MAS-ML Tool: Um Ambiente de Modelagem de Sistemas Multi-Agentes

Kleinner Farias[1], Ingrid Nunes[1], Viviane Silva[2], Carlos Lucena[1]

[1]Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
R. Marquês de São Vicente, 225 RDC – RJ – Brasil

[2]Departamento de Ciência da Computação
Universidade Federal Fluminense (UFF)
R. Passo da Pátria, 156 São Domingos – Niterói – RJ – Brasil

{kfarias,ionunes,lucena}@inf.puc-rio.br, viniane.silva@ic.uff.br

Resumo. *Sistemas Multi-agentes (SMAs) surgiram como uma abordagem promissora para o desenvolvimento de sistemas complexos e distribuídos. Entretanto, para que essa abordagem seja efetivamente explorada em um contexto industrial é essencial que ferramentas dêem suporte ao desenvolvimento de SMAs. Portanto, existe a necessidade de ferramentas para a modelagem de SMAs, dado que criar e manipular modelos sem o apoio de um ambiente adequado são tarefas tediosas e que demandam tempo. Este artigo visa atacar essa necessidade através da construção de um ambiente de modelagem específico de domínio para SMAs, implementado como um plug-in da plataforma Eclipse. O ambiente é baseado na MAS-ML, uma linguagem de modelagem de SMAs. Dentre os diagramas estruturais definidos na linguagem, este trabalho concentra-se na implementação do ambiente para a construção de diagramas de classes.*

1. Introdução

A crescente complexidade de sistemas de software modernos cria a necessidade do surgimento de novas tecnologias e de novos paradigmas de desenvolvimento. Alguns dos requisitos que estão presentes nesse tipo de sistemas consistem em: (1) operar em ambiente distribuído com diferentes tipos de dispositivos (por exemplo, rede sensores, celulares, e computadores pessoais); (2) ter comunicação e troca de dados de diferentes tipos (por exemplo, áudio e vídeo); e (3) ter a capacidade de adaptar-se dinamicamente perante as mudanças de requisitos e do ambiente de operação. Sendo assim, projetar, analisar e implementar sistemas modernos utilizando os paradigmas de desenvolvimento centrados em código, requer um maior esforço, custo elevado, e um nível de dificuldade significativo.

Um fator significante por trás dessa dificuldade é a existência de uma lacuna conceitual entre o domínio do problema e domínio da solução. Tentar preencher essa lacuna com abordagens que exigem extensiva habilidade de implementação dos desenvolvedores pode dar origem a complexidades inesperadas que tornam o projeto, a análise e a implementação ainda mais custosos e difíceis. Diante desse cenário, os Sistemas Multi-Agentes (SMAs) surgem como uma abordagem na tentativa de diminuir

essa lacuna através da decomposição do problema em abstrações de mais alto nível, os *agentes* de software. De acordo com [1], SMAs podem ser entendidos como sociedades de agentes em que entidades autônomas, pró-ativas e heterogêneas podem trabalhar de forma conjunta para fins similares ou totalmente distintos.

SMAs têm se tornado um poderoso paradigma de engenharia de software [2] e têm sido aplicados com sucesso para o desenvolvimento de diferentes tipos de sistemas de software tanto na academia quanto na indústria [3], [4]. Esse sucesso é devido, em parte, à capacidade de lidar não apenas com abstrações e conceitos presentes no domínio do problema, como também com questões e conceitos presentes no domínio da solução. Além disso, SMAs permitem trabalhar de modo satisfatório em domínios de aplicações onde integração de sistemas e mudanças de requisitos são freqüentes.

Neste artigo defende-se que o processo de analisar um problema, encontrar uma solução, e expressá-la em uma linguagem de programação de alto nível pode ser visto como uma forma implícita de modelagem [5]. Com isso em mente, é possível argumentar que o desenvolvimento de software é essencialmente uma atividade de resolver problemas baseando-se em modelos. Conseqüentemente, a fim de se desenvolver aplicações seguindo o paradigma de SMAs, é necessário realizar uma atividade essencial: a modelagem de SMAs. Desse modo, as linguagens de modelagem para SMAs desempenham um papel central dentro do processo de desenvolvimento. Assim, esse trabalho representa um esforço no desenvolvimento de um ambiente de modelagem específico de domínio que torne possível modelar SMAs usando as definições e as abstrações especificadas na linguagem MAS-ML (*Multi-Agent System Modeling Language*) [6], [7], [8], [9]. A escolha da linguagem MAS-ML para a construção do ambiente foi motivada pelo seu reconhecimento perante a academia, pela necessidade de modelagem de SMAs em sistemas reais – ilustrada por estudos de casos como, por exemplo, OLIS [10] e ExpertCommittee [11] – e por apresentar as definições necessárias para modelar os conceitos e abstrações apresentadas no paradigma de SMAs.

A ausência de um ambiente de modelagem para MAS-ML que permita expressar soluções usando uma linguagem de modelagem de SMAs pode ocasionar alguns problemas. Primeiro, isto pode comprometer, de fato, o processo de projetar, analisar e implementar sistemas de software seguindo o paradigma de SMAs. Segundo, o tempo e esforço necessários para fazer a modelagem manualmente pode fazer com que essa tarefa não seja devidamente realizada, causando possivelmente retrabalho na implementação do sistema.

O ambiente MAS-ML foi construído tomando como base a definição da linguagem de modelagem MAS-ML, a qual estende a UML (*Unified Modeling Languagem*). MAS-ML define três diagramas estruturais: o diagrama de classes estendido, de papéis e de organizações. Até o presente momento, este trabalho concentrou-se na implementação do ambiente para a construção de diagrama de classes como um *plug-in* da plataforma Eclipse [28].

Este artigo é organizado como segue. Na Seção 2 é apresentada uma fundamentação teórica, na qual a linguagem MAS-ML é introduzida, bem como seu diagrama de classes. Na Seção 3 é apresentado o ambiente proposto nesse artigo, descrevendo alguns dos seus benefícios, limitações e mencionando algumas questões

sobre a implementação. E por fim, na Seção 4, são apresentadas as conclusões e os trabalhos futuros.

2. Fundamentação Teórica

Nesta seção são apresentados os principais conceitos relacionados com a linguagem MAS-ML, juntamente com uma breve descrição do seu diagrama de classes.

2.1. MAS-ML

A UML (*Unified Modeling Language*) [12] é uma linguagem de modelagem que tem sido usada tanto na indústria quanto na academia e foi adotada como padrão para a modelagem de sistemas de software desenvolvidos seguindo o paradigma de orientação a objetos (OO). A UML tem um grande escopo de uso, podendo ser usada em um conjunto diversificado de domínio de aplicações. Porém, quando aplicada para modelar SMAs, a UML tem-se mostrado insuficiente. Entre outros problemas apresentados, é possível destacar, por exemplo, que o metamodelo da UML não oferece suporte à modelagem de agentes, seus papéis e organizações; entidades tipicamente definidas em SMAs [9]. Além disso, o uso de estereótipos da UML não é suficiente para modelar SMAs, pois tais entidades possuem relacionamentos e características específicos. Estereótipos estendem as capacidades de modelagem através da extensão semântica e não estrutural das metaclasses existentes.

Diante desse cenário, a MAS-ML visa solucionar essas deficiências através de uma extensão da UML, ou seja, ela introduz no metamodelo da UML as entidades necessárias para modelar SMAs, bem como define relacionamentos entre as mesmas. Uma extensão da UML para SMAs deve oferecer recursos para os desenvolvedores a fim de que diferentes características e perspectivas do software desenvolvido possam ser representadas. Especialmente, as características relacionadas aos aspectos dinâmicos e estruturais do software. Por um lado, os aspectos estruturais definem as entidades, suas propriedades e seus relacionamentos. Por outro, os aspectos dinâmicos representam o comportamento que o software possa apresentar.

Na MAS-ML, tanto o aspecto estático quanto o dinâmico são contemplados. Em outras palavras, com a linguagem [9] é possível modelar os aspectos dinâmicos e estruturais definidos no *framework* conceitual TAO (*Taming Agents and Objects*) [6] [8] [9]. Sua especificação é centrada na definição do metamodelo da linguagem que foi cuidadosamente projetado e integrado ao metamodelo da UML. Esta integração representa a extensão propriamente dita, a qual é feita de acordo com os conceitos definidos no TAO.

A linguagem define os diagramas estruturais (estáticos) e dinâmicos para representar todos os aspectos do TAO. Os diagramas estruturais definidos na MAS-ML são diagramas de classes estendido, papéis e organizações. Com tais diagramas, é possível modelar os aspectos estruturais de todas as entidades definidas no TAO. Este trabalho concentrará seus esforços no diagrama de classes que será discutido a seguir.

2.2. Diagrama de Classes da MAS-ML

O diagrama de classes apresentado na definição do metamodelo da UML [12] trata-se do diagrama utilizado para representar o conceito de “classe”, o qual possui estado (atributos) e comportamento (métodos), juntamente com os tipos de relacionamento

estático que podem ser estabelecidos entre as classes, como, por exemplo, associação, composição e agregação [9], [12]. MAS-ML estende o diagrama de classes da UML com o objetivo de inserir as entidades e os relacionamentos presentes em SMAs, e até então ausentes na UML. Como o foco deste trabalho se detém na extensão do diagrama de classes será dada uma maior atenção ao mesmo.

De acordo com [9], o diagrama de classes estendido representa os relacionamentos entre as (i) classes e os ambientes, (ii) classes e agentes e (iii) classes e organizações. Além disso, ele também foi estendido para representar os relacionamentos entre agentes, entre ambientes e entre organizações. As entidades que podem participar desse diagrama, além dos definidos pela UML, são: agentes, organização e ambiente. Os relacionamentos usados nesse diagrama são aqueles definidos na UML, mais os relacionamentos novos listados a seguir:

- *inhabit* – usados entre classes e classes de ambiente;
- *association* – usado entre classes do agente e classes, entre classes e classes de organização e entre classes de ambiente; e
- *specialization* – usado entre classes do agente, entre classes de organização e entre classes de ambiente.

3. MAS-ML Tool

Nesta seção são apresentadas as funcionalidades, as tecnologias utilizadas e os detalhes relacionados à implementação do ambiente de modelagem específico de domínio, identificado como MAS-ML *Tool*.

MAS-ML *Tool* é um ambiente de modelagem específico de domínio que atende a modelagem de sistemas multi-agentes de acordo com a especificação do metamodelo definido em MAS-ML [8], [9]. O objetivo da MAS-ML *Tool* é fornecer um ambiente de modelagem em que desenvolvedores possam trabalhar com os conceitos do domínio do problema, ao mesmo tempo que utilizem explicitamente conceitos definidos no domínio da solução, neste caso os conceitos e abstrações definidas no paradigma de SMAs. Partindo deste ponto de vista, é possível reduzir a lacuna entre o domínio do problema e a representação de sua solução no paradigma de SMAs.

Os agentes definidos na MAS-ML *Tool* seguem o modelo BDI (*belief-desire-intention*) [13], dado que MAS-ML baseia-se nessa arquitetura. Assim, os agentes possuem um conjunto de crenças, que definem seu estado, um conjunto de objetivos, que definem o estado que estes desejam alcançar (suas motivações), e intenções, que definem o componente deliberativo do sistema. Intenções são os objetivos que o agente está comprometido efetivamente a atingir, tendo escolhido um plano para isso. Assim, agentes MAS-ML possuem um conjunto de planos, os quais são compostos por ações. O modelo BDI vem sendo bastante utilizado para modelar agentes cognitivos, e suas vantagens incluem: usado com sucesso em diversas aplicações, baseado em fundamentos filosóficos e possuem diversas plataformas que o implementam, como, por exemplo, Jadex, Jason e 3APL.

3.1. Desenvolvimento da MAS-ML *Tool*

O ambiente implementado trata-se de um *plug-in* da plataforma *Eclipse* [14], como mencionado anteriormente. Isso implica, por um lado, permitir fazer uso dos recursos oferecidos pela plataforma, e por outro, permitir que os usuários trabalhem com

modelagem de SMAs ao mesmo tempo em que usam a plataforma *Eclipse*. Dado que muitas plataformas de agentes são implementadas em Java, tais como JADE, Jadex, Jason, o uso da plataforma *Eclipse* também facilita uma possível geração de código dentro do mesmo ambiente de desenvolvimento. O processo de desenvolvimento da ferramenta pode ser resumido pela necessidade de ligar dois pontos (lacunas): o domínio do problema (o metamodelo da linguagem) ao domínio da solução (a MAS-ML *tool*). Ou seja, deseja-se a partir do metamodelo da linguagem derivar a ferramenta de modelagem. Para isto, foi utilizada uma abordagem de desenvolvimento dirigido por modelos, onde o modelo central e de maior abstração é o próprio metamodelo da MAS-ML. Diante deste cenário, tem-se o metamodelo guiando todo o processo de desenvolvimento, sendo transformado até chegar ao objetivo final, a MAS-ML *tool.* A seguir, são descritas, de forma resumida, as cinco etapas realizadas no processo de desenvolvimento:

1. **Definição do Modelo de Domínio.** Primeiro, o metamodelo foi especificado usando o EMOF (*Essencial Meta-Object Facility*), uma linguagem de definição de metamodelo usada para definição da UML. O EMOF é parte integrante do *plug-in* EMF (*Eclipse Metamodel Framework*) que consiste em um *framework* de modelagem e de geração de código para construir aplicações baseadas em modelos [16].
2. **Definição do Modelo Gráfico.** Esta etapa concentra-se na definição das entidades e de suas propriedades, assim como de seus relacionamentos que podem ser expressos na ferramenta. Para isto, foram levadas em consideração as entidades e os relacionamentos definidos no metamodelo da linguagem.
3. **Definição do Modelo de Ferramenta.** Esta etapa é dedicada para a especificação de quais elementos farão parte da paleta da ferramenta. Para isso, essa etapa recebe como entrada o modelo de domínio e o modelo gráfico determinados anteriormente.
4. **Definição do Mapeamento.** A atividade desenvolvida nesta etapa se concentra na construção de um mapeamento entre os três modelos: modelo de domínio, modelo gráfico, e o modelo de ferramenta. O mapeamento gerado é usado como entrada em um processo de transformação que tem como objetivo criar um modelo específico de plataforma.
5. **Geração da Ferramenta.** Seguindo uma abordagem generativa [15], o próximo passo consiste na geração do código da ferramenta levando em consideração o modelo específico de plataforma gerado na etapa anterior. Nesta etapa, é feito uso do *plug-in* do GMF o qual fornece um componente generativo e uma infra-estrutura *runtime* para desenvolver editores gráficos baseado no EMF e GEF (*Graphical Eclipse Framework*) [18]. A integração entre o GMF e GEF é fundamental no desenvolver do editor gráfico tomando como base o metamodelo da MAS-ML [17].

A Figura 1 mostra uma visão geral da MAS-ML *Tool* juntamente com alguns dos seus componentes destacados com letras. Cada componente desempenha uma funcionalidade específica dentro da ferramenta, sendo discutida brevemente a seguir:

- ***Package Explorer* (A).** Para cada novo projeto de modelagem que é criado, surge a necessidade de criar e importar arquivos que são utilizados durante o processo de modelagem (como, por exemplo, bibliotecas, documentos de texto, imagens, e etc). O *package explorer* tem como funcionalidade central permitir a organização dos arquivos em uma estrutura de árvore a fim de que seja possível um melhor gerenciamento e manipulação dos arquivos;
- ***Modeling View* (B).** Os modelos que são criados precisam necessariamente ser visualizados com o objetivo de atender dois requisitos básicos quando se usa modelos: compreensibilidade e a comunicação. Diante dessa necessidade, a *modeling view* permite os desenvolvedores visualizar e editar os modelos de forma interativa;
- ***Nodes Palette* (C).** Os construtores que podem fazer parte do diagrama de classes proposto pela MAS-ML encontram-se na *nodes palette.* Sendo assim, os desenvolvedores podem criar instâncias desses construtores, as quais são visualizadas na *modeling view.* Observando a Figura 1 é possível identificar alguns desses elementos como, por exemplo, *ActionClass* e *AgentClass*.
- ***Relationship Palette* (D).** Os relacionamentos que podem ser estabelecidos entre os construtores presentes na *nodes palette* são disponibilizados na *relationship palette.* Observando a Figura 1 é possível identificar alguns desses relacionamentos como, por exemplo, a associação estabelecida entre os dois agentes, *AgenteA* e *AgenteB*, mostrados na *modeling view*.
- ***Properties View* (E).** Preocupações sempre constante durante a elaboração do metamodelo da MAS-ML juntamente com a extensão da UML, consistem na: (1) identificação e representação das características dos conceitos presentes no paradigma SMAs dentro das metaclasses da linguagem (essas características são inseridas como propriedades das metaclasses); (2) organização e tratamento das propriedades que são herdadas das metaclasses que são estendidas da UML visando evitar a criação de inconsistência ou conflitos. Essas propriedades definem precisamente os modelos e são usadas, por exemplo, para diferenciar os modelos presentes nos diagramas. Observando a Figura 1, a diferença entre os dois agentes presentes na *modeling view* consiste, dentre outras coisas, na diferença das *Strings* "*AgenteA"* e o "*AgenteB"* atribuída à sua propriedade *name*. Desse modo, a importância da *properties view* se evidencia pelo fato de ela permitir manipular de forma precisa as propriedades dos modelos. Essa *view* exibe as propriedades definidas no metamodelo da MAS-ML quando o modelo é exibido e selecionado na *modeling view.* Na Figura 1 é possível observar as propriedades do *AgenteB.goal02.*
- ***Problems View (F).*** Diante da necessidade de validar os modelos criados em relação ao metamodelo da linguagem, é disponibilizada a funcionalidade de "validar modelo". Com isso, é possível verificar se existe alguma inconsistência no modelo criado em relação à definição do metamodelo da linguagem. Caso exista alguma inconsistência, ela será mostrada em *problems view*. Na Figura 1 são observados alguns problemas que foram capturados no momento da validação do modelo presente na *modeling view*, ou seja, o modelo apresenta inconsistências. Por exemplo, as regras de boa formação não respeitadas seriam: (1) todo agente deve ter uma ação; (2) todo agente deve ter um objetivo; (3) todo agente deve ter um plano; e assim por diante. Estas inconsistências são em relação ao modelo apresentado em (B). Esta funcionalidade é particularmente

importante para viabilizar o uso de modelagem de SMAs dentro do contexto do desenvolvimento dirigido por modelos, na qual modelos são vistos como artefatos de primeira ordem. Isto é devido ao fato de modelos inconsistentes prejudicar as transformações de modelos presentes em MDD.

- ***Outline View* (G).** Uma vez que um modelo tenha sido criado, um *overview* da distribuição dos elementos presentes no mesmo poderá ser visualizado através da *outline view*.

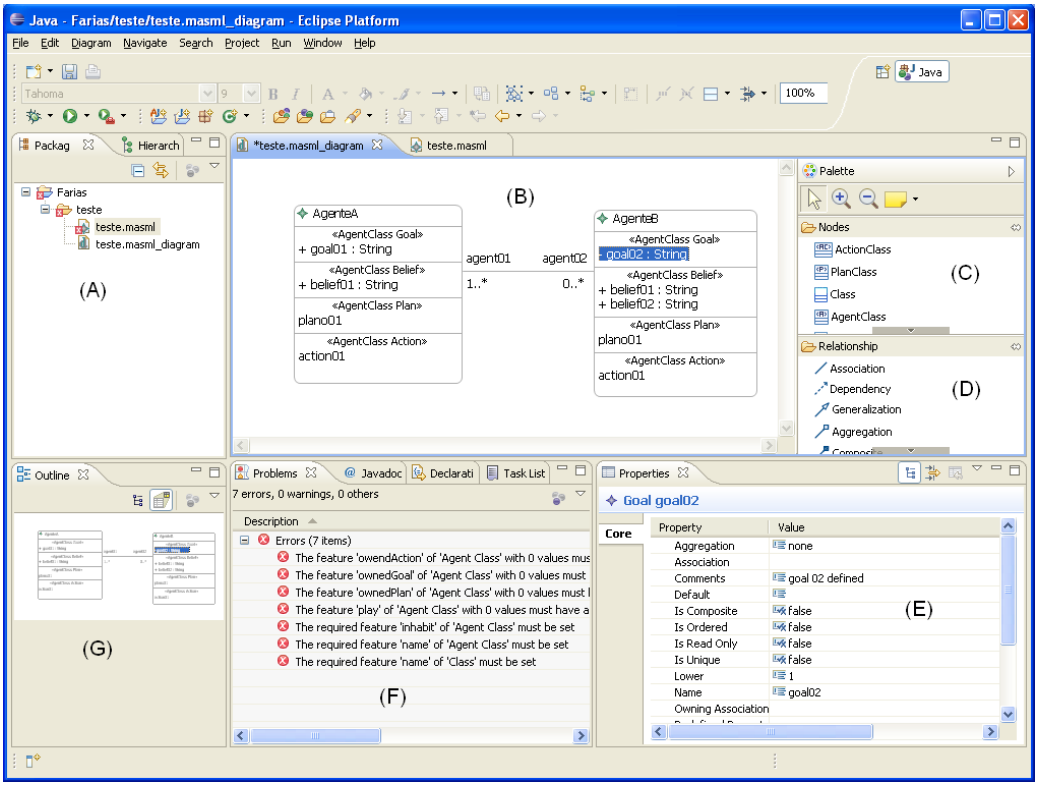

Figura 1. Visão Geral da MAS-ML *Tool*

3.2. Benefícios e Limitações

A falta de uma ambiente que dê suporte à linguagem de modelagem MAS-ML representa um ponto-chave na decisão do uso eficiente, ou não, da linguagem. Visto que modelar SMAs sem o apoio de uma ferramenta torna o trabalho difícil de ser realizado, custoso, e, podendo até certo ponto, ser considerado impraticável, é fundamental que ferramentas sejam propostas para que a linguagem seja efetivamente utilizada.

Os benefícios obtidos com o MAS-ML *Tool* emergem diante da resolução de alguns desafios encontrados até então. A seguir são destacados alguns desses desafios que são superados com a MAS-ML *Tool*:

- ***Conhecer a linguagem em detalhes:*** os usuários precisam conhecer a linguagem em detalhes para fazer uso correto da mesma. Ou seja, modelar um SMA implica em conhecer não apenas os conceitos e os relacionamentos definidos no metamodelo da linguagem como também às restrições aplicadas a esses elementos. Caso contrário, um modelo criado manualmente pode apresentar inconsistência ao ponto de torná-lo inválido. Capturando restrições da linguagem MAS-ML, validando diagramas de acordo com elas e exibindo problemas na *problems view* permite que mesmo desenvolvedores não experientes modelem diagramas consistentes;
- ***Dificuldade na representação dos elementos da linguagem:*** um dos motivos para o uso de linguagem de modelagem é a questão da comunicação que se deseja conseguir através da representação dos aspectos estáticos e dinâmicos do software. Para ter uma boa comunicação é fundamental que os modelos sejam de fácil compreensão e entendimento. Desse modo, se não existe um padrão de representação, ou as representações pré-definidas para os modelos definidos na linguagem não são seguidas, isto pode comprometer a compreensão e o bom entendimento dos modelos. Além disso, ambigüidade e inconsistência podem ser uma conseqüência do mau uso. Dessa forma, com o uso da ferramenta, os modelos tem uma representação padrão dos elementos da MAS-ML, além de serem mais facilmente desenhados do que utilizando ferramentas de desenho livre; e
- ***Validação do modelo:*** todo modelo criado deve ser validado em relação às restrições definidas no metamodelo da linguagem. Caso estas restrições não sejam respeitadas, o modelo criado não estará condizente com as definições do metamodelo, podendo até mesmo não representar um SMA. Além disso, caso inconsistências não sejam percebidas em tempo de projeto, encontrá-las em tempo de implementação pode ser uma tarefa muito mais custosa.

A MAS-ML *Tool* foi projetada de forma a ser estendida para incorporar os demais modelos estáticos da MAS-ML. Assim, como já foi apresentado anteriormente, uma das limitações atuais da ferramenta é que sua atual versão apenas contempla o diagrama de classes da MAS-ML. Além disso, a ferramenta ainda não realiza geração de código, funcionalidade já provida por algumas ferramentas de SMAs existentes (ver próxima seção). Nessa primeira versão da MAS-ML *Tool*, o foco foi desenvolver a ferramenta com uma arquitetura extensível para a incorporação dos demais modelos da MAS-PL e a captura do metamodelo do MAS-PL na ferramenta.

4. Trabalhos Relacionados

Nesta seção são discutidos os trabalhos relacionados os quais foram selecionados com base em dois critérios: a relevância do trabalho e a sua contribuição para o desenvolvimento do trabalho proposto. A análise é fundamentada em identificar e comparar as semelhanças dos recursos oferecidos, o foco dos trabalhos, a plataforma utilizada para o desenvolvimento, assim como as vantagens e desvantagens em relação ao trabalho aqui apresentado. O que é possível antecipar é que uma ferramenta de modelagem de SMAs para ser usada em um ambiente industrial, seria formada pela união das vantagens apresentadas em cada uma delas, antes que o uso de uma em particular.

PASSI ToolKit. PASSI é uma metodologia de desenvolvimento de sociedades de agentes, que usa técnicas de inteligência artificial e o paradigma de desenvolvimento orientado a objetos de uma forma integrada. A ferramenta PASSI *ToolKit* [22] visa dar suporte à PASSI, permitindo que os artefatos de software definidos em cada fase da metodologia possam ser construídos integrados a uma reconhecida ferramenta de modelagem, *Rational Rose*. PASSI, diferentemente de MAS-ML, é uma metodologia e não uma linguagem de modelagem, porém usa uma notação baseada na UML para a modelagem dos agentes e conceitos relacionados. Ela trabalha com um número maior de diagramas, o que é justificado devido à necessidade de contemplar as fases de desenvolvimento de SMAs. As diferenças da PASSI *ToolKit* e da MAS-ML *Tool* podem ser justificadas pela diferença no foco das duas ferramentas. A primeira visa dar suporte a uma metodologia de desenvolvimento orientada à SMAs, o que leva a ter preocupações com diferentes tipos de artefatos de software, desde documento de requisitos a código; enquanto a segunda é focada em uma linguagem de modelagem de SMAs, onde a prioridade é com a fase de projeto de SMAs. Por exemplo, a ferramenta PASSI *ToolKit*, visando suportar os artefatos de especificação de requisitos da metodologia PASSI, suporta a criação de (*i*) modelo de especificação de requisitos; (*ii*) descrição de requisitos do domínio; (*iii*) identificação de agentes; (*iv*) identificação de papeis; e entre outros. Porém, em comum com a MAS-ML tem-se, por exemplo, o diagrama de identificação de papeis. Apesar de PASSI e MAS-ML serem baseados na UML, PASSI usa a UML para a representação dos agentes e não a estende, trazendo com isto problemas para a representação de SMAs que MAS-ML visa resolver. Um motivo para isto é que PASSI tem por objetivo modelar agentes para serem implementados em JADE, um *framework* de agentes baseado em Java. Um ponto a ser observado é que PASSI *Toolkit* é integrada ao *Rational Rose* (ferramenta proprietária), enquanto que a MAS-ML é integrada à plataforma *Eclipse* (software livre).

Prometheus Design Tool (PDT). A ferramenta foi idealizada com o objetivo de dar suporte à metodologia *Prometheus* [21], sendo aplicada tanto na fase de análise quanto no desenvolvimento de SMAs. Enquanto que a ferramenta proposta foi derivada a partir da definição do metamodelo da MAS-ML e, no seu estágio atual, é apenas aplicada à fase de projeto. Os principais recursos da ferramenta são: (*i*) interface gráfica juntamente com descritor de texto estruturado; (*ii*) permite a propagação de informações de uma fase do desenvolvimento para outra, sempre que é possível. Por exemplo, se um objetivo está associado a um papel, e este papel é associado a um agente, então o objetivo está associado ao agente; (*iii*) visão hierárquica do processo de definição dos agentes, ou seja, é possível ter a definição dos agentes em diferentes níveis de abstração.

agentTool. agentTool [25] trata-se de um ambiente de modelagem que tem como objetivo auxiliar o projetista de software durante as fases de análise, projeto, e implementação de sistemas de software seguindo o paradigma de SMAs. Semelhante ao *prometeus*, a *agentTool* está associada a uma metodologia de desenvolvimento de SMAs, porém, neste caso, a metodologia utilizada é a O-MaSe (*Organization-based Multiagent Systems Engineering*) [23], [24]. Quanto à sua implementação, a ferramenta trata-se de um *plug-in* da plataforma *Eclipse* e visa, assim como a ferramenta proposta, uma integração entre os ambientes de desenvolvimento orientado a objetos e orientado a agentes. Esta integração é particularmente importante para permitir que SMAs possam ser adotados em projetos, não apenas na academia, mas também na indústria.

Foi observado que cada ferramenta possui um foco diferente o que, de certa forma, pode ser visto como um empecilho na realização de uma comparação mais precisa entre elas. Também, uma questão essencial é que as ferramentas diferem-se na linguagem que elas suportam, assim, as vantagens e desvantagens entre essas linguagens são propagadas para as ferramentas que as implementam. Além disso, avaliando o grau de maturidade sobre a ótica do número de publicações gerado, foi possível constatar que as ferramentas estudadas possuem um considerável grau de maturidade, que pode ser visto de forma positiva para a adoção de SMAs tanto na academia quanto na indústria.

VisualAgent. Trata-se de um ambiente de desenvolvimento de software que tem a finalidade de auxiliar desenvolvedores na especificação, projeto e implementação de SMAs [26], [27]. Similar a ferramenta apresentada neste artigo, a *VisualAgent* é baseada na linguagem de modelagem MAS-ML e disponibiliza a criação de modelos de classes. Ela também propõe uma abordagem dirigida por modelo para o desenvolvimento de SMAs fazendo uso do ASF (*Agent Society Framework*), o que representa um ponto relevante da abordagem. Uma das principais vantagens da MAS-ML tool em relação à *VisualAgent* é a capacidade de realizar verificação do modelo em relação ao metamodelo da MAS-ML. A ausência desta funcionalidade na *VisualAgent* pode comprometer as sucessivas transformações dos modelos presentes na abordagem MDD proposta e levar a um mal entendimento dos modelos criados. Sem falar que um modelo inconsistente pode gerar código com erros.

5. Conclusões e Trabalhos Futuros

Sistemas de software modernos tendem a apresentar características, tais como autonomia e alta interatividade, que resultam na existência de uma lacuna entre o domínio do problema e o domínio da solução, dificultando a modelagem desses sistemas com os paradigmas de desenvolvimento atuais. Acredita-se que os sistemas multiagentes sejam uma abordagem promissora para a redução desta lacuna, através da introdução de abstrações de mais alto nível. Assim, dada a existência de linguagens de modelagem para SMAs, este artigo apresentou um ambiente de modelagem específico de domínio para SMAs permitindo que estes sistemas sejam modelados com o apoio de uma ferramenta. Com isso, realiza-se a disponibilização de recursos de modelagem que permitem os desenvolvedores a trabalhar com conceitos do domínio do problema ao mesmo tempo em que lida com conceitos do domínio da solução.

Partindo do ponto de vista que a modelagem é uma atividade essencial dentro do processo de desenvolvimento de software orientado a objetos, não sendo diferente no desenvolvimento de SMAs, então uma ferramenta que dê suporte à modelagem é de grande importância. Caso contrário, a atividade de modelagem torna-se tediosa e custosa, muitas vezes não sendo realizada. Com isso, foi apresentada ao longo do artigo um ambiente de modelagem para SMAs com foco no diagrama de classes definido na descrição da MAS-ML. Com o ambiente é possível, dentre outras coisas, realizar a modelagem, validar os modelos criados e persistir os modelos.

A avaliação inicial da proposta tem se mostrado eficiente e satisfatória. Essa avaliação consiste na modelagem de SMAs simples, desenvolvidos com trabalhos da disciplina de Projeto de Sistemas de Software da PUC-Rio. Obviamente, alguns melhoramentos necessitam ser realizados como, por exemplo, melhorar a representação gráfica dos construtores, assim como realizar estudos experimentais. Quanto aos

trabalhos futuros, eles se concentrarão em criar os ambientes para os demais diagramas estáticos da MAS-ML, bem como os diagramas dinâmicos que não foram contemplados neste trabalho. Visa-se também realizar transformação dos modelos criados para código específico de plataformas de agentes existentes.

Referências

[1] N. Jennings and M. Wooldrige*, Agent-Oriented Software Engineering*, In Bradshaw, J. (Ed.) Handbook of Agent Technology, AAAI/MIT Press, 2000.

[2] H. Mubarak, *Developing Flexible Software Using Agent-Oriented Software Engineering*, IEEE Software, Sep/Oct, IEEE Computer Society, pp. 12-15, 2008.

[3] J. Lind, *Issues in agent-oriented software engineering*, In: P. Ciancarini e M. Wooldride (Eds.) Agent-Oriented Software Engineering, LNCS 1957, Germany, Springer, p.45-58. 2001.

[4] M. Wooldridge and P. Ciancarini, *Agent-Oriented Software Engineering: the State of the Art*, In: M. Wooldridge and P. Ciancarini (Eds.), Agent-Oriented Software Engineering, LNCS 1957, Berlin: Springer, p. 1-28, 2001.

[5] R. France and B. Rumpe, *Model-Driven Development of Complex Software: A Research Roadmap*, in Future of Software Engineering (FOSE'07) co-located with ICSE'07, Minnesota, EUA, May 2007, pp. 37–54.

[6] V. Silva, A. Garcia, A, Brandão, C. Chavez, C. Lucena, P. Alencar, *Taming Agents and Objects in Software Engineering*. In: A. Garcia, C. Lucena, F. Zamboneli, A. Omicini, J. Castro (Eds.) Software Engineering for Large-Scale Multi-Agent Systems, LNCS 2603, Berlin: Springer, 2003.

[7] V. Silva, and C. Lucena, In: K. Sycara, M. Wooldridge (Eds.), Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, 2004.

[8] V. Silva, R. Choren, C. Lucena, *MAS-ML: a Multi-agent System Modelling Language*, International Journal of Agent-Oriented Software Engineering, Vol. 2, No. 4, 2008.

[9] V. Silva, *Uma Linguagem de Modelagem para Sistemas Multi-Agentes Baseada em um Framework Conceitual para Agentes e Objetos*. Tese de Doutorado, Departamento de Informática, Março 2004.

[10] C. Nunes, U. Kulesza, C. Sant'Anna, I. Nunes, A. Garcia, C. Lucena, *Assessment of the Design Modularity and Stability of Multi-Agent System Product Lines* (to appear), Journal of Universal Computer Science, 2009.

[11] I. Nunes, C. Nunes, U. Kulesza, C. Lucena, *Developing and Evolving a Multi-Agent System Product Line: An Exploratory Study*. In: Michael Luck; Jorge J. Gomez-Sanz. (Org.). Agent-Oriented Software Engineering IX. : Springer-Verlag, 2009, v. 5386, p. 228-242.

[12] Unified Modeling Language: Infrastructure version 2.1, Object Management Group, February 2007.

[13] A. Rao and M. Georgeff, *BDI-agents: from theory to practice*, In: International Conference on Autonomous Agents and Multiagent Systems,1995.

[14] Eclipse Platform, www.eclipse.org, Acessado em 15 de Julho de 2009.

[15] K. Czarnecki, U. Eisenecker, *Generative Programming - Methods, Tools, and Applications*, Adison-Wesley, June 2000.

[16] EMF, www.eclipse.org/modeling/emf/, Acessado em 15 de Julho de 2009.

[17] GEF, www.eclipse.org/gef, Acessado em 15 de Julho de 2009.

[18] GMF, www.eclipse.org/modeling/gmf/, Acessado em 15 de Julho de 2009.

[19] J. Odell, H. Parunak, and B. Bauer, *Extending UML for Agents*. In: G. Wagner, Y. Lesperance (Eds.), Anais of the Agent- Oriented Information Systems Workshop, Austria, p. 3-17, 2000.

[20] V. Silva, C. Lucena, P. Alencar, C. Cowan. *A Model-Based Transformational Approach for Implementing Multi-Agent Systems*, Technical Report CS2004-12, School of Computer Science, University of Waterloo, Canada, 2004.

[21] L. Padgham, J. Thangarajah and M. Winikoff, *Prometheus Design Tool*, in 23th AAAI Conference on Artificial Intelligence, Chicago, EUA, pp.1882-1883 2008.

[22] PASSI ToolKit, http://www.pa.icar.cnr.it/passi/Tools/PTK/ptkIndex.html, acessado acesso em 15 de Julho de 2009.

[23] J. Garcia-Ojeda, S. DeLoach, and Robby, *agentTool Process Editor: Supporting the Design of Tailored Agent-based Processes*, In: 24th Annual ACM Symposium on Applied Computing, Honolulu, Hawaii, USA, pp. 8 - 12, March 2009.

[24] S. DeLoach, *Developing a Multiagent Conference Management System Using the O-MaSE Process Framework*, In: 8th International Workshop on Agent Oriented Software Engineering co-located at SAC'07, Honolulu, Hawaii, May 2007.

[25] agentTool, http://agenttool.cis.ksu.edu/, acessado em 15 de Julho de 2009.

[26] V. Silva, B. Maria, C. Lucena, *A MDE-Baed Apporach for Developing Multi-Agent Systems*, Eletronic Communication of the EASST, v. 3, p. 12-26, 2006.

[27] B. Maria, V. Silva, C. Lucena, *VisualAgent: A Software Development Environment for Multi-Agent Systems*, In: Brazilian Symposium on Software Engineering (SBES2005), 2005, Uberlândia, Tool Track, 2006.

[28] Eclipse Platform, http://www.eclipse.org/, Acessado em 28 de Agosto de 2009.