Usando MDA no Desenvolvimento de Sistemas Multi-Agentes

Beatriz Alves De Maria Viviane Torres da Silva Carlos José Pereira de Lucena

Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro,
Rio de Janeiro, RJ, Brasil
{biamar,viviane,lucena}@inf.puc-rio.br

PUC-RioInf.MCC39/04 Novembro, 2004

Abstract: A great number of methodologies, frameworks and languages are being proposed to support the development of multi-agents systems (MAS). In this paper, we propose the use of MDA in the development of this kind of systems. MDA specifies a structured software development process in modeling stages. In PIM stage, where platform independent models are specified, we propose the use of MAS-ML modeling language for MAS. In PSM stage, where platform specific models are specified, we propose the use of UML modeling language. The MAS-ML models defined on PIM stage are transformed in UML models at PSM stage, based on a framework to implement MAS using object orientation. In the last development stage, the application code is generated from UML models. In this work, we will detail PIM and PSM stages on the development process of MAS and the transformations to generate source code.

Keywords: multi-agent systems, agents, modeling language, XMI, MAS-ML, MDA

Resumo: Um grande número de metodologias, frameworks e linguagens estão sendo propostas para dar apoio a construção de sistemas multi-agentes (SMA). Neste artigo, propomos o uso de MDA no processo de desenvolvimento destes sistemas. MDA especifica um processo de desenvolvimento de software estruturado em etapas de modelagem. Na etapa PIM, onde são especificados modelos independentes de plataforma, propomos a utilização da linguagem de modelagem MAS-ML para SMA. Na etapa PSM, onde são definidos modelos específicos de plataforma, propomos utilizar a linguagem de modelagem UML. Os modelos MAS-ML definidos na etapa PIM são transformados em modelos UML na etapa PSM com base em um framework para implementação de SMA utilizando orientação a objetos. Na ultima etapa do desenvolvimento, o código da aplicação é gerado a partir dos modelos UML. Neste trabalho, iremos detalhar as etapas PIM e PSM do processo de desenvolvimento de SMA e as transformações necessárias para a geração de código.

Palavras-chave: sistemas multi-agentes, agentes, linguagem de modelagem, XMI, MAS-ML, MDA.

1. Introdução

Nos últimos anos, o desenvolvimento de sistemas multi-agentes (SMA) cresceu rapidamente. Aplicações de diferentes, domínios, como bibliotecas digitais, ex. [1], mercados virtuais, ex.: [13,18] e sistemas de informação em geral, ex.[4,5], foram desenvolvidas utilizando-se a abordagem de sistemas multi-agentes. Devido ao crescente uso destes sistemas, surge a necessidade de se ter ferramentas que facilitem a modelagem e a construção de sistemas baseados na tecnologia de agentes de software. Linguagens de modelagem, linguagens de programação, metodologias e várias outras técnicas de modelagem e implementação voltadas para SMA foram propostas como o objetivo de abordar as abstrações existentes nestes sistemas e lidar com as características intrínsecas destas abstrações.

Como em qualquer outro processo de desenvolvimento de software, o desenvolvimento de SMA também está centrado no levantamento dos requisitos da aplicação, na análise do domínio, no design, na implementação e nos testes do sistema. Cada etapa do desenvolvimento gera um conjunto de artefatos que devem ser refinados em artefatos da etapa subseqüente. Várias metodologias [6, 43, 8] propostas na literatura de SMA descrevem as diferentes etapas do desenvolvimento destes sistemas e algumas abordam ainda o refinamento dos artefatos de uma etapa para outra.

Neste artigo propomos o uso da arquitetura Model Driven Architecture (MDA) [25], proposto pela OMG, no processo de desenvolvimento de SMA. MDA é uma arquitetura para desenvolvimento de software, que dá suporte a todo o ciclo de desenvolvimento, ou seja, engloba todas as fases do desenvolvimento de um sistema. Sua principal característica é a importância dada aos modelos gerados nas diferentes etapas do processo de desenvolvimento e no rastreamento dos modelos de uma etapa para a etapa seguinte.

A arquitetura MDA é composta por quatro etapas principais, onde ao final de cada etapa tem-se como resultado modelos formais. A passagem de uma etapa para a etapa seguinte é realizada através de transformações de um modelo para outro. Estas transformações possibilitam a rastreabilidade dos modelos. Existem três tipos básicos de modelos definidos no processo de desenvolvimento de MDA: modelos independentes de computação (CIM), modelos independentes de plataforma (PIM) e modelos específicos de plataforma (PSM). Modelos CIM correspondem aos modelos conceituais da aplicação que não são dependentes de características computacionais. Tais modelos são gerados na primeira etapa do processo de desenvolvimento do software. Modelos PIM, gerados na segunda etapa do processo de desenvolvimento, são modelos de alto nível de abstração independentes de plataforma. Já os PSM, correspondentes a terceira etapa, são modelos gerados a partir dos modelos PIM com a inclusão de detalhes específicos de um tipo de plataforma de implementação. Na quarta e última etapa do processo de desenvolvimento, os modelos PSM são transformados gerando-se o código da aplicação. Além da especificação dos modelos, MDA define ainda que um conjunto de transformações consecutivas deve ser aplicado aos modelos para permitir transformar modelos de mais alto nível de abstração em código.

O objetivo da proposta apresentada é oferecer um ambiente que permita o desenvolvimento de SMA utilizando a abordagem MDA. Neste trabalho, utilizamos a linguagem de modelagem MAS-ML[34] para modelar SMA em um alto nível de abstração. A modelagem de um SMA gerada utilizando-se MAS-ML corresponderá aos modelos PIM propostos em MDA. Tais modelos são independentes da plataforma de implementação pois MAS-ML não restringe ou especifica a plataforma onde os modelos devem ser implementados. MAS-ML é uma linguagem de modelagem para SMA que estende UML incluindo abstrações relacionadas a agentes. Utilizando-se MAS-ML é possível modelar os aspectos estruturais (as abstrações, suas propriedades e relacionamentos entre as abstrações) e os aspectos dinâmicos (as interações entre as abstrações e as execuções internas das mesmas) de SMA.

De acordo com MDA, após a geração dos modelos PIM estes modelos são transformados em modelos PSM. Em nossa proposta, os modelos MAS-ML são transformados em modelos UML [28] utilizando-se o framework ASF (Agen Society Framework)[36]. O framework ASF possibilita a implementação de SMA utilizando a linguagem de programa orientada a objetos Java. Os modelos

UML definidos nesta etapa incluem características dependentes do framework e da linguagem de programação onde estes serão implementados. Por fim, na última etapa do processo de desenvolvimento, os modelos UML específicos de plataforma são transformados em código. Utilizamos XMI [27] para representar os modelos UML e possibilitar a geração de código da aplicação a partir destes modelos.

Cada etapa definida anteriormente está diretamente relacionada às etapas presentes em MDA. A transformação de MAS-ML em UML está relacionada a transformação em MDA do nível independente de plataforma para o nível específico da plataforma, correspondendo assim, a transformação dos modelos PIM em PSM. Já a transformação dos modelos UML em código está relacionada à última transformação existente em MDA, que é responsável por gerar código a partir de modelos PSM. A representação dos modelos CIM como também a geração dos modelos PIM através destes modelos não são tratadas neste artigo.

A utilização da abordagem de MDA no processo de desenvolvimento proposto traz uma série de benefícios para o desenvolvimento de SMA. Devido ao design de modelos independentes de plataforma através do uso de MAS-ML, os modelos PIM de descrição do problema são modelos portáveis que podem ser reutilizados para geração de diferentes modelos computacionais utilizando-se diferentes plataformas de implementação. Em nossa proposta utilizamos o framework ASF para geração de modelos PSM a partir de modelos MAS-ML porém a plataforma Jadex[29,30], por exemplo, também poderia ter sido utilizada. Outra vantagem no uso de MDA é a geração de diferentes modelos cujos focos estão em diferentes pontos de vista do sistema. Os modelos MAS-ML de alto nível de abstração focam na descrição do problema descrevendo o sistema no nível de abstração de agentes. Já os modelos UML de mais baixo nível de abstração focam na descrição da solução do problema incluindo detalhes de implementação. Durante a manutenção dos modelos MAS-ML o designer não se preocupa com os detalhes de implementação e se concentra na definição do problema. Se as alterações a serem feitas estiverem relacionadas restritamente à plataforma de implementação os modelos UML serão modificados porém os modelos MAS-ML não sofrerão alterações. O uso de MDA possibilita a baixo acoplamento entre os diferentes modelos gerados durante o processo de desenvolvimento.

A Seção 2 deste documento apresenta de forma sucinta a principais características da linguagem de modelagem MAS-ML, do framework ASF e da arquitetura MDA. Na Seção 3, será apresentada o processo de desenvolvimento de SMA utilizando MDA. Na Seção 4, é apresentada uma ferramenta que visa facilitar o processo de desenvolvimento de software de acordo com a abordagem descrita na seção anterior. A Seção 5 apresenta os trabalhos relacionados e por fim, na Seção 6, são apresentados a conclusão e os trabalhos futuros.

2. Background

2.1 MAS-ML

A linguagem MAS-ML [34] é uma linguagem de modelagem específica para sistemas multi-agentes. Esta linguagem é uma extensão de UML utilizando os conceitos apresentados no framework conceitual TAO [33]. O framework conceitual TAO define as abstrações comumente encontradas em SMA assim como suas propriedades, seus relacionamos e a maneira como as entidades executam e interagem. Sendo assim, utilizando-se MAS-ML é possível modelar não somente objetos (provenientes da extensão de UML) mas também agentes, ambientes, organizações e papéis; entidades definidas no TAO e comumente encontradas em SMA. Fig. 1 ilustra todas as entidades que podem ser modeladas em MAS-ML e todas os relacionamentos definidos entre estas entidades.

A fim de modelar os aspectos estruturais e os aspectos dinâmicos das entidades relacionadas a agentes definidas no TAO, MAS-ML define três diagramas estruturais e um diagrama dinâmico. Os diagramas estruturais possibilitam a modelagem de todas as classes das entidades, suas propriedades e seus relacionamentos. Utilizando-se diagramas dinâmicos é possível modelar o comportamento das instâncias das entidades enfocando a execução interna das mesmas e as interações entre elas.

Os diagramas estruturais propostos por MAS-ML são o diagrama de classes de UML estendido, o diagrama de organizações e o diagrama de papéis. O diagrama de classes representa agentes, organizações e ambientes e os relacionamentos entre estas entidades e classes. O diagrama de organizações tem por objetivo modelar as organizações existentes no sistema identificando seus ambientes, os papéis que as organizações definem e os elementos (objetos, agentes e sub-organizações) que desempenham esses papéis. Já o diagrama de papéis tem como finalidade esclarecer as relações presentes entre todos os papéis. Por fim, o dinâmico de MAS-ML é o diagrama de seqüência de UML estendido para modelar a interação entre agentes, organizações e ambientes e a execução interna destas entidades.

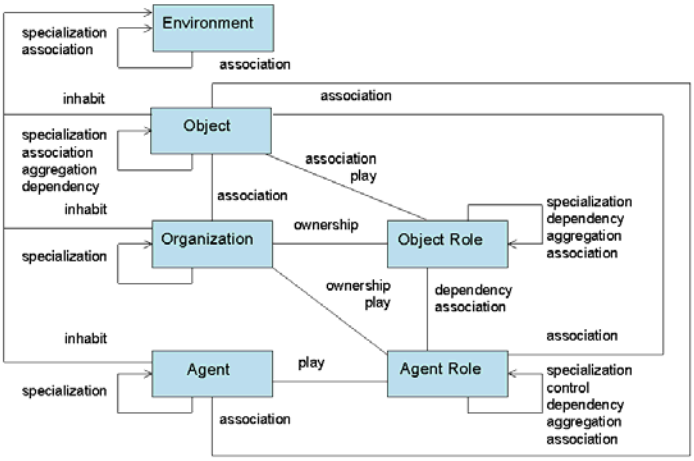


Fig. 1. Abstrações e relacionamentos propostos por TAO

2.2 ASF Framework

O framework ASF [36] possibilita a implementação de SMA usando-se a tecnologia orientada a objetos (OO). Assim como MAS-ML, o framework ASF também é baseado no framework conceitual TAO. Com o uso do framework é possível implementar os agentes, papéis, organizações e ambientes modelados em MAS-ML utilizando-se orientação a objetos. Pelo fato destas entidades modeladas em MAS-ML não serem abstrações existentes em OO, o framework propõe o mapeamento destas entidades em um conjunto de módulos OO.

Cada módulo que compõem o framework representa um tipo de entidade existente em SMA. O conjunto de classes e relacionamentos definidos em um módulo possibilita a implementação dos aspectos estruturais de uma entidade (suas propriedades e relacionamentos com outras entidades) e dos aspectos dinâmicos da mesma (seu comportamento). A Fig. 2 apresenta todas as classes dos cinco módulos que estão presentes no framework. Os módulos que representam agentes (delimitado através de um retângulo contínuo), organizações (definido pelo enlace pontilhado) e papéis de agentes (marcado pelo retângulo tracejado) são os módulos mais complexos. A complexidade se dá pela necessidade da criação de classes para representar as propriedades destes tipos de entidades. Os módulos para representação de ambientes (círculo tracejado) e objetos (círculo contínuo) são simplesmente representados por uma classe devido às propriedades destas entidades poderem ser representadas diretamente como atributos e métodos. Para maiores detalhes sobre a definição dos módulos se referir a [36].

Para implementar um tipo de ambiente ou um tipo de papel desempenhado por um objeto utilizando-se o framework, é bastante simples. Basta estender as classes *Environment* e *ObjectRole*, respectivamente. Já a implementação de um tipo de agente se dá através da criação de uma classe que estenda a classe abstrata *Agent* definida no framework e a implementação do construtor da classe para criar os objetivos, crenças, planos e ações de acordo com as propriedades do agente. Os objetivos e crenças do agente são criados instanciando-se objetos das classes *Goal* and *Belief*. A representação dos planos e das ações dos agentes ocorre através da criação de classes concretas que estendam as classes abstratas *Plan* and *Action* e que implementem o comportamento definido planos e ações.

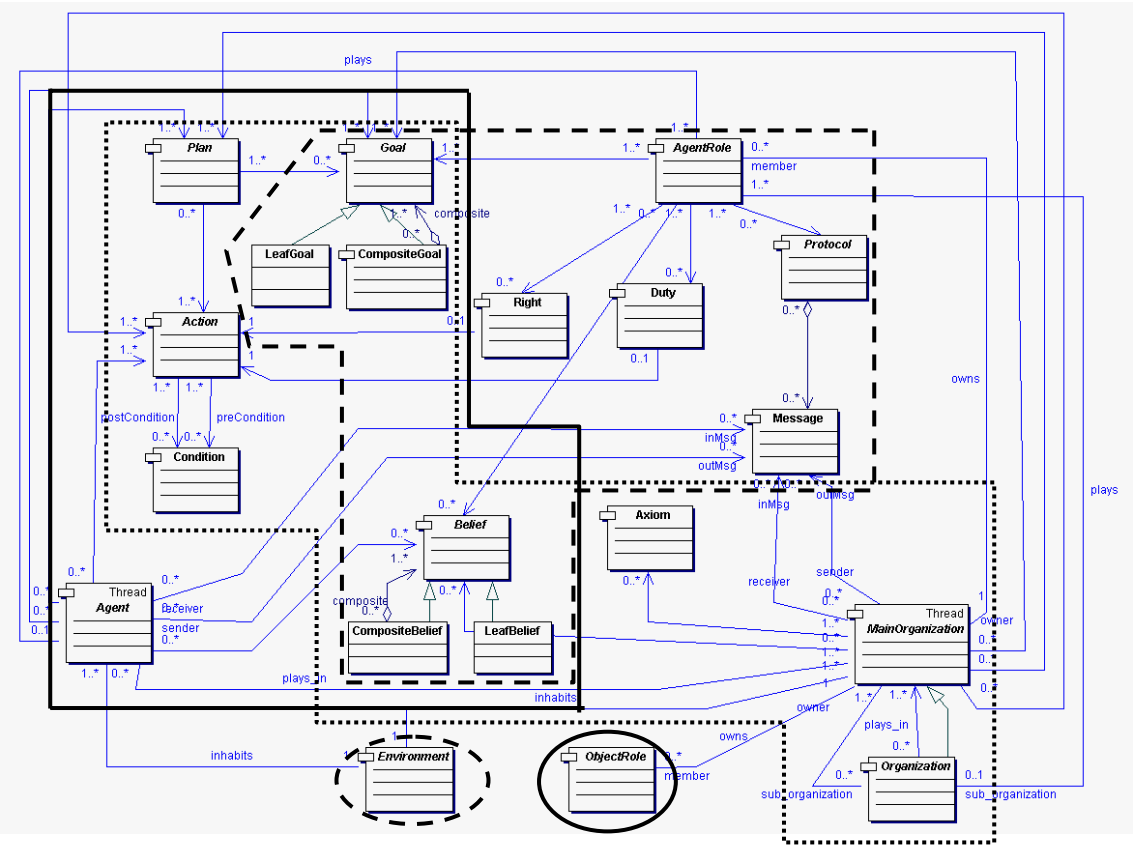


Fig. 2. Classes existentes no ASF Framework

A implementação de um tipo de organização é bastante parecida com a implementação de um tipo de agente a menos de duas diferenças. A classe que representará o tipo de organização poderá estender da classe *MainOrganization* ou da classe *Organization*, de acordo com sua definição. Além disso, será necessário criar instâncias da classe *Axiom* no construtor da organização para representar as regras (ou leis) da organização.

Um tipo de papel de agente é implementado através da criação de uma classe que estenda a classe *AgentRole*. A definição dos direitos e deveres do papel se dará pela instanciação das classes *Right* and *Duty*. Os protocolos são definidos através da criação de classes concretas que implementem o comportamento definido nos protocolos e que estendem a classe *Protocol*.

2.3 Model Driven Architecture (MDA)

Model Driven Architecture[25] é uma arquitetura conceitual para desenvolvimento de software definida pela OMG. Esta arquitetura visa separar decisões orientadas ao negócio de decisões de plataforma permitindo assim maior flexibilidade durante as fases de especificação e de desenvolvimento de sistemas. A idéia central de MDA é a criação de diferentes modelos com diferentes níveis de abstração e a ligação destes modelos através de refinamentos e transformações. Alguns desses modelos existirão independente da plataforma, outros serão específicos para uma plataforma.

Os três modelos a seguir compõem os principais elementos de MDA:

- CIM (*Computational Independent Model*): descreve o domínio da aplicação, sem ater aos detalhes da estrutura do sistema. É importante para a ajuda no entendimento do problema, como também é uma fonte de vocabulário a ser usado nos demais modelos.

- PIM (*Plataform Independent Model*): descreve o sistema, porém não apresenta os detalhes da tecnologia que será usada na implementação. O PIM oferece especificações formais da estrutura e funcionalidade do sistema, abstraindo-se de detalhes técnicos.
- PSM (*Plataform Specific Model*): combina a especificação do modelo PIM com detalhes específicos de uma determinada plataforma.

A transformação de modelos é a chave principal desta abordagem e consiste no processo de converter um modelo em outro modelo do mesmo sistema[25]. A idéia principal é construir modelos no seu mais alto nível de abstração e transformá-los em modelos com baixa abstração, de forma automática ou semi-automática, facilitando e tornando mais rápido o processo de desenvolvimento. A Fig. 3 representa as etapas definidas em MDA e as transformações entre as etapas.

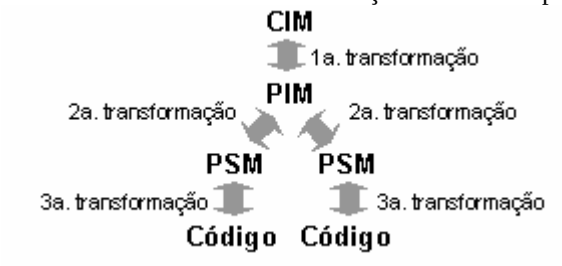


Fig. 3. O relacionamento entre CIM, PIM, PSM e Código

3. Sistemas Multi-Agentes e MDA

O processo de desenvolvimento de sistemas complexos e de larga escala, como os SMA, envolve a construção de diferentes modelos baseados em uma série de requerimentos. A transformação de uma especificação em modelos e de modelos em código é feita normalmente de maneira desorganizada e não é facilmente adaptada a mudanças de tecnologia.

Como a tecnologia de agentes está ganhando aceitação e está sendo amplamente utilizada, surge uma crescente necessidade de se criar métodos de fácil entendimento para o desenvolvimento de sistemas multi-agentes. O processo de desenvolvimento proposto nesta seção tem como objetivo facilitar o desenvolvimento de aplicações orientadas a agentes. Este processo dá suporte ao desenvolvimento de SMA seguindo a iniciativa MDA. Nele é possível criar modelos independentes de plataforma (PIM), transformá-los em modelos dependentes de plataforma (PSM) e gerar código a partir destes modelos.

É importante ressaltar que o processo de desenvolvimento de SMA baseado em MDA pode acontecer de diferentes formas, conforme apresentado na Fig. 4. A partir do modelo de descrição do sistema especificado na etapa CIM é possível gerar diferentes modelos independentes de plataforma (PIM) utilizando-se diferentes abordagens. Na Fig. 4 ilustra-se o uso de duas abordagens diferentes. O modelo CIM aparece transformado em modelos PIM descritos usando-se a linguagem de modelagem MAS-ML e utilizando-se a linguagem de modelagem AUML[2]. Após a geração dos modelos PIM, usando-se por exemplo MAS-ML, modelos PSM específicos de plataforma devem ser gerados. Na transformação de modelos MAS-ML diferentes plataformas como framework ASF, a arquitetura Jadex [29,30] ou RETSINA[37,38] poderiam ser utilizadas. Ambos são exemplos de plataformas desenvolvidas para a implementação de sistemas multi-agentes. Por último, temos a transformação dos modelos PSM em código de acordo com a linguagem de implementação da plataforma.

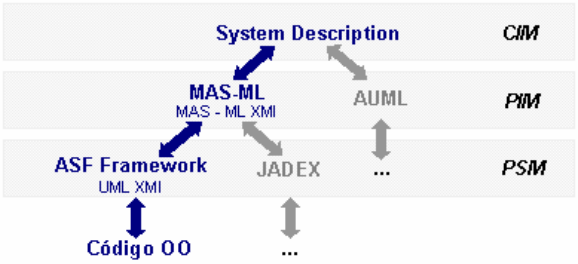


Fig. 4. Processo de desenvolvimento Proposto

O processo de desenvolvimento proposto neste documento consiste em apenas três etapas, baseadas nas etapas do MDA. O processo não aborda a etapa CIM e a transformação dos modelos CIM para PIM. A primeira etapa proposta em nossa abordagem, que corresponde à etapa PIM de MDA, define a modelagem do SMA através do uso da linguagem MAS-ML. Os modelos MAS-ML são modelos independentes de plataforma de implementação. MAS-ML não restringe ou especifica a plataforma onde os modelos devem ser implementados. Diferentes plataformas orientadas a objetos podem ser utilizadas para implementar sistemas modelados utilizando-se MAS-ML.

A segunda etapa do processo proposto está relacionada à forma como o sistema será implementado. Os modelos gerados nesta etapa são modelos UML dependentes de plataforma (modelos PSM de acordo com MDA). Tais modelos são dependentes de plataforma pois propomos a utilização do framework ASF para a implementação de SMA. Sendo assim, os modelos UML acrescentam detalhes da implementação do sistema definidas no framework.

A terceira e última etapa do processo de desenvolvimento corresponde a geração de código a partir dos modelos UML da etapa anterior. O código gerado com base na nossa proposta é descrito utilizando-se a linguagem orientada a objetos Java. No decorrer desta seção serão explicados com mais detalhes cada etapa e o modo como é feita a passagem de uma etapa para outra.

O estudo de caso escolhido para a exemplificação do processo proposto é uma aplicação de comércio eletrônico. Tais aplicações são consideradas *brenchmarks* no domínio de sistemas multi-agentes [13,19,44]. Esta aplicação é constituída de uma loja virtual onde usuários cadastrados podem comprar itens de diversos tipos existentes na loja. Agentes representantes do usuário negociam com agentes representantes da loja os itens que desejam comprar. O gerenciamento das vendas e controle de estoque é feito pela própria loja representada no sistema por uma organização.

3.1 Modelo CIM

Os modelos desta etapa devem ser independentes das abstrações computacionais utilizadas na solução do problema. O problema é definido sem detalhar as abstrações escolhidas para solucionar o problema, isto é, sem detalhar se serão abstrações orientadas a agentes ou a objetos, por exemplo. Durante a definição dos modelos CIM, o sistema pode ser modelado utilizando-se cenários, features e baseando-se em ontologias específicas de domínio. Os cenários, os modelos de features ou as instâncias da ontologia irão definir o problema em questão.

Uma vez definido o problema, o modelo gerado nesta etapa deve ser transformado em modelos PIM. É durante o processo de transformação dos modelos CIM para PIM que as abstrações usadas na solução do problema são escolhidas. Não é escopo deste trabalho demonstrar a transformação de modelos CIM para PIM no contexto de SMA. Como trabalho relacionado a esta transformação encontra-se o trabalho publicado em [24]. Nesta abordagem os autores mapeiam modelos de features que definem um problema para modelos UML baseando-se em um framework orientado a objetos.

3.2 Modelo PIM

Os diagrama de MAS-ML abordados nesta proposta são os diagramas estruturais (diagrama de classes de UML estendido, diagrama de organização e diagrama de papéis). Não faz parte do escopo deste

artigo mostrar as transformações entre modelos e a geração de código a partir dos diagramas dinâmicos de MAS-ML (diagrama de seqüência UML estendido).

A escolha de MAS-ML como linguagem de modelagem para esta etapa foi influenciada por três principais fatores. Primeiramente, esta linguagem não é uma linguagem dependente de plataforma de implementação. Devido estender UML, MAS-ML utiliza conceitos de orientação a objetos mas não restringe a implementação dos seus modelos a uma linguagem de programação específica. Outro ponto importante, MAS-ML, assim como outras linguagens que estendem UML como AUML, é compatível com o framework MOF [26], o que facilita o processo de desenvolvimento proposto pois utilizamos XMI [27] durante as transformações entre modelos. Por fim, MAS-ML aborda abstrações como organizações, ambientes e papéis de agentes que não são abordadas adequadamente em outras linguagens de modelagem como AUML. Sem o uso destas abstrações não é possível modelar, por exemplo, agentes desempenhando diferentes papéis em diferentes organizações.

Para exemplificar a modelagem MAS-ML de um SMA, representamos na Fig. 5 o diagrama de organização da aplicação de comércio eletrônico detalhada anteriormente. Este diagrama juntamente com o detalhamento da organização nele representada e ilustrada na Fig. 6 serão utilizados para exemplificar a transformação dos modelos MAS-ML em modelos UML. Tal transformação é apresentada na Seção 3.3. Devido a restrição de espaço não será possível demonstrar as transformações de todos os diagramas e todas as entidades definidas em MAS-ML.

O diagrama da Fig. 5 descreve a organização que representa a loja virtual (*General_Store*), os agentes do sistema, os papéis que eles podem desempenhar, os objetos e os papéis dos objetos. Na loja em questão podem existir dois tipos de agentes, os agentes que representam os usuários (*User_Agent*) e os que representam os lojistas (*Store_Agent*). Os usuários desempenham o papel de comprador (*Buyer*) quando desejam adquirir um item da loja. Os lojistas desempenham o papel de vendedor (*Seller*) no momento em que interagem com o comprador para vender um item.

Os compradores e os vendedores possuem diferentes interpretações para os itens que estão negociando. Um item é objeto de desejo (*Desire*) do comprador e é uma oferta (*Offer*) para o vendedor. Estas interpretações diferentes caracterizam os diferentes papéis dos objetos associados ao item. É importante salientar que o diagrama da Fig. 5 está representado de forma simplificada, visto que estão suprimidos os compartimentos central e inferior de cada elemento onde são detalhadas as propriedades estáticas e dinâmicas, respectivamente.

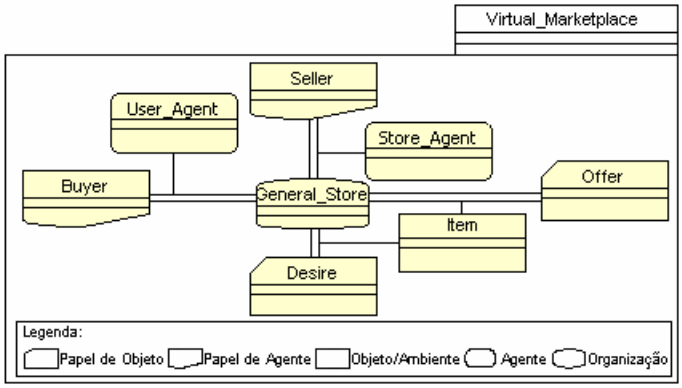


Fig. 5. Diagrama de Organização da Loja Principal

Na Fig. 6 representamos de forma mais detalhada a loja virtual *General_Store*. Nesta figura estão representados um objetivo, uma crença, um axioma, duas ações e um plano. As principais metas (*goals*) da organização são o gerenciamento dos vendedores (*management_of_sellers*), das ordens de pedido e dos lucros. Para poder gerenciar os vendedores, esta organização necessita conhecer todos os vendedores que estão negociando. Este conhecimento é representado por uma crença conforme ilustrado na Fig. 6. Com o objetivo de garantir que todos os vendedores ao efetuar uma venda informem a loja sobre a venda foi definido o axioma *send_information_about_sale*.

Para alcançar suas metas, a loja tem os seguintes planos: criar vendedores para negociar com os compradores (*creating_sellers*), atualizar o ambiente com as informações dos itens que foram vendidos e calcular os lucros que resultaram das vendas. Na Fig. 6 estão as ações que compõem o plano *creating_sellers* e o objetivo ao qual o plano está relacionado.

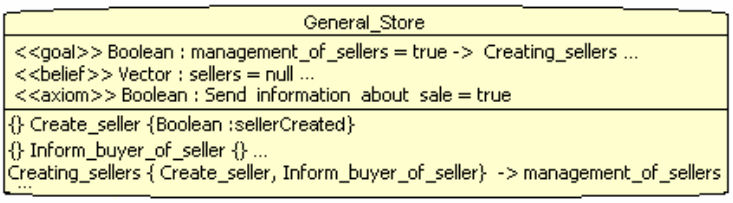


Fig. 6. A loja virtual (*General_Store*) detalhada em MAS-ML

3.3 Transformação do PIM para o PSM

A transformação dos modelos MAS-ML em modelos UML ocorre em duas etapas. A primeira etapa consiste na descrição textual dos modelos da aplicação graficamente representados utilizando-se MAS-ML. Os modelos gráficos são descritos utilizando-se o formato padrão XMI [27]. XMI é um padrão para dar suporte à modelagem, gerenciamento e intercâmbio de meta-dados permitindo a interoperabilidade entre as ferramentas de software que produzem e compartilham meta-dados.

Para a representação de modelos MAS-ML em XMI foi criado o MAS-ML DTD que é uma extensão do UML DTD [27] proposto pela OMG. A extensão do UML DTD foi definida de acordo com a extensão do meta-modelo de UML proposta por MAS-ML. O XMI da aplicação gerado a partir do MAS-ML DTD é aqui chamado de MAS-ML XMI. Utilizando-se o MAS-ML XMI é possível descrever de forma textual os agentes, organizações, ambientes, papéis e objetos da aplicação.

A segunda etapa da transformação compreende o refinamento do MAS-ML XMI para UML XMI de acordo com o framework ASF. É este refinamento que representa a transformação dos modelos PIM para PSM. Os modelos PIM descritos em MAS-ML XMI são transformados em modelos PSM descritos em UML XMI. O UML XMI irá representar a aplicação modelada em MAS-ML e implementada através da instanciação do framework ASF. O UML XMI contem os detalhes da implementação da aplicação.

Primeira etapa: Criação do MAS-ML XMI

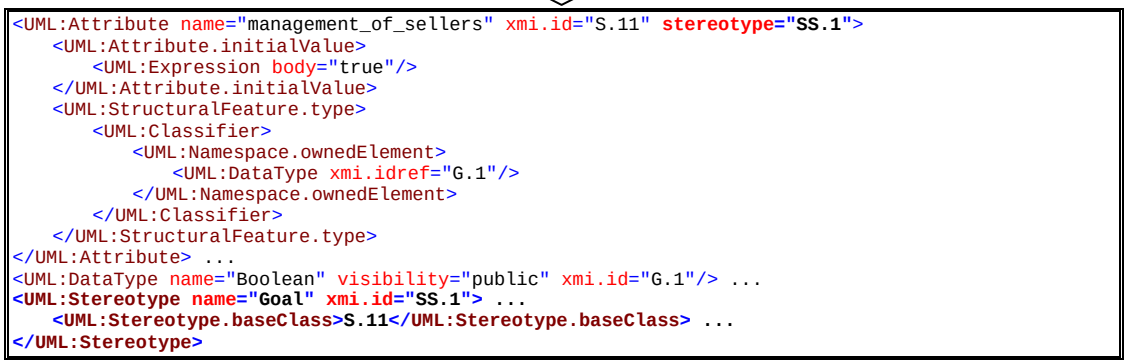
Nesta etapa todos os diagramas MAS-ML que modelam o SMA são descritos em XMI. A fim de exemplificar a representação dos diagramas em XMI, iremos descrever o diagrama de organizações e detalhar a organização representada nas Fig. 5 e Fig. 6. A figura a seguir apresenta em XMI o diagrama de organização representado na Fig. 5. Neste XMI são omitidos alguns detalhes visando não estender demais este documento. Nas Fig. 7, Fig. 8, Fig. 9 e Fig. 9 detalharemos a organização *General_Store* modelada na Fig. 6.

```
<XMI xmi.version="1.1" xmlns:MASML="org/MASML1.0" xmlns:UML="org/UML1.5">
  <XMI.content>
    <UML:Model name="MarketPlace" xmi.id="S.1">
      <UML:Namespace.ownedElement>
        <UML:Class name="Item" namespace="S.1" xmi.id="S.2"/>
        <MASML:AgentClass name="Store_Agent" namespace="S.1" xmi.id="S.3"/>
        <MASML:AgentClass name="User_Agent" namespace="S.1" xmi.id="S.4"/>
        <MASML:OrganizationClass name="General_Store" namespace="S.1" xmi.id="S.5">...
        </MASML:OrganizationClass>
        <MASML:ObjectRoleClass name="Offer" namespace="S.1" xmi.id="S.6"/>
        <MASML:ObjectRoleClass name="Desire" namespace="S.1" xmi.id="S.7"/>
        <MASML:AgentRoleClass name="Buyer" namespace="S.1" xmi.id="S.8"/>
        <MASML:AgentRoleClass name="Seller" namespace="S.1" xmi.id="S.9"/>
        <MASML:PassiveEnvironmentClass name="VirtualMarketPlace" namespace="S.1" xmi.id="S.10"/>
        <MASML:Play member="S.3" played="S.9" player="S.5" xmi.id="L.1"/>
        <MASML:Play member="S.4" played="S.8" player="S.5" xmi.id="L.2"/>
        <MASML:Play member="S.2" played="S.7" player="S.5" xmi.id="L.3"/>
        <MASML:Play member="S.2" played="S.6" player="S.5" xmi.id="L.4"/>
        <MASML:Inhabit citzen="S.4" environment="S.10" xmi.id="L.5"/>
        <MASML:Inhabit citzen="S.2" environment="S.10" xmi.id="L.6"/>
        <MASML:Inhabit citzen="S.5" environment="S.10" xmi.id="L.7"/>
        <MASML:Inhabit citzen="S.3" environment="S.10" xmi.id="L.8"/>
          ...
      </UML:Namespace.ownedElement>
    </UML:Model>
  </XMI.content>
</XMI>
```

Fig. 7. Representação em MAS-ML XMI do diagrama de organização

O código XMI que representa um objetivo, uma crença, um plano, uma ação e um axioma é ilustrado a seguir. A Fig. 8 descreve um objetivo da organização *General_Store*. Como definido em MAS-ML, a representação de um objetivo é feita através do uso de um estereótipo associado a um atributo. A ligação entre o objetivo em si e o estereótipo que define o objetivo é feita através de indicadores de referência no documento conforme ilustrado na Fig. 8.

<<goal>> Boolean : management_of_sellers = true -> Creating_sellers



```
<UML:Attribute name="management_of_sellers" xmi.id="S.11" stereotype="SS.1">
    <UML:Attribute.initialValue>
        <UML:Expression body="true"/>
    </UML:Attribute.initialValue>
    <UML:StructuralFeature.type>
        <UML:Classifier>
            <UML:Namespace.ownedElement>
                <UML:DataType xmi.idref="G.1"/>
            </UML:Namespace.ownedElement>
        </UML:Classifier>
    </UML:StructuralFeature.type>
</UML:Attribute> ...
<UML:DataType name="Boolean" visibility="public" xmi.id="G.1"/> ...
<UML:Stereotype name="Goal" xmi.id="SS.1"> ...
    <UML:Stereotype.baseClass>S.11</UML:Stereotype.baseClass> ...
</UML:Stereotype>
```

Fig. 8. Representação em MAS-ML XMI da propriedade Goal

Na Fig. 9 estão representadas em MAS-ML XMI a crença e o axioma da organização modelada na Fig. 6. Assim como uma meta, uma crença e um axioma também são tratados como um estereótipo de atributo.

<<belief>> Vector : sellers = null
<<axiom>> Boolean : Send_information_about_sale = true

```
<!-- Belief -->
<UML:Attribute name="sellers" xmi.id="S.12" stereotype="SS.3">
    <UML:Attribute.initialValue>
        <UML:Expression body="null"/>
    </UML:Attribute.initialValue>
    <UML:StructuralFeature.type>
        <UML:Classifier>
            <UML:Namespace.ownedElement>
                <UML:DataType xmi.idref="G.2"/>
            </UML:Namespace.ownedElement>
        </UML:Classifier>
    </UML:StructuralFeature.type>
</UML:Attribute> ...
<UML:DataType name="Vector" visibility="public" xmi.id="G.2"/> ...
<UML:Stereotype name="Belief" xmi.id="SS.3"> ...
    <UML:Stereotype.baseClass>S.12</UML:Stereotype.baseClass> ...
</UML:Stereotype>

<!-- Axiom -->
<UML:Attribute name="Send_information_about_sale" xmi.id="S.13" stereotype="SS.5">
    <UML:Attribute.initialValue>
        <UML:Expression body="true"/>
    </UML:Attribute.initialValue>
    <UML:StructuralFeature.type>
        <UML:Classifier>
            <UML:Namespace.ownedElement>
                <UML:DataType xmi.idref="G.1"/>
            </UML:Namespace.ownedElement>
        </UML:Classifier>
    </UML:StructuralFeature.type>
</UML:Attribute> ...
<UML:DataType name="Boolean" visibility="public" xmi.id="G.1"/> ...
<UML:Stereotype name="Axiom" xmi.id="SS.5">
    <UML:Stereotype.baseClass>S.12</UML:Stereotype.baseClass>
</UML:Stereotype>
```

Fig. 9. Representação em MAS-ML XMI das propriedades Belief e Axiom

A Fig. 10 apresenta o código em MAS-ML XMI utilizado para representar as ações e os planos da organização da Fig. 6. Para representar um plano em MAS-ML XMI é necessário indicar quais ações estão relacionadas a este plano e a meta que o plano tenta atingir, caso exista. O plano exemplificado abaixo está relacionado à meta apresentada na Fig. 8. Esta associação pode ser verificada através do identificador de propriedade marcado com um circulo na figura.

```
{} Create_seller {sellerCreated}
{} Inform_buyer_of_seller {}
Creating_sellers { Create_seller, Inform_buyer_of_seller} -> management_of_sellers
```

```
<MASML:AgentAction name="Create_seller" xmi.id="S.14">
   <MASML:AgentAction.posCondition>
      <Condition default="" name="sellerCreated" type="G.1">
   </MASML:AgentAction.posCondition>
</MASML:AgentAction>
<MASML:AgentAction name="Inform_buyer_of_seller" xmi.id="S.15"/>
<MASML:AgentPlan name="Creating_sellers" visibility="public" xmi.id="S.16">
    <MASML:AgentPlan.actions>
        <Action xmi.idref="S.14"/>
        <Action xmi.idref="S.15"/>
    </MASML:AgentPlan.actions>
    <MASML:AgentPlan.goal stereotype="SS.1" xmi.idref="S.11"/>
</MASML:AgentPlan>
<UML:DataType name="Boolean" visibility="public" xmi.id="G.1"/>
```

Goal

Fig. 10. Representação em MAS-ML XMI das propriedades Action e Plan

Segunda etapa: Criação do UML XMI

Nesta etapa o MAS-ML XMI gerado na etapa anterior é convertido para UML XMI através da instanciação do framework ASF. Esta transformação utiliza o arquivo UML XMI que contem as especificações do framework. Este arquivo descreve todas as classes e os relacionamentos entre as classes que são definidas no framework. Durante a transformação, o arquivo UML XMI do framework é estendido com os detalhes da aplicação que estão descritos no MAS-ML XMI. A extensão do arquivo caracteriza a instanciação do framework e a geração da aplicação.

Para exemplificar o mapeamento realizado, utilizaremos o MAS-ML XMI que descreve a organização *General_Store* modelada na Fig. 6 e descrita textualmente nas Fig. 7, Fig. 8, Fig. 9 e Fig. 10. Conforme descrito na Seção 2.2, a instanciação do framework para implementar uma organização consiste de três etapas: (i) criação da classe concreta que irá representar a organização e irá estender a classe abstrata *MainOrganization* ou *Organization*, (ii) implementação do construtor desta classe de acordo com as propriedades modeladas e (iii) criação de classes concretas para representar os planos e as ações. Na Fig. 11 estão representadas as classes do framework associadas ao módulo que representa organizações e as classes relativas à instanciação do framework para implementar a organização *General_Store*. As classes da aplicação estão marcadas com um círculo. Classes foram criadas para representar a organização *General_Store*, um dos planos da organização e as ações do plano.

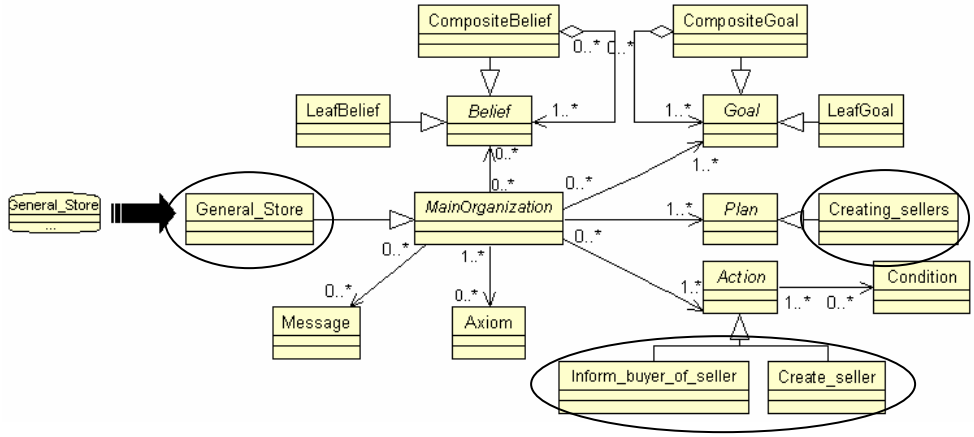


Fig. 11. Transformação de MAS-ML para UML (ASF Framework)

Na Fig. 12 estão descritas partes do UML XMI que representam o modelo UML da Fig. 11. No trecho do UML XMI estão as classes do framework que compreendem o módulo de organizações assim como as classes da aplicação. As classes da aplicação descritas no XMI são: *General_Store* que representa a organização principal, *Inform_buyer_of_seller* e *Create_seller* que representam ações da organização e *Creating_sellers* que representa um plano. O método construtor da classe *General_Store* foi implementado para representar a instanciação das propriedades da organização modeladas na Fig. 6.

```
<UML:Model name="Project Name" visibility="public" xmi.id="S.1">
 <UML:Namespace.ownedElement>
  <UML:Class name="Message" namespace="S.1" xmi.id="S.2"/>
  <UML:Class name="CompositeBelief" namespace="S.1" xmi.id="S.3"/>
  <UML:Class name="Belief" namespace="S.1" xmi.id="S.4">...</UML:Class>
  <UML:Class name="LeafBelief" namespace="S.1" xmi.id="S.5"/>
  <UML:Class name="Axiom" namespace="S.1" xmi.id="S.14"/>
  <UML:Class name="LeafGoal" namespace="S.1" xmi.id="S.12"/>
  <UML:Class name="CompositeGoal" namespace="S.1" xmi.id="S.13"/>
  <UML:Class name="Goal" namespace="S.1" xmi.id="S.15">...</UML:Class>
  <UML:Class name="Action" namespace="S.1" xmi.id="S.7">
   <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="L.5"/>
   </UML:GeneralizableElement.generalization> ...
  </UML:Class>
```

FRAMEWORK

```
  <UML:Class name="Plan" namespace="S.1" xmi.id="S.10">
   <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="L.3"/>
   </UML:GeneralizableElement.generalization> ...
  </UML:Class>
  <UML:Class name="MainOrganization" namespace="S.1" xmi.id="S.20">
   <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="L.13"/>
   </UML:GeneralizableElement.generalization>
  </UML:Class>
  <UML:Class name="Inform_buyer_of_seller" namespace="S.1" xmi.id="S.6"/>
  <UML:Class name="Create_seller" namespace="S.1" xmi.id="S.11"/>
  <UML:Class name="Creating_sellers" namespace="S.1" xmi.id="S.8"/>
  <UML:Class name="General_Store" namespace="S.1" xmi.id="S.19">
  <UML:Classifier.feature>
   <UML:Operation xmi.id="S.22" name="General_Store" ownerScope="instance">
    <UML:ModelElement.stereotype>
     <UML:Stereotype xmi.idref="S.23"/>
    </UML:ModelElement.stereotype>
    <UML:BehavioralFeature.parameter>
     <UML:Parameter xmi.id="S.24" name="return" kind="return">
      <UML:Parameter.type>
       <UML:Class xmi.idref="S.19"/>
      </UML:Parameter.type>
     </UML:Parameter>
     <UML:Parameter xmi.id="S.25" name="theEnvironment" kind="in"/>
    </UML:BehavioralFeature.parameter>
   </UML:Operation>
   <UML:Method xmi.id="S.26" isQuery="false">
    <UML:Method.body>
     <UML:ProcedureExpression xmi.id="S.27" language="java" body="
Belief  Belief objectBelief = new LeafBelief ("Vector","sellers",null);
        this.beliefs.add (objectBelief);
Meta    Goal objectGoal = new
        LeafGoal("boolean","management_of_sellers","true");
        this.goals.add (objectGoal);
Plano   Plan objectPlan = new Creating_sellers ();
        objectPlan.setGoal (objectGoal);
        objectGoal.setPlan (objectPlan);
Ações   Action objectAction = new Create_seller ();
        this.actions.add (objectAction);
        Condition objCondition=new Condition("boolean","sellerCreated","")
        objectAction.setPostCondition( objectCondition );
        objectPlan.setAction (objectAction);
        objectAction = new Inform_buyer_of_seller ();
        this.actions.add (objectAction);
Axioma  objectPlan.setAction (objectAction);
        Axiom objectAxiom = new
        Axiom("boolean","Send_information_about_sale","true");
        this.axioms.add(objectAxiom); "/>
    </UML:Method.body>
    <UML:Method.specification>
     <UML:Operation xmi.idref="S.22"/>
    </UML:Method.specification>
   </UML:Method>
  </UML:Classifier.feature>
  </UML:Class>
```

APLICAÇÃO

```
  <UML:Generalization xmi.id="L.3">
   <UML:Generalization.child>
    <UML:GeneralizableElement xmi.idref="S.8"/>
   </UML:Generalization.child>
   <UML:Generalization.parent>
    <UML:GeneralizableElement xmi.idref="S.10"/>
   </UML:Generalization.parent>
  </UML:Generalization>
  <UML:Generalization xmi.id="L.5">
   <UML:Generalization.child>
    <UML:GeneralizableElement xmi.idref="S.11"/>
   </UML:Generalization.child>
   <UML:Generalization.parent>
    <UML:GeneralizableElement xmi.idref="S.7"/>
   </UML:Generalization.parent>
  </UML:Generalization> ...
</UML:Namespace.ownedElement></UML:Model>
```

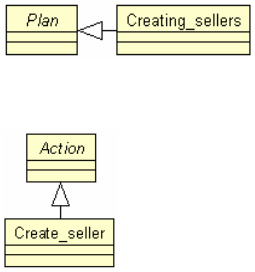


Fig. 12. UML XMI referente ao modelo apresentado na figura 11

3.4 Modelo PSM

O UML XMI gerado na etapa anterior representa o modelo específico de plataforma da aplicação (modelos PSM). Utilizando-se ferramentas gráficas que importem arquivos XMI, é possível gerar modelos UML da aplicação. Exemplos de ferramentas que já provêem suporte a geração de diagramas UML a partir do XMI são Together [40], Rational Rose[32], Poisedon[31].

O modelo UML gerado nesta etapa é um diagrama de classes UML que contém as classes do framework ASF e as classes correspondentes à aplicação que instancia o framework. Todas as entidades, propriedades e os relacionamentos entre as entidades modeladas nos três diagramas estáticos de MAS-ML (diagrama de classes estendido, diagrama de organizações e diagrama de papéis) de acordo com a descrição da aplicação são transformados, com base no framework ASF, gerando o diagrama de classes UML da aplicação. Na Fig. 13 estão representadas todas as classes do framework relacionadas ao módulo de organizações, algumas classes de ações e planos da organização *General_Store* e todas as classes que representam o ambiente, a organização, o objeto, os agentes, os papéis de agentes e os papéis de objeto modelados no diagrama de organizações da Fig. 5.

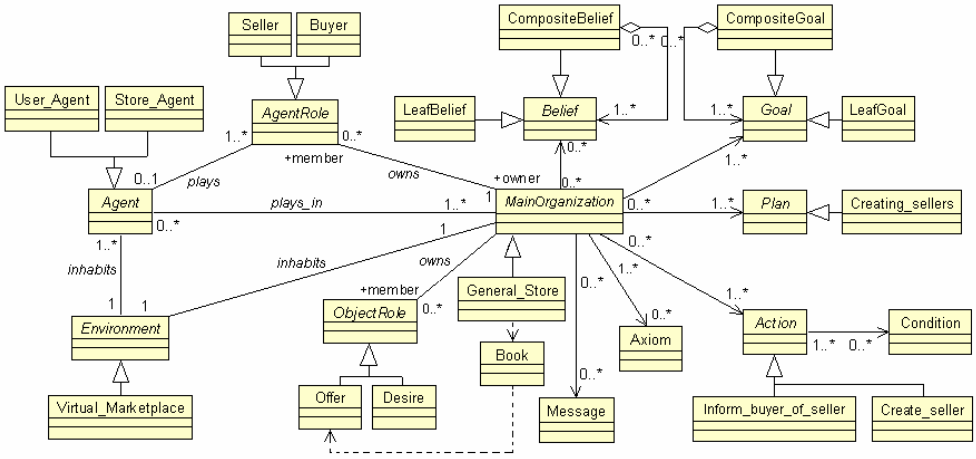


Fig. 13. Modelo UML com as entidades do diagrama apresentado na figura 5

3.5 Transformação do PSM para código

A última etapa do processo de desenvolvimento proposto é a transformação dos modelos UML, resultantes da etapa anterior, em código. Este tipo de transformação corresponde a última etapa de MDA que transforma o modelo PSM em código. A Fig. 14 ilustra o código da classe *General_Store*

gerado de acordo com o UML XMI representado na Fig. 12 . A geração de código a partir de modelos UML já é provida por ferramentas que importam o XMI e geram modelos UML.

```
public class General_Store extends MainOrganization{
    public General_Store (Environment theEnvironment){ ...
        Belief objectBelief = new LeafBelief ("Vector", "sellers", null);
        this.beliefs.add (objectBelief);
        Goal objectGoal = new
          LeafGoal("boolean","management_of_sellers","true");
        this.goals.add (objectGoal);
        Action objectAction = new Create_seller ();
        this.actions.add (objectAction);
          Condition objCondition = new Condition("boolean","sellerCreated","")
          ObjectAction.setPostCondition( objectCondition );
        Plan objectPlan = new Creating_sellers ();
        objectPlan.setGoal (objectGoal);
        objectPlan.setAction (objectAction);
        objectGoal.setPlan (objectPlan);
        objectAction = new Inform_buyer_of_seller ();
        this.actions.add (objectAction);
        objectPlan.setAction (objectAction);
        Axiom objectAxiom = new
          Axiom("boolean","Send_information_about_sale","true");
        this.axioms.add(objectAxiom);...
    }...
}
```

Fig. 14. Código referente a classe General_Store gerado pela Ferramenta Poseidon

4. Ferramenta de Apoio ao Processo de Desenvolvimento

Para viabilizar o processo de desenvolvimento proposto, foi desenvolvida uma ferramenta que a modelagem e a implementação de SMA. Esta ferramenta, similar as ferramentas CASE existentes para UML, permite ao usuário modelar graficamente seus sistemas de forma rápida e simples utilizando-se MAS-ML. O armazenamento dos modelos desenhados na ferramenta é feito no formato MAS-ML XMI.

A ferramenta MAS-ML tem também como outra funcionalidade realizar a transformação de MAS-ML XMI para UML XMI, de acordo com o framework ASF. Esta transformação será baseada no mapeamento dos elementos MAS-ML representados em XMI, em modelos UML também representados em XMI. Para realizar esta transformação o usuário necessitará apenas optar por salvar a modelagem feita em MAS-ML no formato UML XMI.

A transformação do modelo PSM em código, ou seja, dos modelos UML gerados a partir dos modelos MAS-ML é concluída utilizando-se ferramentas CASE já existentes no mercado que tenham a funcionalidade de transformar o UML XMI em código, como exemplo temos Together, Rational Rose.

5. Trabalhos Relacionados

Vallecillo e os outros autores [41] mostraram como MDA pode ser utilizado para derivar implementações de SMA a partir de modelos orientados a agentes, independente da metodologia usada e da plataforma selecionada. Este trabalho propõe a utilização de MDA para transformar modelos de acordo com a metodologia Tropos em modelo Malaca. Na fase de análise, Tropos é utilizado para detalhar as interações, capacidades e planos dos agentes. Esta modelagem é então traduzida para a descrição de agentes definido em Malaca. A transformação entre os modelos definidos em Tropos para o modelo Malaca, não é totalmente automatizado, necessitando assim a intervenção manual em algumas regras de transformação, diferentemente do trabalho proposto neste documento onde a transformação de um modelo para outro é realizado de forma automática. Além disso, em [41] não é tratada a transformação do modelo Malaca em código, enquanto que no processo proposto esta etapa está endereçada através da utilização de ferramentas CASE já existentes capazes traduzir modelos no formato UML XMI em código.

Novikay[22] analisa como GR[9] baseado no modelo visual de Tropos pode estar relacionado a MDA. A abordagem de MDA é interpretada pelo autor como uma atividade de modelagem visual onde modelos mais abstratos são refinados em modelos mais detalhados utilizando técnicas de transformação. Este trabalho cobre apenas a etapa de requisitos existente em Tropos. O processo proposto contempla todas as etapas necessárias para o desenvolvimento de uma aplicação, desde a modelagem até a implementação.

Em [16] Kasakov e os outros autores desenvolvem uma metodologia baseada na abordagem orientada a modelos para o desenvolvimento de sistemas distribuídos de agentes móveis. Neste trabalho, é definido um modelo conceitual de agentes móveis para ambientes distribuídos, descrevendo um conjunto de componentes, representados por uma coleção de agentes móveis inteligentes. A metodologia apresentada é referente a um domínio de aplicação específico, enquanto que o trabalho proposto neste documento não está limitada a um domínio específico.

Thiefaine e outros autores[39] propõe uma arquitetura baseada em modelos denominada MetaDIMA. A MetaDIMA é inspirado em MDA e propõe realizar a ligação entre as arquiteturas existentes para agentes com suas ferramentas de desenvolvimento e metodologias também baseadas em agentes. O objetivo principal desta proposta é iniciar o processo de desenvolvimento baseando-se em uma arquitetura existente e em suas ferramentas e elaborar meta-modelos, baseados nas arquiteturas já existentes para SMA. Este trabalho tem uma dependência muito grande da plataforma DIMA. No processo proposto neste documento, o desenvolvedor está apto a escolher diferentes plataformas existentes para agentes. A modelagem de um SMA gerada utilizando-se MAS-ML corresponderá aos modelos PIM propostos em MDA. Tais modelos são independentes da plataforma de implementação. MAS-ML não restringe ou especifica a plataforma onde os modelos devem ser implementados.

6. Conclusões e Trabalhos Futuros

Este trabalho apresenta um processo de desenvolvimento de SMA desde a sua modelagem até a sua implementação utilizando a abordagem MDA. Este processo é composto por 3 etapas: (i) modelagem de SMA através da linguagem MAS-ML, que é uma linguagem de modelagem especifica para sistemas multi-agentes independente de plataforma; (ii) geração de modelos UML dependentes de plataforma, baseada no ASF Framework, a partir dos modelos MAS-ML e (iii) geração do código a partir dos modelos UML. A geração dos modelos UML em código é feita através da utilização da tecnologia XMI. Cada etapa do processo proposto está relacionada as etapas do MDA, onde os modelos MAS-ML correspondem ao modelo PIM e os modelos UML ao modelo PSM de MDA.

O processo de desenvolvimento para SMA proposto se beneficia das vantagens decorrentes do uso de MDA. Quatro principais características podem ser destacadas: portabilidade, reusabilidade, interoperabilidade e baixo acoplamento. Os modelos MAS-ML são modelos portáveis para diferentes plataformas de implementação. Tais modelos descrevem o SMA sem incluir detalhes de implementação dos mesmos. Sendo assim, um modelo MAS-ML pode ser reutilizado por diferentes desenvolvedores que desejem implementar o sistema utilizando diferentes plataformas.

A interoperabilidade dos modelos gerados no processo proposto se dá pela descrição destes modelos utilizando-se XMI. Tanto os modelos MAS-ML como os modelos UML são interoperáveis devido ao uso de MAS-ML XMI e UML XMI. Já o baixo acoplamento existente entre o modelo conceitual da aplicação e o modelo da solução (modelo computacional) é uma característica comum aos processos de desenvolvimento que se baseiam em MDA. MDA propõe o desenvolvimento de modelos independentes de plataforma e de modelos dependentes de plataforma separadamente. Em nossa abordagem, os modelos MAS-ML são modelos PIM que não incluem detalhes relacionados a plataforma escolhida. Tais detalhes estão presentes apenas nos modelos UML, modelos PSM.

Com o objetivo de melhorar a ferramenta que dá suporte ao desenvolvimento de sistemas multi-agente utilizando MDA dois pontos importantes precisam ser analisados. Atualmente a ferramenta não dá suporte a geração de código a partir dos modelos dinâmicos de MAS-ML. Apenas os modelos estruturais são transformados em código Java. É de suma importância para o processo de geração de código da aplicação a partir de modelos de alto nível que a modelagem do comportamento das entidades seja considerada.

Outro ponto importante que deve ser considerado é a consistência entre os modelos de diferentes níveis de abstração. Mudanças nos modelos MAS-ML devem ser propagadas para os modelos UML e para o código. Contudo, nem todas as atualizações do código da aplicação devem ser propagas para os modelos UML e para os modelos MAS-ML. Mudanças relacionadas a detalhes de implementação não devem ser propagadas para os modelos MAS-ML e nem todas as mudanças relativas a implementação devem ser propagadas para os modelos UML. Sendo assim, a consistência entre o modelo MAS-ML, o modelo UML e o código é importante para uma ferramenta cujo propósito é dar suporte ao processo de desenvolvimento de sistemas.

Referências

1. Agent Based Digital Library, http://www.sics.se/isl/diglib/, outubro, 2004.
2. Bauer, B., Muller, J. e Odell, J. "Agent UML: A Formalism for Specifying Multiagent Interaction". Springer-Verlag, Berlin, 91-103. 2001.
3. Bettin, J. "Model Driven Architecture: Implementation and Metrics". SoftMetaWare. 2003.
4. Bergholtz, M., Jayaweera, P., Johannesson, P., Wohed, P. "Modeling Institutional, Communicative and Physical Domains". Agent-oriented Information Systems. LNAI3030. 2004.
5. Bresciani, P., Donzelli, P. "A Practical Agent-Based Approach to Requirements Engineering". Agent-oriented Information Systems, LNAI3030. 2004.
6. Bresciani, P. et al. "Tropos: An Agent-Oriented Software Development Methodology". International Journal of Autonomous Agents and Multi-Agents Systems. 2003.
7. Brown, A. "An Introduction to Model Driven Architecture Part I: MDA and Today's Systems". The Rational Edge (IBM). 2004.
8. Caire, G., Leal, F., Chainho, P. e Evans, R. "Agent Oriented Analysis using MESSAGE/UML". Conference'00. 2000.
9. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R. e Löwe, M. "Algebric Approaches to Graph Transformation. Part 1: Basic Concepts and Double Pushout Approach". Handbook of Graph Grammars and Computing by Graph Transformation". Foundations. 1997.
10. Ding, H. "Exploiting Intelligent Agent Negotiation Strategy for Constructing Federated Digital Libraries". IAWTIC'04. 2004.
11. Frankel, D. S. "Model Driven Architecture: Applying MDA to Enterprise Computing". OMG Press. 2003.
12. Grose, T., Doney, G., Brodsky, S. "Mastering XMI". OMG Press. 2002.
13. He, M., Jennings, N.R. e Leung, H. "On Agent-Mediated Eletronic Commerce". IEEE Trans on Knowledge and Data Engineering 15 (4) 985-1003. 2003.
14. Iba, T., Matsuzawa, Y. e Aoyama, N. "From Conceptual Models to Simulation Models: Model Driven Development of Agent-Based Simulations". WEHIA'04. 2004.
15. Jezequel, J.M., Emmerich, W. "Panel MDA in Pratice". Proceedings of the 26th International Conference on Software Engineering (ICSE'04). 2004.
16. Kazakov, M., Abdulrab, H. e Debarbouille, G. "A model Driven Approach for Design of Mobile Agent Systems for Concurrent Enginneering: MAD4CE Project". Rapport Interne 01-002, Université et INSA de Rouen. 2002.
17. Kepple, A., Warmer, J., Bast, W. "MDA Explained: The Model Driven Architecture – Pratice and Promise". Addison Presley. 2003.
18. Kia, A.,Aïmeur, E. e Kropf, P. "aKIA Automobile: a Multi-Agent System in E-Commerce". International Conference on Electronic Commerce (ICERC-5), volume CD-ROM. 2002.
19. Lomuscio, A. R., Wooldridge, M.e Jennings, N. "A classification scheme for negotiation in electronic commerce". International Journal of Group Decision and Negotiation", vol.12, no.1, 31-56. 2003.
20. Mellor, S.J., Scott, K., Uhl, A., Weise, D. "MDA Distilled: Principles of Model Driven Architecture". Addison Presley. 2004.
21. Miller, J. e Mukerji. "Model Driven Architecture". 2001. Disponível em: http://www.omg.org/mda/presentations.htm. Acessado em: Novembro, 02, 2004.
22. Novikau, A. "Model Driven Architecture approach in Tropos". Technical Report T04-06-03, Istituto Trentino di Cultura. 2004.
23. Odell, J., Dyke, Parunak, H. e Bauer, B. "Extending UML for Agents". Proceedings of AOIS Workshop at AAAI. 2000.
24. Oliveira, T.C., Mathias Filho, I., Lucena, C.J.P., Cowan, D.D., Alencar, P. "Software Process Representation and Analysis for Framework Instatiation". IEEE - Transaction on Software Engineering. 2004.
25. OMG, "OMG MDA Guide". Versão 1.0.1. 2002. Disponível em: http://www.omg.org/docs/omg/03-06-01.pdf. Acessado em: Novembro, 02, 2004.

26. OMG, "Meta Object Facility (MOF) 2.0 Core Specification". Versão 2.0, 2003. Disponível em: http://www.omg.org/docs/ptc/03-10-04.pdf. Acessado em: Novembro, 02, 2004.
27. OMG, "OMG XML Metadata Interchange". Versão 1.2. 2003. Disponível em: http://www.omg.org/docs/formal/02-01-01.pdf. Acessado em: Novembro, 02, 2004.
28. OMG, "OMG Unified Modeling Language Specification". Versão 1.5, 2003. Disponível em: http://www.omg.org/docs/formal/03-03-01.pdf. Acessado em: Novembro, 02, 2004.
29. Pokahr, A., Braubach, L. e Lamersdorf, W. "Jadex: Implementing a BDI-Infrastructure for JADE Agents". EXP - In Search of Innovation (Special Issue on JADE), vol. 3, no. 3 , Telecom Italia Lab, Turin, Italy, S. 76-85. 2003.
30. Pokahr, A., Braubach, L. e Lamersdorf, W. "Jadex: A Short Overview". Net.ObjectDays AgentExpo (to be published). 2004.
31. Poseidon. Disponível em: http://www.gentleware.com. Acessado em: Novembro, 02, 2004.
32. Rational Rose. Disponível em: http://www-306.ibm.com/software/rational/. Acessado em: Novembro, 02, 2004.
33. Silva, V., Garcia A., Brandão A., Chavez C., Lucena C., Alencar P. "Taming Agents and Objects in Software Engineering". In Garcia A., Lucena C., Zamboneli F., Omicini A, and Castro J. (eds.) Software Engineering for Large-Scale Multi-Agent System, LNCS, Springer.
34. Silva V., Lucena C. "From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language", Technical Report CS2003-03, School of Computer Science, University of Waterloo, Canada (submitted to JAAMAS). 2003.
35. Silva, V., Choren, R. "Using the MAS-ML to Model a Multi-Agent System". SELMAS, 129-148. 2003.
36. Silva, V., Cortés, M., Lucena, C. "An Object-Oriented Framework for Implementing Agent Societies". MCC32/04. Technical Report, Computer Science Department, PUC-Rio. Rio de Janeiro, Brazil. 2004.
37. Sycara, K., Giampapa, J.A., Langley, B.K., Paolucci, M., "The RETSINA MAS, a Case Study" Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications, Garcia, A., Lucena, C., Zambonelli, F., Omici, A., Castro, J., ed., Springer-Verlag, Berlin Heidelberg, Vol. LNCS 2603, July, 2003, 232—250. 2003.
38. Sycara, K., Paolucci, M., van Velsen, M., Giampapa, J., "The RETSINA MAS Infrastructure", in the special joint issue of Autonomous Agents and MAS, vol. 7, nos. 1 e 2. 2003.
39. Thiefaine, A. e Guessoum, Z. "MetaDIMA: a Model Driven Architecture for Multi-Agent Systems".
40. Together. Disponível em: http://www.borland.com/together. Acessado em: Novembro, 02, 2004.
41. Vallecillo, A., Amor, M. e Fuentes, L. "Bridging the Gap Between Agent-Oriented Design and Implementation Usind MDA". Proceedings of the AAMAS Agent-Oriented Software Engineering Workshop (AOSE'04). 2004.
42. Warmer, J., Kepple, A. "The Object Constraint Language: Getting Your Models Ready for MDA". 2nd Edition. Addison Presley. 2003.
43. Wooldridge, M., Jennings, N. R. e Kinny, D. . "The Gaia methodology for agent-oriented analysis and designs". Journal of Autonomous Agent and Multi-Agent Systems. 2000.
44. Wooldridge, M. e Jennings, N. "Applications of Intelligent Agents". Jennings, J. e Wooldridge, M. (Eds.), "Agent Technology: Foundations, Applications, and Markets", vol.3, no.28. 1998.