários novos modelos que ofereçam aos desenvolvedores conceitos em alto-nível de abstração.

Este artigo apresenta um meta-modelo independente de metodologia, que integra conceitos de computação ubíqua aos conceitos de SMA. O principal objetivo é prover um meta-modelo para a modelagem de conceitos de ubiquidade para qualquer SMA, independentemente do processo ou metodologia utilizados para criá-lo. Este meta-modelo é baseado no FAML [Beydoun 2009], um meta-modelo genérico para o desenvolvimento de SMAs. O restante deste artigo é organizado como a seguir: a seção 2 apresenta o referencial teórico sobre agentes de software, computação ubíqua e o meta-modelo FAML. A seção 3 apresenta o meta-modelo proposto, ilustrando-o em relação aos aspectos de *runtime* e *design-time*. A seção 4 apresenta os trabalhos relacionados. Por fim, são apresentadas a conclusão e trabalhos futuros na seção 5.

2. Referencial teórico

2.1. Sistemas Multiagentes e Computação Ubíqua

Segundo Weiss (1999), um agente pode ser definido como um sistema de computador situado em algum ambiente e capaz de agir de forma autônoma para atingir um objetivo, onde a autonomia refere-se à capacidade de agir de acordo com sua própria linha de controle. Os sistemas multiagentes são apropriados para domínios que são naturalmente distribuídos. O uso dos conceitos de agentes para a engenharia de sistemas distribuídos provê diversas vantagens na redução de complexidade [Jennings 2001, Silva *et al*. 2003], como a autonomia e interações em alto nível. A autonomia de um agente possibilita a interação com ambos ambientes físicos ou virtuais, tornando o paradigma de agentes apropriado para ser utilizado em computação ubíqua.

A computação ubíqua é uma visão de futuro onde sistemas computacionais estarão integrados em nosso dia-a-dia, provendo serviços e informações em qualquer lugar e a qualquer momento [Weiser 1999, Ark & Selker 1999]. A computação ubíqua também pode ser caracterizada por sistemas e tecnologias que são [Symonds 2010]: (i) **embarcadas,** diversos dispositivos integrados ao ambiente comunicando-se através de uma rede; (ii) **conscientes de contexto,** estes dispositivos podem reconhecer o usuário e o contexto do ambiente; (iii) **personalizadas,** podem ser customizadas para as necessidades de cada usuário; (iv) **adaptativas,** o sistema pode alterar o seu comportamento em resposta ao usuário; e, (v) **antecipatórias,** pode antecipar o desejo do usuário sem a necessidade de mediação. A fim de melhorar o entendimento sobre o processo de construção do meta-modelo proposto neste trabalho, a próxima seção apresenta o meta-modelo FAML.

2.2. FAML – Um Meta-modelo Genérico para o Desenvolvimento de SMAs

O FAML [Beydoun 2009] é um meta-modelo que unifica conceitos e relacionamentos presentes em meta-modelos e metodologias existentes para o desenvolvimento de sistemas multiagentes. O FAML foi construído com o compromisso de ser genérico,

considerando conceitos que são comuns a qualquer SMA. Dessa forma, conceitos específicos de domínio, como atuadores (comuns na robótica) ou relacionados a um único agente, foram omitidos. De acordo com seus autores, os conceitos do FAML podem ser especializados a fim de incluir conceitos específicos de domínio.

O meta-modelo FAML foi criado a partir de um processo interativo que consiste dos seguintes passos: (1) determinação dos conceitos gerais de SMAs. Como mencionado anteriormente, conceitos específicos de domínio foram omitidos; (2) listagem de definições (candidatas) para os conceitos determinados no passo um; (3) reconciliação entre as definições de cada conceito para manter a consistência interna do meta-modelo; (4) separação dos conceitos em dois conjuntos: *design-time* e *runtime*; e, (5) identificação dos relacionamentos entre conceitos dos conjuntos *design* e *runtime*. A saída para este processo consiste em quatro categorias de conceitos. Cada categoria refere-se a um escopo no meta-modelo, que são: nível de sistema (*design-time* e *agent-external*), nível de definição do agente (*design-time* e *agent-internal*), nível de ambiente (*runtime*, *agent-externals*), e, nível do agente (*runtime*, *agent-internal*). Nessa seção apresentaremos brevemente os principais conceitos do FAML que são necessários para o entendimento do meta-modelo proposto, apresentado na seção 3.

O conceito central em tempo de projeto é o SYSTEM. Ele representa o produto final de um projeto de desenvolvimento orientado a agentes. Um SYSTEM possui um AGENTDEFINITION, que por sua vez é a especificação do estado inicial de um agente logo após sua criação. Um SYSTEM também possui zero ou mais papéis. Um papel (ROLE) determina um padrão comportamental esperado por agentes em um sistema. O papel também define quais facetas um agente interage, e quais agentes podem alterar ou sentir essas facetas. Uma faceta (FACET) é uma propriedade do ambiente com o qual os agentes podem interagir. Um sistema possui zero ou mais FACET, definidas através da classe FACETDEFINITION. Um FACETDEFINITION especifica a estrutura de uma faceta, incluindo seu nome, tipo de dado e modo de acesso.

Os agentes que existirão em tempo de execução são descritos através da classe AGENTDEFINITION. Ela serve para inicializar todos os agentes relacionados ao sistema. Ela possui também uma função específica de papel, servindo para inicializar um agente quando ele desempenha um papel em execução. Um AGENTDEFINITION consiste de um estado inicial e de um número de planos definidos através da classe PLANSPECIFICATION. Cada plano é composto por um conjunto de ações (ACTIONSPECIFICATION), que podem ações que especificam como alterar uma faceta (FACETACTIONSPECIFICATION), ou ações que especificam como enviar uma mensagem em um determinado esquema (MESSAGEACTIONSPECIFICATION). Além da definição de planos e ações, também é necessária a especificação dos recursos que um agente utiliza. Um RESOURCESPECIFICATION especifica algo que possui um nome, uma representação, e que pode ser adquirido, compartilhado ou produzido. A próxima seção apresenta um cenário a fim de ilustrar algumas lacunas identificadas na utilização do FAML para a modelagem de uma aplicação ubíqua.

2.2.1. Um Exemplo de SMA Ubíquo

Peter está chegando em seu apartamento e seu agente pessoal (Bill), executando em seu celular, percebe que Peter aproxima-se da entrada. Bill possui todas as preferências de seu usuário armazenadas. A fim de satisfazer seu usuário, Bill decide configurar a iluminação do apartamento e ligar o rádio. Para fazer isso, Bill migra do celular para o computador pessoal de Peter, que possui todos os recursos necessários para Bill realizar as configurações no ambiente. Nesse cenário, a mobilidade do agente é uma capacidade

fundamental. Essa capacidade não é coberta pelo FAML. Outro fato importante é a definição de múltiplos ambientes. No FAML, um agente pertence a um único ambiente. Como podemos ver no cenário, o agente atua ao menos em dois ambientes. Além de definir os ambientes onde os agentes atuarão, é necessário definir todos os dispositivos disponíveis no ambiente.

No cenário, existem quatro dispositivos principais que compreendem o computador pessoal, o aparelho de som, o sistema de iluminação e o celular de Peter. Cada dispositivo possui interfaces (atuadores) permitindo que entidades como agentes possam realizar comandos, alterando o estado destes dispositivos (como por exemplo, ligar ou desligar uma lâmpada). O FAML não possui um conceito que possa representar adequadamente um dispositivo em função de seus atributos. Além disso, o FAML não apresenta um elemento para a definição de atuadores. Atuadores são elementos utilizados para alterar o estado do ambiente ou de algum outro elemento, porém, eles não possuem estado. O conceito no FAML que mais se aproxima para o mapeamento de atuadores é o FACETDEFINITION, no entanto, segundo os autores, uma faceta pode ser alterada, isto é, pode variar o estado independente do agente. Por isso, não seria adequado definirmos um atuador como uma especialização de faceta.

Os agentes devem ser conscientes de suas localizações em um sistema ubíquo, a fim de agirem adequadamente no ambiente. A localização pode delimitar o conjunto de ações que um agente pode realizar. Considerando o cenário, quando o agente percebe que está no apartamento de seu dono, ele pode automaticamente executar ações para cumprir os objetivos de seu usuário. O FAML também não possui um elemento capaz de mapear a localização de um agente. A classe FACETDEFINITION poderia ser um conceito candidato para este mapeamento, entretanto, uma faceta pode ser alterada explicitamente pelo agente. Uma localização, entretanto, somente pode ser sentida pelo agente, pois ela varia independentemente de suas ações.

3. U-MAS: Um Meta-modelo para o Desenvolvimento de Aplicações Multiagentes Ubíquas

Na literatura são encontradas diversas arquiteturas, frameworks e linguagens que visam apoiar o desenvolvimento de aplicações ubíquas. Muitas dessas abordagens compartilham conceitos em comum, porém, em alguns casos, esses conceitos são contraditórios ou existem divergências em seus significados. Além disso, algumas dessas abordagens propõem soluções *ad-hoc,* dificultando a reutilização.

O meta-modelo proposto foi primeiramente construído através da determinação do conjunto inicial de conceitos a serem utilizados. Esses conceitos são frequentemente encontrados em aplicações ubíquas, e suas definições foram baseadas na literatura existente. O processo de criação do meta-modelo foi baseado nos mesmos passos utilizados na construção do meta-modelo FAML [Beydoun 2009]. O meta-modelo proposto visa prover um conjunto de conceitos genéricos, capazes de suportar o desenvolvimento de aplicações multiagentes ubíquas. A idéia é tornar os conceitos de computação ubíqua inerentes a qualquer agente executando em um ambiente ubíquo. Através da extensão do FAML, nossa intenção é reutilizar um meta-modelo existente para o desenvolvimento de SMAs estabelecendo conceitos de computação ubíqua no mesmo nível de abstração dos conceitos de agentes. Para tornar essa extensão possível foram necessários alguns ajustes no FAML, apresentados na próxima seção.

3.1. Conceitos em Tempo de Projeto

Em tempo de projeto (Figura 1), o conceito SYSTEM representa o produto final de um projeto de desenvolvimento orientado a agentes. Um sistema ubíquo possui um ou mais espaços onde os agentes atuam, definidos através da classe SPACEDEFINITION. Espaços [Campiolo *et al*. 2007, Jansen *et al*. 2005] agrupam elementos tais como dispositivos, agentes e recursos no ambiente. Espaços são definidos como a composição de diversas localizações. Uma localização (LOCATION) pode ser definida como uma posição ou área de interesse [Campiolo *et al*. 2007, Barron & Cahill 2006]. Através da localização, o sistema pode gerenciar recursos, agentes, e demais entidades presentes nos ambientes. Uma localização possui uma posição (POSITION), e um sistema de posicionamento (POSITIONINGSYSTEM), que define a forma como a posição deve ser interpretada. Uma posição consiste de um valor inicial, e um sistema de posicionamento consiste de um nome que o identifica.

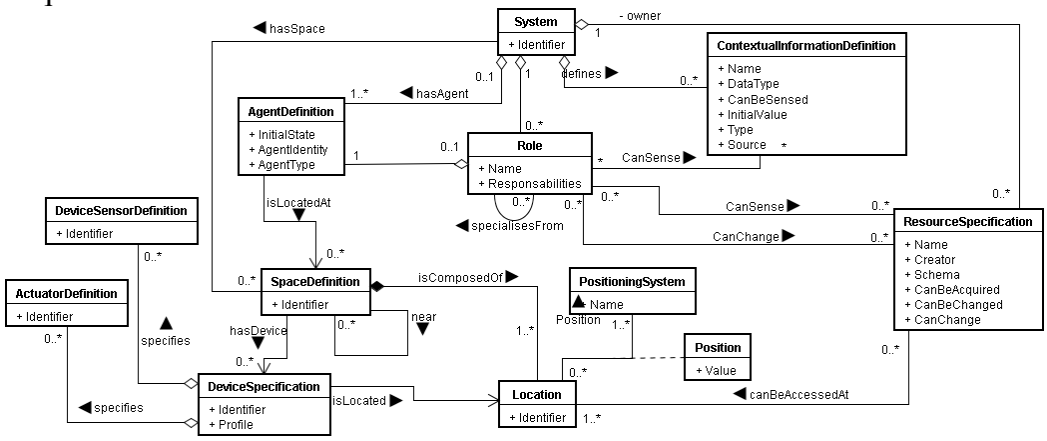

Figura 1: Classes externas ao agente em tempo de projeto (adaptado de [Beydoun 2009])

Um ambiente ubíquo também possui uma série de dispositivos, que podem ser de diversos tipos, tais como dispositivos embarcados no ambiente físico [Hansmann *et al*. 2001], e dispositivos móveis, tais como celulares e PDAs. Dispositivos são definidos através da classe DEVICESPECIFICATION. Cada dispositivo possui um identificador e um perfil com suas características e capacidades que podem ser definidas em função de seus atributos [Ilyas & Mahgoub 2004]. Os atributos podem ser estáticos (invariantes no tempo) ou dinâmicos (variantes no tempo). Por exemplo, a resolução máxima de uma tela é um atributo estático, enquanto o nível atual de bateria é um atributo dinâmico. Dispositivos também possuem uma localização dentro de um ambiente, permitindo assim que possam ser identificados. Além dos atributos, dispositivos também podem possuir sensores e atuadores. Sensores [Jansen *et al*. 2005, Seyler *et al*. 2007] são componentes de software que provêem operações para a coleta de informações no ambiente. Sensores não podem alterar o estado do ambiente, eles somente o observam. A fim de diferenciar sensores de dispositivos dos sensores de um agente, no meta-modelo, um sensor de dispositivo é definido através da classe DEVICESENSOR-DEFINITION. Dessa forma, um dispositivo especifica zero ou mais DEVICESENSOR-DEFINITION. Assim como sensores, dispositivos podem especificar zero ou mais atuadores. Atuadores [Jansen *et al*. 2005] são elementos capazes de influenciar o estado do ambiente. Por exemplo, um atuador de temperatura pode aumentar a temperatura de uma sala. Um atuador é definido através da classe ACTUATORDEFINITION.

Todos os agentes em um sistema ubíquo, possuem uma localização dentro de um espaço. Para permitir o relacionamento entre um agente e um espaço, um novo

relacionamento foi adicionado à classe AGENTDEFINITION, relacionando-a à zero ou mais espaços. Um sistema também possui recursos, que podem ser vistos como objetos pertencentes ao sistema [Kurkovsky 2007], tais como impressoras, arquivos, bancos de dados, etc. Um recurso é algo que possui um nome, uma representação, e, pode ser adquirido, compartilhado ou produzido. A fim de adequar a definição de recurso, dois novos relacionamentos e novas propriedades foram adicionadas à classe RESOURCE-SPECIFICATION. No meta-modelo proposto, um sistema é dono de zero ou mais recursos, e, um papel (ROLE) define se um agente poderá sentir ou alterar um recurso. Um recurso consiste de um nome, um criador, um esquema para a sua representação, e duas propriedades indicando se ele pode ser adquirido ou alterado.

Outra característica importante em aplicações ubíquas, é a noção de contexto. Contexto [Saha & Mukherjee 2003] pode ser definido como qualquer informação que pode ser utilizada para caracterizar a situação de uma entidade. No FAML, as características de uma faceta são similares as de uma informação contextual. A fim de evitar interpretações errôneas, no meta-modelo proposto o conceito FACETDEFINITION foi substituído pelo conceito CONTEXTUALINFORMATIONDEFINITION. Os seguintes atributos de FACETDEFINITION em CONTEXTUALINFORMATION-DEFINITION foram mantidos: NAME, DATATYPE, INITIALVALUE e CANBESENSED. Também foram preservados os relacionamentos DEFINES e CANSENSE, que agora relacionam respectivamente um sistema a suas informações contextuais, e um papel e as informações contextuais que ele pode sentir. Os atributos CANBECHANGED e CANCHANGE foram removidos de informação contextual, pois ela pode variar independentemente dos agentes. Além disso, os agentes somente percebem o valor de uma informação contextual. Na classe CONTEXTUALINFORMATIONDEFINITION, foram criados os atributos TYPE e SOURCE. De acordo com [Henricksen *et al*. 2002, Cortese *et al*. 2005], informações contextuais podem ser temporais, atemporais, sentidas ou inferidas. O atributo TYPE foi criado para armazenar o tipo de informação contextual, de acordo com esse esquema de classificação. Já o atributo SOURCE foi criado para indicar a fonte utilizada para a captura da informação contextual. A associação CANCHANGE foi removida, pois considera-se que os agentes não possuem autonomia para modificar o estado de uma informação contextual.

Em relação aos aspectos internos do agente (Figura 2), foram feitas as seguintes alterações: a classe SENSORDEFINITION foi criada para especificar a estrutura de um sensor em tempo de projeto, incluindo o seu padrão de captura de informações. O FAML não indica como capturar informações no ambiente. Também foi criado um relacionamento entre SENSORDEFINITION e AGENTDEFINITION. Este relacionamento indica que um agente pode possuir zero ou mais definições de sensores. A classe FACETACTIONSPECIFICATION foi removida, devido a remoção do conceito FACET. Essa remoção justifica-se, pois agora um agente não pode alterar uma informação contextual. Com a remoção dessa classe, também foi removida a referência à classe FACET-DEFINITION (que consistia originalmente no relacionamento CHANGES). A classe RESOURCEACTIONSPECIFICATION foi criada para permitir que os agentes alterem, removam e publiquem novos recursos no ambiente. Essa classe herda da classe ACTIONSPECIFICATION e é associada com a classe RESOURCESPECIFICATION através do relacionamento CHANGES, indicando que essa ação pode modificar um recurso.

3.2. Conceitos em Tempo de Execução

Em tempo de execução (Figura 3), ENVIRONMENT representa o conceito principal. No FAML, ambientes são gerados a partir do elemento SYSTEM, através do relacionamento

GENERATES. Esse relacionamento pode dificultar a construção de sistemas multiagentes abertos, composto por ambientes heterogêneos. A semântica do termo *generates* tende a definir SMAs onde os ambientes são definidos a *priori* (em tempo de projeto). Devido a isso, o relacionamento entre SYSTEM e ENVIRONMENT foi alterado para ISCOMPOSEDOF, significando agora que um sistema é composto por um ou mais ambientes.

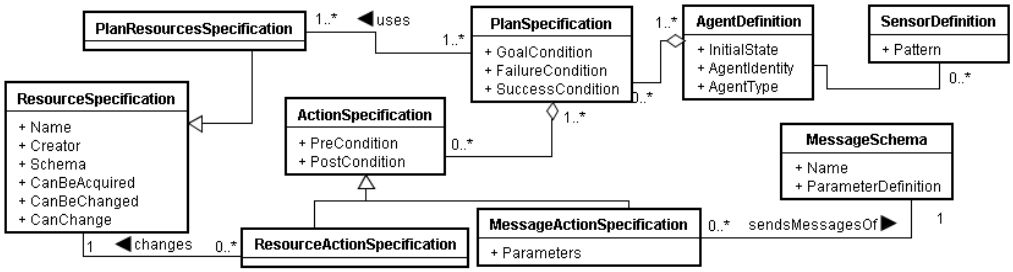

Figura 2: Classes internas ao agente em tempo de projeto (adaptado de [Beydoun 2009])

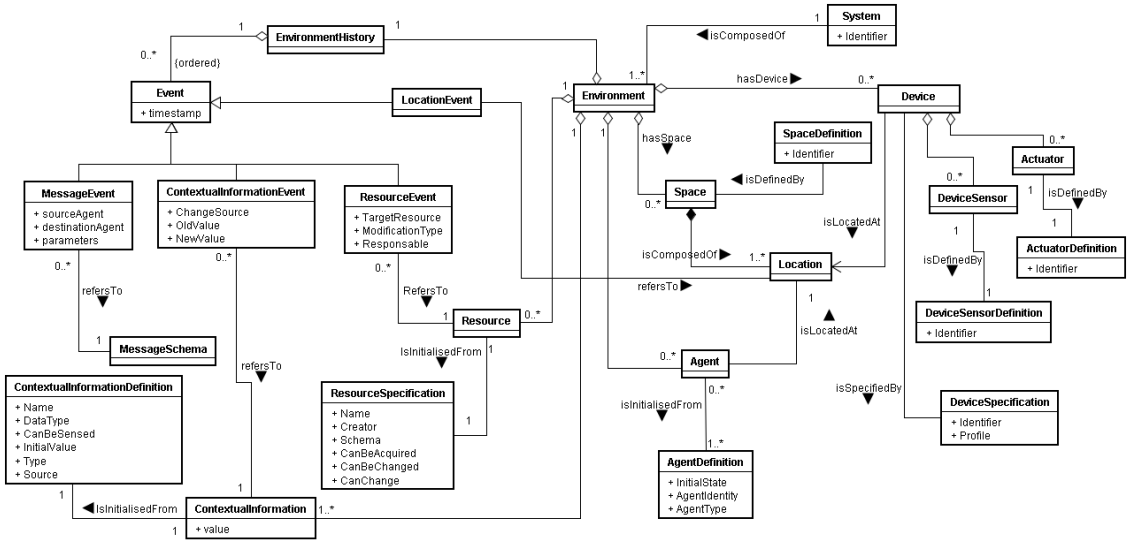

Figura 3: Classes externas ao agente em tempo de execução (adaptado de [Beydoun 2009])

O ambiente é essencial para o agente existir e atuar. O ambiente também restringe todas as informações contextuais que podem ser percebidas pelos agentes. Uma informação contextual (CONTEXTUALINFORMATION) é qualquer informação ou propriedade do ambiente que pode ser sentida pelos agentes. Um ambiente também hospeda um conjunto de recursos (RESOURCE). Recursos são elementos que possuem um nome, um criador, uma representação e podem ser adquiridos, compartilhados ou produzidos. Em um ambiente ubíquo, também é necessária a especificação de todos os espaços presentes no ambiente. Espaços agrupam elementos e são representados pela classe SPACE. Espaços são compostos por localizações definidas através da classe LOCATION. Uma localização serve para localizar dispositivos, agentes, e para delimitar o acesso aos recursos no ambiente. Dispositivos são definidos através da classe DEVICE. Cada dispositivo pode possuir seus próprios sensores e atuadores, definidos respectivamente pelas classes DEVICESENSOR e ACTUATOR. Adicionalmente, um ambiente também possui histórico, composto por todos os eventos que ocorreram. Eventos (classe EVENT), nesse caso, podem ser de mensagens (quando mensagens são trocadas por agentes), de informações contextuais (quando o estado de uma informação

contextual é alterado), recursos (quando um recurso é modificado, compartilhado ou removido), ou de localização (quando dispositivos ou agentes mudam de localização).

Os aspectos internos ao agente (Figura 4) foram alterados como a seguir. A classe RESOURCEACTION foi incluída baseada na classe RESOURCEACTION-SPECIFICATION, definida em tempo de projeto para permitir que os agentes alterem, removam e publiquem recursos em tempo de execução. O relacionamento de herança foi criado, visto que RESOURCEACTION representa um tipo de ação executada pelo agente. O relacionamento entre RESOURCEACTION e RESOURCE foi criado para indicar que esta ação é capaz de modificar um recurso.

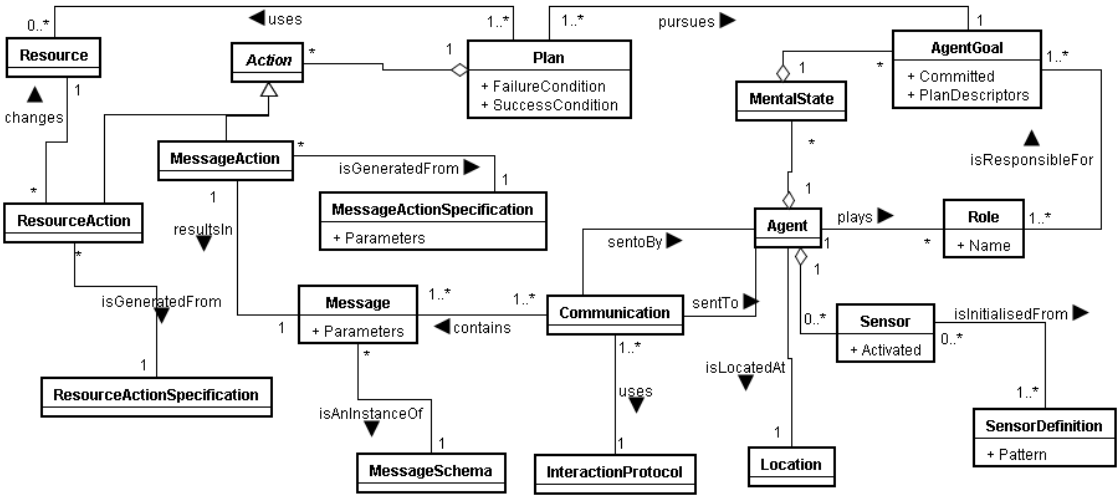

Figura 4: Classes internas ao agente em tempo de execução (adaptado de [Beydoun 2009])

A classe RESOURCEACTIONSPECIFICATION foi criada para indicar que uma RESOURCEACTION é gerada a partir (inclusão do relacionamento ISGENERATEDFROM) de uma RESOURCEACTIONSPECIFICATION. As classes FACETACTION, FACETACTION-SPECIFICATION e FACET foram removidas devido a substituição pelas classes referentes a informações contextuais. A classe SENSOR foi criada para representar as estruturas utilizadas pelos agentes para capturar informações no ambiente. Essa classe possui o atributo ACTIVATED, indicando se o sensor está capturando ou não informações no ambiente. Um sensor é inicializado a partir de um SENSORDEFINITION, definido em tempo de projeto. Em tempo de execução, um agente pode possuir zero ou mais sensores. A classe LOCATION representa uma localização ou ponto de interesse. Um agente está sempre consciente de sua localização através do seu relacionamento com a classe LOCATION.

3.3. Instanciação do Meta-modelo

A fim de ilustrar os conceitos do meta-modelo proposto, foi utilizado o sistema CVIS (*Cooperative Vehicle Infrastructre System*) [CVIS 2010] como base para a construção de um SMA, que envolve tanto os conceitos de agentes como os de computação ubíqua. Nas próximas seções, são descritas a aplicação CVIS e como agentes podem ser aplicados de acordo com o meta-modelo.

3.3.1. Aplicação CVIS

O CVIS é um projeto de desenvolvimento e pesquisa cujo objetivo é projetar, desenvolver e testar tecnologias capazes de permitir a interação entre veículos e a infra-

estrutura de sinalização de trânsito [CVIS 2010], melhorando e criando novos serviços para o controle de tráfego [Koenders & Vreeswijk 2008].

Com o CVIS, motoristas podem influenciar diretamente o sistema de controle de tráfego, podendo, por exemplo, serem guiados individualmente ao longo da rota até seus destinos [Koenders & Vreeswijk 2008]. Informações sobre as rodovias, exibidas em painéis na estrada, estarão disponíveis e acessíveis para serem exibidas aos condutores em seus veículos. Além de informações sobre a rodovia, por exemplo, podem ser exibidos alertas aos motoristas sobre a aproximação de um veículo em emergência. A próxima seção ilustra o CVIS utilizando os conceitos do meta-modelo. Os conceitos do meta-modelo são indicados utilizando um formato de fonte diferente.

3.3.2. Aplicando Agentes Segundo o Meta-modelo

Um sistema veicular cooperativo compreende todos os elementos envolvidos em um sistema de trânsito, como equipamentos de sinalização, veículos, pedestres, etc. Nesse sistema (SYSTEM), equipamentos de trânsito (como placas e semáforos) são dispositivos (DEVICE) capazes de interagir no ambiente. Alguns outros dispositivos podem fazer parte do ambiente, como por exemplo, computadores de bordo dos veículos e celulares. Cada um desses dispositivos possui uma localização (LOCATION). Essa localização pode ser invariante, como no caso dos semáforos, ou variantes, como no caso dos veículos e celulares. A localização destes dispositivos pode ser expressa através de um sistema GPS. O GPS é um sistema de posicionamento (POSITIONINGSYSTEM) que utiliza latitude e longitude para especificar uma determinada posição (POSITION).

Além de comunicar-se, dispositivos possuem outras funcionalidades, providas por seus sensores e atuadores. Os sensores (DEVICESENSOR) são responsáveis pela captura de algum tipo de informação no ambiente; em contraste, os atuadores (ACTUATOR) são responsáveis pela alteração do estado do ambiente (alterando o estado de alguma propriedade de um dispositivo). Em um sistema de controle de tráfego, existem sensores em semáforos, veículos, etc. Esses sensores podem, por exemplo, perceber quando um carro ou uma pessoa aproxima-se de uma faixa de segurança. Da mesma forma, atuadores podem ser usados em equipamentos de iluminação nos carros, em semáforos alertando os pedestres, etc.

Um sistema também define todos os espaços (SPACE) que fazem parte do ambiente (ENVIRONMENT). Um espaço é uma composição de localizações. Elas podem ser quarteirões, ruas, faixas de segurança, prédios, entre outros. São nesses espaços que os dispositivos estão presentes, e onde os agentes (AGENT) atuam. Os agentes por sua vez, podem ser de vários tipos. Por exemplo, um agente representando os interesses de um usuário, atuando em seu celular, ou agentes especializados responsáveis por funções de controle de tráfego, controlando semáforos, equipamentos de sinalização, etc. Esses agentes interagem trocando mensagens e executando ações, a fim de melhorar o sistema de trânsito, assim como buscando satisfazer seus próprios objetivos (como por exemplo, os objetivos de seus usuários). Para capturar informações no ambiente (como por exemplo, mensagens e informações contextuais), os agentes possuem sensores (SENSOR). Um ambiente também define o conjunto de informações contextuais (CONTEXTUALINFORMATION). Uma informação contextual pode ser o número total de dispositivos em uma determinada localização, a temperatura, etc. Da mesma forma, recursos (RESOURCE) podem ser definidos. Um exemplo de recurso é um ponto de acesso à Internet. Este ponto de acesso possui área de cobertura e pode ser utilizado em uma determinada região.

4. Trabalhos Relacionados

A literatura provê muitos trabalhos que propõem linguagens para a modelagem de sistemas ubíquos, suporte para a modelagem de informações contextuais [Henricksen *et al*. 2002, Dey 2000], e processos para o desenvolvimento de aplicações ubíquas [Muñoz *et al*. 2004].

O PervML [Serral *et al*. 2010] é uma linguagem de modelagem para a representação de contexto em sistemas ubíquos em alto nível de abstração. Ela é uma linguagem específica de domínio para a especificação de sistemas ubíquos, utilizando primitivas conceituais (tais como serviços, *triggers*, interações, etc.). A linguagem oferece modelos para a descrição dos tipos de serviços que o sistema deve realizar, os diferentes serviços para cada tipo providos pelo sistema, como estes serviços interagem, e, políticas de privacidade e dados pessoais de usuários. Serviços são utilizados para abstrair elementos do mundo real e funcionalidades que o sistema oferece, como por exemplo, a intensidade luminosa e a identificação da presença de pessoas em um ambiente são exemplos de serviços.

Dentre diversas abordagens para a modelagem de informações contextuais, selecionamos os trabalhos de [Henricksen *et al*. 2002, Chen *et al*. 2003]. Henricksen e co-autores (2002) propõem uma coleção de conceitos projetados para capturar diferentes tipos de informações contextuais que são relevantes para a construção de um sistema ubíquo. Nesse trabalho, contexto refere-se a "circunstâncias ou situações no qual uma tarefa computacional ocorre". Nessa abordagem, elementos são representados através de entidades físicas ou conceituais, tais como pessoas, dispositivos ou canais de comunicação. O trabalho de Chen e co-autores (2003) utiliza uma ontologia para descrever o contexto. Nela são descritos conceitos como localização, atributos do ambiente (como por exemplo temperatura), pessoas, dispositivos, objetos e agentes de software. Além desses conceitos, também são descritos os serviços oferecidos e as atividades atribuídas para as pessoas, além de entidades computacionais, seus papéis, crenças e intenções. O trabalho de [Muñoz *et al*. 2004] propõe um método baseado em MDA (*Model Driven Architecture*) para o desenvolvimento de aplicações ubíquas. Nele incluem técnicas para cada fase de um processo de desenvolvimento baseado em MDA, que compreende à: uma linguagem para a descrição em alto nível dos elementos que compõem o sistema; o modelo para a transformação dos elementos em código fonte; e os *templates* para a geração de código fonte.

Na literatura não foram encontrados trabalhos provendo um meta-modelo para a construção sistemas multiagentes ubíquos. Nossa pesquisa incluiu os mecanismos de busca das principais bibliotecas digitais, como *ACM Digital Library*, IEEEXplore e *Science Direct*. A *string* de busca utilizada foi padronizada e adequadamente adaptada para cada mecanismo de busca, a fim de evitar inconsistência nos resultados obtidos. Os termos da *string* restringem o domínio de busca, através de termos como: *pervasive* e *ubiquitous computing*, *agent* e *multiagent systems*, e, *meta-model*.

5. Conclusão

Neste artigo foi apresentado um meta-modelo que suporta o desenvolvimento de aplicações baseadas em agentes para ambientes ubíquos. Nosso trabalho visa tornar os conceitos de computação ubíqua no mesmo nível de abstração dos conceitos de agentes e sistemas multiagentes, tornando-os inerentes aos agentes. O U-MAS foi construído através da extensão do meta-modelo FAML, a fim de serem incluídos os conceitos de computação ubíqua.

O meta-modelo FAML não cobre requisitos fundamentais necessários para a adição de conceitos relacionados à computação ubíqua; dessa forma, ele não suporta diretamente a modelagem de uma aplicação ubíqua, como ilustrado neste artigo. Com a extensão do FAML, buscamos prover aos desenvolvedores um meta-modelo capaz de modelar aplicações baseadas em agentes para ambientes ubíquos de uma forma unificada. Assim como no FAML, o meta-modelo proposto é independente de metodologia. Isso significa que ele pode ser utilizado em projetos de desenvolvimento baseados em qualquer metodologia de agentes ou de computação ubíqua. O meta-modelo também visa ser utilizado como entrada para projetos de desenvolvimento baseados em modelos, onde, após a aplicação de transformações baseadas em regras [Pastor & Molina 2007], código-fonte possa ser gerado para uma plataforma de SMA.

Como trabalhos futuros, estuda-se a construção de uma linguagem de programação baseada no meta-modelo proposto. A idéia é tornar a sintaxe dessa linguagem similar a uma linguagem de programação difundida, como Java. Também espera-se possibilitar a geração de código, utilizando regras de transformação seguindo um ciclo do desenvolvimento baseado em MDD. Assim como o meta-modelo, essa linguagem também será independente de metodologia. O código gerado poderá ser transformado para primitivas de uma plataforma de desenvolvimento de SMAs, como por exemplo, o JADE [JADE 2010] e o SemantiCore [Blois *et al*. 2007].

Referências

Ark, W., Selker, T. (1999). A Look at Human Interaction with Pervasive Computers. IBM Systems Journal, 38(4) (pp. 504-507).

Barron, P. and Cahill, V. (2006). YABS: a domain specific language for pervasive computing based on stigmergy. In *Proceedings of the 5th international conference on Generative programming and component engineering* (GPCE '06) (pp. 285-294). ACM, New York, NY, USA. doi: 10.114./1173706.1173730

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J. And Gonzalez-Perez, C. (2009). FAML: A Generic Metamodel for MAS Development. In IEEE Transactions on Software Engineering (Vol. 99, pp.841-863).

Blois, M., Escobar, M. and Choren, R (2007). Using Agents and Ontologies for Application Developmenton the SemanticWeb. Journal of the Brazilian Computer Society (Vol. 1, pp. 1-15).

Campiolo, R., Cremer, V. and Sobral, J. B. M. (2007). On modeling for pervasive computing environments. In *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems* (MSWiM '07). ACM, New York, NY, USA (pp. 240-243). doi:10.1145/1298126.1298168

Chen, H., Finin, T. and Joshi, A. (2003). An ontology for context-aware pervasive computing environments. The Knowledge Engineering Review (Vol. 18-3, pp. 197-207).

Cortese, G., Davide, F. and Lunghi, M. (2005). Context Awareness for Physical Service Environments. Ambient Intelligence (Vol. 6, pp. 71-97).

CVIS Project. Disponível em: http://www.cvisproject.org.

Gunasekera, K., Zaslavsky, A., Loke, S.W. and Krishnaswamy, S. (2008). Context Driven Compositional Adaptation of Mobile Agents. In *Ninth International Conference on Mobile Data Management Workshops* (MDMW' 08) (pp. 201-208).

Hansmann, U., Merk, L., Nicklous, M. and Stober, T. (2001). Pervasive computing handbook. Springer-Verlag New York, Inc., New York, NY, USA.

Henricksen, H., Indulska, J. and Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. In *First International Conference on Pervasive Computing* (pp. 167-180).

Ilyas, M. and Mahgoub, I. (2004). Mobile computing handbook. CRC Press (1st edition).

JADE - Java Agent DEvelopment Framework. Disponível em: http://jade.tilab.com/.

Jansen, E., Abdulrazak, B., Yang, H., King, J. and Helal, S. (2005, December). A Programming Model for Pervasive Spaces. In *proceedings of 3rd International Conference on Service Oriented Computing* (pp. 12-15). Amsterdam, Netherlands.

Jennings, N.R. (2001). An agent-based approach for building complex software systems. Communications of the ACM (44(4), pp. 35-41).

Koenders, E. and Vreeswijk, J. (2008). Cooperative Infrastructure. In IEEE Intelligent Vehicles Symposium (pp. 721-726).

Kurkovsky, S. (2007, December). Pervasive computing: Past, present and future. Information and Communications Technology, ICICT at the 5th International Conference on ITI (pp. 65-71). doi: 10.1109/ITICT.2007.4475619

Muñoz, J., Pelechano, V. and Fons, J. (2004). Model Driven Development of Pervasive Systems. In *Intl. Workshop on Model-Based Methodologies for Pervasive and Embedded Software* (MOMPES) (pp 3-14). Canada: Hamilton.

Pastor, O. and Molina, J. C. (2007). Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Saha, D. and Mukherjee, A. (2003, March). Pervasive Computing: A Paradigm for the 21st Century. Computer 36, (25-31). doi: http://dx.doi.org/10.1109/MC.2003.1185214

Satyanarayanan, M. (2001, August). Pervasive computing: Vision and challenges. In IEEE Personal Communications (Vol. 8, pp. 10-17).

Serral, E., Valderas, P. and Pelechano, V. (2010, April). Towards the Model Driven Development of contextaware pervasive systems. In Pervasive Mob. Comput. 6 (pp. 254-280). doi=10.1016/j.pmcj.2009.07.006

Seyler, F., Taconet, C. and Bernard, G. (2007). Context Aware Orchestration Meta-Model. In *Proceedings of the Third International Conference on Autonomic and Autonomous Systems* (ICAS '07) (pp. 17-). IEEE Computer Society, Washington, DC, USA. doi:10.1109/CONIELECOMP.2007.69

Silva, N., Rocha, J. and Cardoso J. (2003). E-Business interoperability through ontology semantic mapping. In *Proceedings of Processes and Foundations for Virtual Organizations* (pp. 315-322).

Symonds, J. (2010). Ubiquitous and Pervasive Computing: Concepts, Methodologies, Tools, and Applications. New Zealand: Auckland University of Technology.

Mark Weiser. (1999). The computer for the 21st century. SIGMOBILE Mob. Comput. Commun. Rev.3, 3 (pp. 3-11). doi:10.1145/329124.329126

Weiss, G. (1999). Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence. MIT Press.