4
Um Processo para Desenvolvimento de Sistemas Multi-Agentes

O processo de desenvolvimento de sistemas complexos e de larga-escala, como os sistemas multi-agentes, compreende a construção de diversos modelos baseados em uma série de requisitos. 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 práticos para o desenvolvimento de sistemas multi-agentes. Os métodos existentes na literatura atualmente têm como principal problema o fato de endereçarem apenas fases específicas do ciclo de desenvolvimento do software, fazendo com que algumas fases importantes para o desenvolvimento sejam deficientes ou não sejam atendidas.

O processo de desenvolvimento proposto neste trabalho tem como principal objetivo dar suporte ao desenvolvimento de aplicações orientadas a agentes utilizando a abordagem MDA, tornando este processo simples e de fácil compreensão pelo desenvolvedor. Através deste processo será possível criar modelos independentes de plataforma (PIMs), transformá-los em modelos dependentes de plataforma (PSMs) e a partir deste modelo, produzir seu código.

O processo de desenvolvimento proposto opta por utilizar a aplicabilidade e o potencial de MDA no âmbito de sistemas multi-agentes, visando identificar todas as etapas e atividades envolvidas no processo de desenvolvimento, como também definir os artefatos gerados em cada etapa. Pode ser visto também como um método que combina o processo de desenvolvimento de sistemas multi-agentes com os conceitos, princípios e tecnologias de MDA, tirando proveito dos seus benefícios como reusabilidade de modelos, transformações automatizadas e preservação de investimentos do desenvolvimento da aplicação.

É importante ressaltar que o processo de desenvolvimento de SMA baseado em MDA pode acontecer de diferentes formas, conforme apresentado na Figura 4. Uma vez definido o modelo de descrição do sistema na etapa CIM,

é possível gerar diferentes modelos independentes de plataforma (PIM) utilizando-se diferentes abordagens. A Figura 4 exemplifica dois tipos de abordagens diferentes que podem ser utilizadas. A descrição do sistema (CIM) pode ser interpretada através da linguagem de modelagem MAS-ML ou AUML [6]. Após modelado o sistema utilizando-se uma dessas linguagens de modelagem, correspondendo a etapa PIM de MDA, esses modelos devem ser transformados em modelos específicos de plataforma, PSMs. No exemplo apresentado, os modelos MAS-ML podem ser transformados em diferentes plataformas como o framework ASF, a arquitetura JADEX [19, 20] ou RETISINA [47]. As plataformas mencionadas são exemplos de plataformas específicas para a implementação de SMA. A última etapa do processo consiste em transformar os PSMs em código, de acordo com a linguagem de implementação da plataforma.

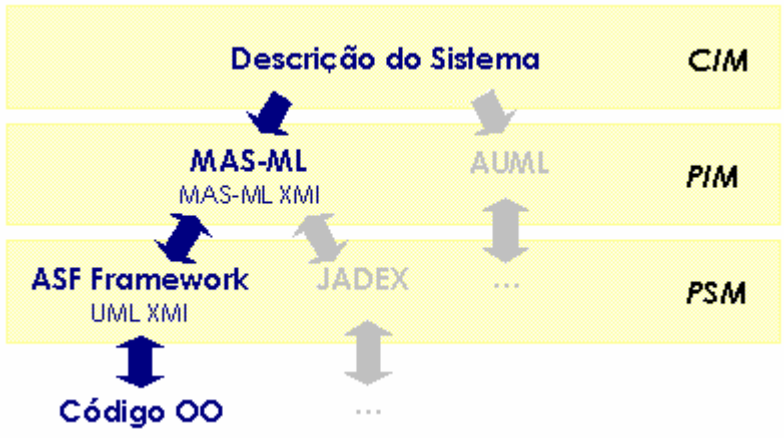


Figura 4 – Processo de Desenvolvimento Proposto

O processo de desenvolvimento proposto neste trabalho compreenderá as três últimas etapas presentes em MDA, ou seja, as etapas PIM, PSM e o próprio código. A etapa CIM e sua transformação em modelos independentes de plataforma (PIM) não serão abordadas neste trabalhado. Deste modo, a primeira etapa proposta em nossa abordagem corresponderá à etapa PIM de MDA, responsável por geração de modelos independentes de plataforma, que será realizada através da geração de modelos utilizando a linguagem MAS-ML específica para SMA. Os modelos MAS-ML são modelos independentes de plataforma de implementação. MAS-ML não restringe ou especifica a plataforma onde serão implementados os modelos. Diferentes plataformas orientadas a objetos podem ser utilizadas para implementar sistemas modelados utilizando-se MAS-ML.

Após a modelagem do SMA, utilizando a linguagem MAS-ML, tem-se a próxima etapa do processo de desenvolvimento, que é responsável pela

modelagem do sistema já com características da plataforma em que o mesmo será implementado. Os modelos gerados neste processo serão modelos UML dependentes de plataforma (modelos PSM de acordo com MDA). Estes modelos serão dependentes de plataforma pois escolhemos a utilização do framework ASF para a implementação de SMA. Portanto, os modelos UML estarão acrescidos de detalhes de implementação do sistema definidos no framework.

A terceira e última etapa do processo de desenvolvimento proposto corresponde à geração do código a partir dos modelos PSM (modelos UML) gerados na etapa anterior. Neste trabalho, o código gerado é descrito utilizando-se a linguagem orientada a objetos Java [48]. Durante esta Seção será esclarecida com mais detalhe cada etapa e a transformação de uma etapa para outra.

4.1. Modelo CIM

Conforme já mencionado anteriormente neste trabalho, o modelo CIM é responsável por descrever o domínio da aplicação, sem ater aos detalhes da estrutura do sistema. Os modelos desta etapa devem ser independentes das abstrações computacionais utilizadas na solução de 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 [49]. Nesta abordagem os autores mapeiam modelos de features que definem um problema para modelos UML baseando-se em um framework orientado a objetos.

4.2. Modelo PIM

Esta etapa corresponde à primeira etapa do processo de desenvolvimento aqui proposto e tem como propósito modelar SMA utilizando a linguagem MAS-ML especifica para este tipo de sistema. Esta etapa está em concordância com a etapa PIM de MDA pelo fato de MAS-ML ser uma linguagem independente de plataforma. Nela são modelados os diagramas estruturais de MAS-ML (diagrama de classes de UML estendido, diagrama de organização e diagrama de papéis). Não fará parte do escopo deste trabalho mostrar as transformações entre modelos e geração de código a partir dos diagramas dinâmicos de MAS-ML (diagrama de seqüência de UML estendido).

A preferência por se utilizar a linguagem MAS-ML como linguagem de modelagem dos SMA nesta etapa do processo foi influenciada por três fatores. Primeiramente, esta linguagem não é uma linguagem dependente de plataforma de implementação. Pelo fato de estender UML, MAS-ML utiliza conceitos de orientação a objetos mas não restringe a implementação de seus modelos a uma linguagem de programação específica. Outro fator importante é o fato de MAS-ML, assim como as outras linguagens que estendem UML como AUML, ser compatível com o framework MOF [50], o que facilita o processo proposto pois utiliza-se a tecnologia XMI durante as transformações entre modelos. Por fim, MAS-ML aborda abstrações como organizações, ambientes e papéis de agentes que não são abordadas adequadamente por outras linguagens de modelagem como AUML [51]. Sem o uso destas abstrações não é possível modelar, por exemplo, agentes desempenhando diferentes papéis em diferentes organizações.

Os diagramas estruturais MAS-ML modelados de acordo com os requisitos do sistema farão parte desta etapa do processo de desenvolvimento, onde os mesmos deverão ser construídos. Os elementos e relacionamentos de MAS-ML que foram estendidos no meta-modelo de UML para atender as características de SMA estão representados nas tabelas a seguir. Na Tabela 2 estão representados graficamente os novos elementos e suas respectivas propriedades. Na Tabela 3 estão representados os relacionamentos criados por MAS-ML e que podem estar descritos nestes diagrama estruturais. Vale a pena ressaltar que as propriedades presentes na caixa central de cada entidade são referentes aos aspectos estruturais da entidade, enquanto que a caixa inferior está relacionada aos aspectos comportamentais.

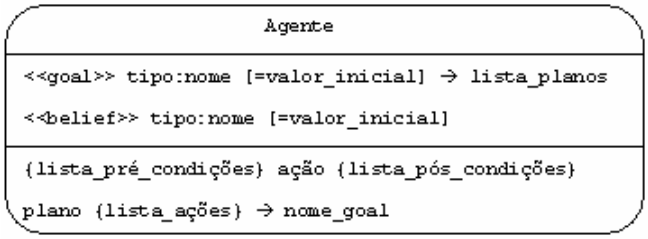
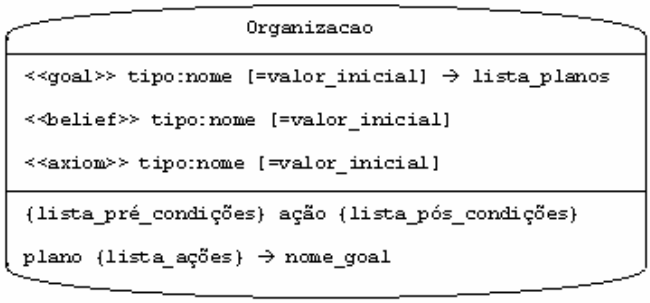
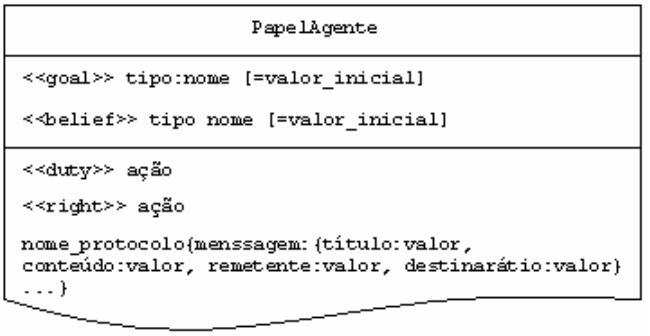
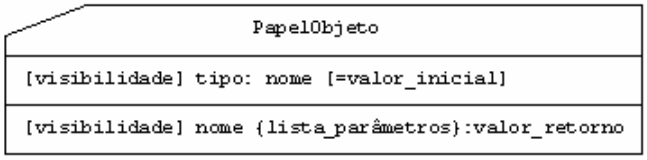
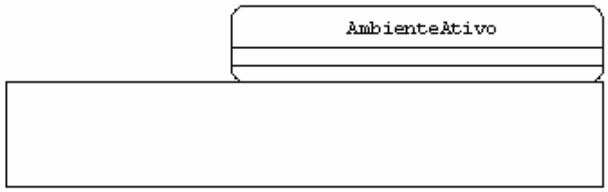
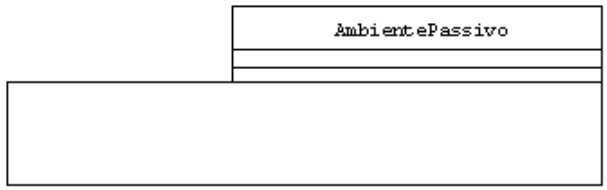
Entidade	Representação Gráfica
Agente	
Organização	
Papel de Agente	
Papel de Objeto	
Ambiente Ativo	
Ambiente Passivo	

Tabela 2 – Representação gráfica das entidades de MAS-ML

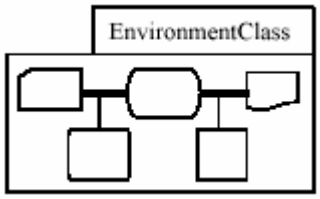
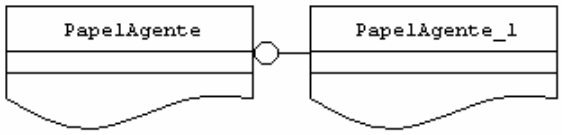
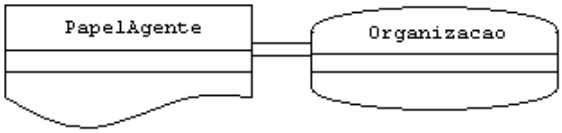
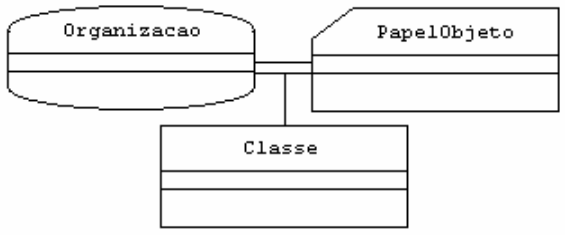
Relacionamento	Representação Gráfica
Inhabit	
Control	
Ownership	
Play	

Tabela 3 - Representação gráfica dos relacionamentos de MAS-ML

Após modelar o sistema com a linguagem MAS-ML, gerando assim modelos independentes de plataforma (PIMs), os mesmos serão transformados em modelos específicos de plataforma (PSMs), de acordo com detalhamento presente na Seção seguinte.

4.3. Transformação do modelo PIM para o modelo PSM

A transformação de modelos MAS-ML para modelos UML é realizada em dois passos. O primeiro passo corresponde a descrever de forma textual os modelos da aplicação graficamente representados utilizando-se MAS-ML. Esta descrição textual dos modelos gráficos de MAS-ML é feita utilizando-se o formato padrão XMI.

Para a representação dos modelos MAS-ML em XMI foi criado o MAS-ML DTD que é uma extensão do UML DTD [18] , referente a versão 1.5 de UML, proposto pela OMG. A extensão do UML DTD foi definida de acordo com a extensão do meta-modelo de UML proposto por MAS-ML. O XMI da aplicação gerado a partir do MAS-ML DTD é chamado neste documento de MAS-ML XMI.

Utilizando-se o MAS-ML XMI é possível descrever textualmente todas as abstrações existentes na aplicação como agentes, organizações, ambientes, papéis e objetos.

O segundo passo referente a esta etapa compreende o refinamento do MAS-ML XMI para o UML XMI de acordo com o framework ASF. É este refinamento que representa a transformação dos PIMs para PSMs. Os PIMs descritos em MAS-ML XMI são transformados em PSMs 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 contém os detalhes da implementação da aplicação, uma vez que estará relacionado ao framework ASF.

4.3.1. Primeiro Passo: Criação do MAS-ML XMI

Neste passo, todos os diagramas estruturais de MAS-ML que modelam SMA são descritos em MAS-ML XMI. Para a descrição textual dos modelos MAS-ML em XMI foi necessária a criação do MAS-ML DTD, que é uma extensão do UML DTD. O UML DTD representa um mapeamento dos elementos existentes em UML para XML, através da representação do meta-modelo de UML em um formato ASCII padronizado. O MAS-ML DTD está de acordo com o meta-modelo de MAS-ML. A extensão realizada procurou obedecer o mesmo padrão existente no UML DTD já especificado, onde foram criados novos elementos e atributos de acordo com a especificação de MAS-ML.

Pelo fato do MAS-ML DTD ser muito extenso (vide Anexo I), são apresentados nesta Seção exemplos de trechos de um documento MAS-ML XMI oriundos da extensão realizada para ilustrar as novas entidades e relacionamentos propostos por MAS-ML. Estes trechos correspondem à transcrição em MAS-ML XMI dos elementos e relacionamentos apresentados nas Tabela 2 e Tabela 3 da Seção anterior. Alguns detalhes dos trechos foram omitidos visando não estender demais este documento.

Para representar o agente existente em MAS-ML é apresentado o trecho de MAS-ML XMI abaixo (Quadro 1). Este trecho reflete a representação gráfica do agente presente na Tabela 2. No Quadro 1, temos em negrito os *tags* que foram inseridos durante a extensão do UML DTD para a representação do agente e suas propriedades. Como é possível perceber, foi criado o *tag* “MASML:AgentClass" para representar a entidade Agente. De acordo com a

definição de MAS-ML, um agente pode ter objetivos, crenças, planos e ações associados. O objetivo (*goal*) e a crença (*belief*) são representados no XMI através de estereótipos associados ao atributo. A ligação entre eles e os estereótipos que os definem é feita através de indicadores de referência conforme ilustrado pelas linhas tracejadas.

Para a representação das ações exercidas por um agente foi criada a *tag* ”MASML:AgentAction”. As pré e pós condições de cada ação são definidas através das *tags* “MASML:AgentAction.preConditions” e “MASML:AgentAction.posConditions”, respectivamente. Dentro destas *tags* podem ser inseridas uma ou mais condições representadas por “Condition”. Visando descrever um plano de um agente em MAS-ML XMI foi definida a *tag* “MASML:AgentPlan”. As ações e as metas relacionadas ao mesmo são representadas pelas *tags* “MASML:AgentPlan.actions” e “MASML:AgentPlan.goal”, respectivamente. A associação entre o plano e seu objetivo é realizada através do identificador de propriedade marcado com um círculo no Quadro 1.

```
<!--Agente-->
<MASML:AgentClass isAbstract="false" name="Agente" visibility="public"
xmi.id="S.2">
  <UML:Classifier.feature>
    <UML:Attribute name="nome" stereotype="SS.1" visibility="public" xmi.id="S.3">
      <UML:Attribute.initialValue>
        <UML:Expression body="valor_inicial"/>
      </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:Attribute name="nome" stereotype="SS.3" visibility="public" xmi.id="S.4">
      <UML:Attribute.initialValue>
        <UML:Expression body="valor_inicial"/>
      </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>
    <MASML:AgentAction name="acao" visibility="public" xmi.id="S.5">
      <MASML:AgentAction.preConditions>
        <Condition default="valor_inicial" name="pre_condicao" type="G.1"/>
      </MASML:AgentAction.preConditions>
      <MASML:AgentAction.posConditions>
        <Condition default="valor_inicial" name="pos_condicao" type="G.1"/>
      </MASML:AgentAction.posConditions>
    </MASML:AgentAction>
    <MASML:AgentPlan name="plano" visibility="public" xmi.id="S.6">
      <MASML:AgentPlan.actions>
        <Action xmi.idref="S.5"/>
      </MASML:AgentPlan.actions>
      <MASML:AgentPlan.goal stereotype="SS.1" xmi.idref="S.3"/>
    </MASML:AgentPlan>
  </UML:Classifier.feature>
```

```
</MASML:AgentClass>
<UML:Stereotype name="Goal" xmi.id="SS.1">
  <UML:Stereotype.baseClass>S.3</UML:Stereotype.baseClass>
</UML:Stereotype>
<UML:Stereotype name="Belief" xmi.id="SS.3">
  <UML:Stereotype.baseClass>S.4</UML:Stereotype.baseClass>
</UML:Stereotype>
<UML:DataType name="tipo" visibility="public" xmi.id="G.1"/> ←
```

Quadro 1 – Representação em MAS-ML XMI de um agente e suas propriedades

A representação dos tipos de dados das propriedades acima, como por exemplo da crença, é realizada através da associação de “UML:DataType”, já existente no UML DTD, com sua referência no atributo (apontados no código por uma seta).

Devido ao fato do elemento organização ser, em MAS-ML, uma meta-classe filha de agente, ele apresenta propriedades semelhantes às de um agente, apenas com o acréscimo da propriedade que representa as regras de uma organização, denominada axioma. No Quadro 2 está representado o trecho MAS-ML XMI da organização representada graficamente na Tabela 2. Em virtude de algumas propriedades da organização já terem sido apresentadas no MAS-ML XMI do agente, as mesmas foram omitidas no trecho abaixo, são elas: objetivo, crença, ação e plano.

```
<!—Organização-->
<MASML:OrganizationClass isAbstract="false" isMain="false" name="Organizacao"
visibility="public" xmi.id="S.2">
  <UML:Classifier.feature>
    <UML:Attribute name="nome" stereotype="SS.5" visibility="public" xmi.id="S.5">
      <UML:Attribute.initialValue>
        <UML:Expression body="valor_inicial"/>
      </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:Classifier.feature>
</MASML:OrganizationClass>
<UML:DataType name="tipo" visibility="public" xmi.id="G.1"/>
<UML:Stereotype name="Axiom" xmi.id="SS.5">
  <UML:Stereotype.baseClass>S.5</UML:Stereotype.baseClass>
</UML:Stereotype>
```

Quadro 2 – Representação em MAS-ML XMI de uma organização e suas propriedades

Pelo trecho acima, é possível perceber que a organização é representada em MAS-ML XMI através da *tag* “MASML:OrganizationClass”. E assim como a meta e o objetivo, o axioma também é tratado como um estereótipo de um atributo (linha tracejada).

Outra entidade que pode ser representada em MAS-ML é o papel de agente. O papel de agente tem como propriedades estruturais objetivos e crenças, cuja representação do MAS-ML XMI já foi apresentada na definição do agente. As outras propriedades deste elemento são os deveres (*rights*), as

obrigações (*duties*) e os protocolos de comunicação, todos representados no código de XMI ilustrado no Quadro 3. Por este trecho de código é possível identificar que os deveres e as obrigações são representados no XMI através de estereótipos associados a ação, conforme definido no meta-modelo de MAS-ML. A ligação entre eles e os estereótipos que os definem é feita através de indicadores de referência destacados pelas linhas tracejadas.

Com a finalidade de representar o protocolo desempenhado pelo papel de agente e as mensagens utilizada no mesmo, foram definidas as *tags* “MASML:AgentProtocol” e “MASML:AgentMessage”. Conforme é possível visualizar, a associação do protocolo com suas mensagens é realizada pela *tag* “MASML:AgentProtocol.messages”. As propriedades existentes em mensagens como: conteúdo, título, remetente e destinatário são representados por *tags* e atributos específicos destacados por uma chave no código.

```
<!-- Papel de agente -->
<MASML:AgentClass isAbstract="false" name="remetente" xmi.id="S.2"/>
<MASML:AgentClass isAbstract="false" name="destinatario" xmi.id="S.3"/>
<MASML:AgentRoleClass isAbstract="false"name="PapelAgente" xmi.id="S.4">
  <UML:Classifier.feature>
    <MASML:AgentAction name="acao" stereotype="SS.5" xmi.id="S.7"/>
    <MASML:AgentAction name="acao2" stereotype="SS.7" xmi.id="S.8"/>
    <MASML:AgentProtocol name="nome_protocolo" xmi.id="S.9">
      <MASML:AgentProtocol.messages>
        <MASML:AgentMessage receiver="S.3" sender="S.2">
          <MASML:AgentMessage.label>value</MASML:AgentMessage.label>
          <MASML:AgentMessage.content>value</MASML:AgentMessage.content>
        </MASML:AgentMessage>
      </MASML:AgentProtocol.messages>
    </MASML:AgentProtocol>
  </UML:Classifier.feature>
</MASML:AgentRoleClass>
<UML:Stereotype name="Duty" xmi.id="SS.5">
  <UML:Stereotype.baseClass>S.7</UML:Stereotype.baseClass>
</UML:Stereotype>
<UML:Stereotype name="Right" xmi.id="SS.7">
  <UML:Stereotype.baseClass>S.8</UML:Stereotype.baseClass>
</UML:Stereotype>
```

Quadro 3 - Representação em MAS-ML XMI de um papel de agente e suas propriedades

A representação do papel de objeto em MAS-ML XMI é muito semelhante à representação de uma classe em UML XMI, pois as propriedades de papel de objeto são atributos e métodos. A única *tag* criada para representar este elemento foi a “MASML:ObjectRoleClass” (Quadro 4).

```
<!-- Papel de objeto -->
<MASML:ObjectRoleClass isAbstract="false" name="PapelObjeto" xmi.id="S.2">
  <UML:Classifier.feature>
    <UML:Attribute name="nome" visibility="public" xmi.id="S.3">
      <UML:StructuralFeature.multiplicity>
        <UML:Multiplicity>
          <UML:Multiplicity.range>
            <UML:MultiplicityRange lower="1" upper="1"/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:StructuralFeature.multiplicity>
      <UML:Attribute.initialValue>
        <UML:Expression body="valor_inicial"/>
      </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:Operation isAbstract="false" name="nome" xmi.id="S.4">
      <UML:BehavioralFeature.parameter>
        <UML:Parameter kind="in" name="arg" type="G.1" xmi.id="XX.1">
          <UML:Parameter.defaultValue>
            <UML:Expression body="valor_inicial"/>
          </UML:Parameter.defaultValue>
        </UML:Parameter>
        <UML:Parameter kind="return" name="nome.return" type="G.2" xmi.id="XX.2"/>
      </UML:BehavioralFeature.parameter>
    </UML:Operation>
  </UML:Classifier.feature>
</MASML:ObjectRoleClass>
```

Quadro 4 - Representação em MAS-ML XMI de um papel de objeto e suas propriedades

Devido ao fato de um ambiente poder ser uma entidade ativa, como um agente ou passiva, como um objeto, existem duas formas de representá-lo. Se o ambiente for ativo, sua representação em XMI utilizará a *tag* “MASML:ActiveEnvironmentClass” e terá as mesmas propriedades de um agente. Caso o ambiente seja passivo, sua representação será “MASML:PassiveEnvironmentClass” e terá as mesmas propriedades de um objeto. Por já ter sido representado em quadros anteriores as propriedades de um agente e um objeto (Quadro 1 e Quadro 4), no Quadro 5 são apresentadas apenas as *tags* criadas para descrever estes dois tipos de ambientes.

```
<!–Ambientes-->
<MASML:PassiveEnvironmentClass name="AmbientePassivo" xmi.id="S.2"/>
<MASML:ActiveEnvironmentClass name="AmbienteAtivo" xmi.id="S.3"/>
```

Quadro 5 – Representação em MAS-ML XMI de ambiente ativo e passivo

Para se ter uma representação completa das abstrações presentes em MAS-ML, além das entidades, também é necessário representar os relacionamentos criados por esta linguagem. No Quadro 6 são apresentadas as descrições textuais em MAS-ML XMI desses relacionamentos, de acordo com a representação gráfica presente na Tabela 3. Para poder ilustrar no trecho abaixo os relacionamentos *control*, *ownership*, *inhabit* e *play* foram criados também os elementos envolvidos nessas relações. Os demais relacionamentos existentes nos diagramas estruturais de MAS-ML não serão representados pois não sofreram alteração com relação ao UML DTD.

```
<!--Entidades-->
<MASML:OrganizationClass isMain="false" name="Organizacao" xmi.id="S.1"/>
<MASML:AgentRoleClass name="PapelAgente" xmi.id="S.2"/>
<MASML:AgentRoleClass name="PapelAgente 1" xmi.id="S.3"/>
<MASML:PassiveEnvironmentClass name="AmbientePassivo" xmi.id="S.4"/>
<UML:Class name="Class" xmi.id="S.5"/>
<MASML:ObjectRoleClass name="PapelObjeto" xmi.id="S.6"/>
<!--Control-->
<MASML:Control controled="S.2" controller="S.3" name="" xmi.id="L.1"/>
<!-- Ownership-->
<MASML:Ownership member="S.2" name="" owner="S.1" xmi.id="L.2"/>
<!--Inhabit-->
```

```
<MASML:Inhabit citzen="S.2" environment="S.4" name="" xmi.id="L.3"/>
<!--Play-->
<MASML:Play member="S.5" name="" played="S.6" player="S.1" xmi.id="L.4"/>
```

Quadro 6 - Representação em MAS-ML XMI dos relacionamentos

Com o objetivo de representar os diagramas presentes em uma modelagem MAS-ML, foi criada a “MASML:Diagram”. Esta *tag* é um extensão da “UML:Diagram”, existente no UML.DTD. A necessidade de se criar um nova *tag* ocorre devido à necessidade de representar dentro dos diagramas elementos que não existem na linguagem UML, como por exemplo agente. No MAS-ML XMI as entidades são criadas dentro de uma estrutura denominada modelo e depois são referenciadas nas *tags* “MASML:Diagram” de acordo com a sua disposição nos diagramas. O trecho abaixo corresponde à criação de três diagramas (classes, organização e papel), denominados Class_Diagram, Organization_Diagram e Role_Diagram, onde no primeiro está representado uma organização, no segundo um agente e no terceiro um papel de agente, respectivamente.

```
<UML:Model isAbstract="false" name="Project Name" xmi.id="S.1">
  <UML:Namespace.ownedElement>
    <MASML:AgentClass name="Agente" namespace="S.1" xmi.id="S.2"/>
    <MASML:OrganizationClass name="Organizacao" namespace="S.1" xmi.id="S.3"/>
    <MASML:AgentRoleClass name="PapelAgente" namespace="S.1" xmi.id="S.4"/>
  </UML:Namespace.ownedElement>
</UML:Model>
<MASML:Diagram name="Class_Diagram" xmi.id="D.1">
  <MASML:GraphElement.semanticModel>
    <MASML:SimpleSemanticModelElement typeInfo="ClassDiagram"/>
  </MASML:GraphElement.semanticModel>
  <MASML:GraphElement.contained>
    <MASML:GraphNode xmi.id="D.10">
      <MASML:GraphElement.semanticModel>
        <MASML:Masml1SemanticModelBridge xmi.id="DD.10">
          <MASML:Masml1SemanticModelBridge.element>
            <MASML:OrganizationClass xmi.idref="S.3"/>
          </MASML:Masml1SemanticModelBridge.element>
        </MASML:Masml1SemanticModelBridge>
      </MASML:GraphElement.semanticModel>
    </MASML:GraphNode>
  </MASML:GraphElement.contained>
</MASML:Diagram>
<MASML:Diagram name="Organization_Diagram" xmi.id="D.2">
  <MASML:GraphElement.semanticModel>
    <MASML:SimpleSemanticModelElement typeInfo="OrganizationDiagram"/>
  </MASML:GraphElement.semanticModel>
  <MASML:GraphElement.contained>
    <MASML:GraphNode xmi.id="D.20">
      <MASML:GraphElement.semanticModel>
        <MASML:Masml1SemanticModelBridge xmi.id="DD.20">
          <MASML:Masml1SemanticModelBridge.element>
            <MASML:AgentClass xmi.idref="S.2"/>
          </MASML:Masml1SemanticModelBridge.element>
        </MASML:Masml1SemanticModelBridge>
      </MASML:GraphElement.semanticModel>
    </MASML:GraphNode>
  </MASML:GraphElement.contained>
</MASML:Diagram>
<MASML:Diagram name="Role_Diagram" xmi.id="D.3">
  <MASML:GraphElement.semanticModel>
    <MASML:SimpleSemanticModelElement typeInfo="RoleDiagram"/>
  </MASML:GraphElement.semanticModel>
  <MASML:GraphElement.contained>
    <MASML:GraphNode xmi.id="D.30">
      </MASML:GraphElement.size>
```

```
      <MASML:GraphElement.semanticModel>
        <MASML:Masml1SemanticModelBridge xmi.id="DD.30">
          <MASML:Masml1SemanticModelBridge.element>
            <MASML:AgentRoleClass xmi.idref="S.4"/>
          </MASML:Masml1SemanticModelBridge.element>
        </MASML:Masml1SemanticModelBridge>
      </MASML:GraphElement.semanticModel>
    </MASML:GraphNode>
  </MASML:GraphElement.contained>
</MASML:Diagram>
```

Quadro 7 – Representação em MAS-ML XMI dos diagramas modelados

Portanto, utilizando as descrições do MAS-ML XMI apresentadas nesta Seção é possível representar todas as abstrações modeladas em MAS-ML, sem qualquer perda de informação.

4.3.2. Segundo passo: Criação do UML XMI

Neste passo o MAS-ML XMI gerado no passo anterior é convertido para o UML XMI através da instanciação do framework ASF. Conforme já mencionado anteriormente, o framework ASF possibilita a implementação de SMA utilizando orientação a objetos, através da implementação de todos os elementos definidos por TAO e representados em MAS-ML. 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.

A transformação do MAS-ML XMI para o UML XMI utiliza o arquivo UML XMI que contém a instanciação do framework. Este arquivo descreve todas as classes e relacionamentos entre as classes definidas no framework. Durante a transformação, o arquivo UML XMI do framework é acrescido 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.

A Tabela 4 apresentada abaixo apresenta as regras de transformações necessárias para a transformação dos elementos representados em MAS-ML em classes OO de acordo com o framework TAO.

Entidade	Regra de Transformação
Agente	1. Criação da classe concreta que representará o agente modelado em MAS-ML e irá extender a classe abstrata *Agent*. 2. Implementação do construtor da classe para criar os objetivos, crenças, planos e ações de acordo com as propriedades modeladas para o agente. A criação dos objetivos e crenças é realizada através da instanciação das classes *Goal* e *Belief*, respectivamente. 3. Criação das classes concretas que estendam as classes *Plan* e *Action* e que implementem o comportamento definido para os planos e ações.

Organização	1. Criação da classe concreta que representará a organização modelada em MAS-ML. Deverá extender a classe abstrata *Organization* ou *MainOrganization*, de acordo com a sua definição. 2. Implementação do construtor da classe para criar os objetivos, crenças, axiomas, planos e ações de acordo com as propriedades modeladas para a organização. A criação dos objetivos, crenças e axiomas é realizada através da instanciação das classes *Goal*, *Belief* e *Axiom*, respectivamente. 3. Criação das classes concretas que estendam as classes *Plan* e *Action* e que implementem o comportamento definido para os planos e ações.
Papel de Agente	1. Criação da classe concreta que representará o papel de agente modelado em MAS-ML e irá extender a classe abstrata *AgentRole*. 2. Implementação do construtor da classe para definição dos direitos e deveres, de acordo com as propriedades modeladas para o papel de agente. A criação dos direitos e deveres é realizada através da instanciação das classes *Right* e *Duty*, respectivamente. 3. Criação de classes concretas para a representação dos protocolos definidos no papel de agente e que irão extender a classe abstrata *Protocol*.
Papel de Objeto	1. Criação da classe concreta que representará o papel de objeto modelado em MAS-ML e irá extender a classe abstrata *ObjectRole*.
Ambiente	1. Criação da classe concreta que representará o ambiente modelado em MAS-ML e irá extender a classe abstrata *Environment*.

Tabela 4 – Regras de transformação de elementos MAS-ML em classes OO, de acordo com o Framework ASF.

Devido ao fato do framework ASF e MAS-ML serem baseados no framework conceitual TAO, as regras necessárias para a transformação dos elementos MAS-ML em classes OO é facilitada, visto que o framework ASF e a linguagem MAS-ML definem os mesmos elementos e relacionamentos.

A Figura 5 ilustra as classes que estão envolvidas na transformação do agente modelado em MAS-ML em classes OO, com base no framework ASF. A classe “Agente” representa o agente presente na Tabela 2. As classes com cor mais escura representam as classes estendidas em função da transformação, ou seja, referentes a instanciação do framework. No Quadro 8 está representado um trecho do UML XMI referente aos elementos ilustrados na Figura 5 e seus relacionamentos com as classes do framework. Os relacionamentos entre classes do framework foram omitidas para não estender demais o documento, como também a estrutura de pacotes utilizadas pelo framework.

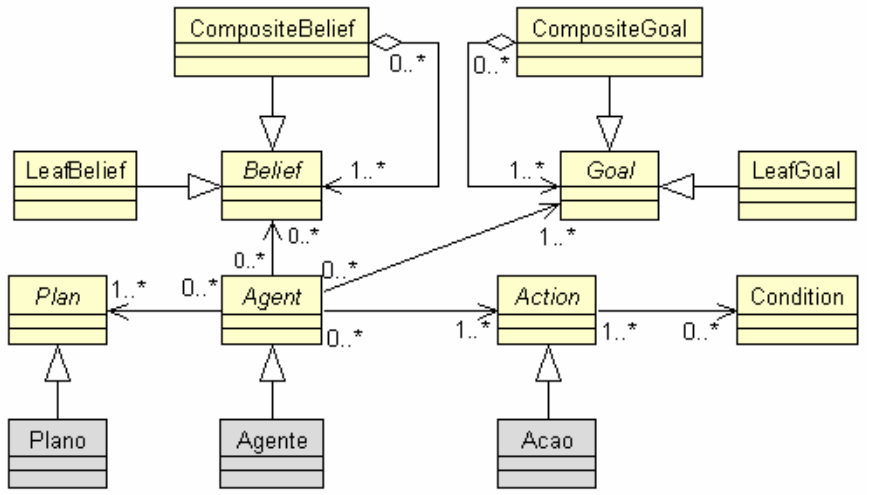
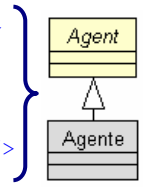
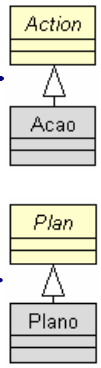


Figura 5 – Transformação de MAS-ML para UML de um agente

```
<UML:Class isAbstract="false" name="Condition" xmi.id="F.4"/>
<UML:Class isAbstract="true" name="Action" xmi.id="F.51"/>
<UML:Class isAbstract="true" name="Belief" xmi.id="F.307"/>
<UML:Class isAbstract="false" name="LeafBelief" xmi.id="F.381"/>
<UML:Class isAbstract="true" name="Goal" xmi.id="F.398"/>
<UML:Class isAbstract="false" name="LeafGoal" xmi.id="F.541"/>
<UML:Class isAbstract="true" name="Plan" xmi.id="F.557"/>
<UML:Class isAbstract="true" name="Agent" xmi.id="F.1271"/>
<UML:Class isAbstract="false" name="Agente" xmi.id="A.2">
  <UML:Classifier.feature>
    <UML:Operation  isAbstract="false" name="Agente" xmi.id="A.3">
     ...
    </UML:Operation>
    <UML:Method xmi.id="A.8">
      <UML:Method.body>
        <UML:ProcedureExpression body="
          LeafGoal goal = new LeafGoal ("tipo","nome","valor_inicial");
          goal.setPlan("Plano");
          LeafBelief belief=new LeafBelief("tipo","nome", "valor_inicial");
          Action action = new Acao();
          action.setPreCondition(new Condition("tipo", "pre_condicao",
"valor_inicial"));
          action.setPosCondition(new Condition("tipo", "pos_condicao",
"valor_inicial"));
          Plan plan = new Plano();…" language="java" xmi.id="A.9"/>
      </UML:Method.body>
      <UML:Method.specification>
        <UML:Operation xmi.idref="A.3"/>
      </UML:Method.specification>
    </UML:Method>
  </UML:Classifier.feature>
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="A.29"/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Class isAbstract="false" name="Acao" xmi.id="A.30">
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="A.35"/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Class isAbstract="false" name="Plano" xmi.id="A.36">
  <UML:GeneralizableElement.generalization>
     <UML:Generalization xmi.idref="A.37"/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Generalization xmi.id="A.29">
  <UML:Generalization.child>
    <UML:Class xmi.idref="A.2"/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref="F.1271"/>
  </UML:Generalization.parent>
</UML:Generalization>
```



```
<UML:Generalization xmi.id="A.35">
  <UML:Generalization.child>
     <UML:Class xmi.idref="A.30"/>
  </UML:Generalization.child>
   <UML:Generalization.parent>
     <UML:Class xmi.idref="F.51"/>
  </UML:Generalization.parent>
</UML:Generalization>
<UML:Generalization xmi.id="A.37">
  <UML:Generalization.child>
     <UML:Class xmi.idref="A.36"/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref="F.557"/>
  </UML:Generalization.parent>
</UML:Generalization>
```



Quadro 8 – Trecho do UML XMI usado para representar um agente

O mapeamento da Organização para o framework ASF é semelhante ao do agente, com exceção de duas diferenças. A classe que representa a organização pode estender a classe abstrata *MainOrganization* ou *Organization*, dependendo de sua definição. Além disso, é necessário criar instâncias da classe *Axiom* no construtor da organização para representar as regras da organização. A Figura 6 apresenta as classes envolvidas na transformação da organização representada na Tabela 2. O UML XMI referente não será apresentado, em virtude do mesmo ser muito parecido ao do agente.

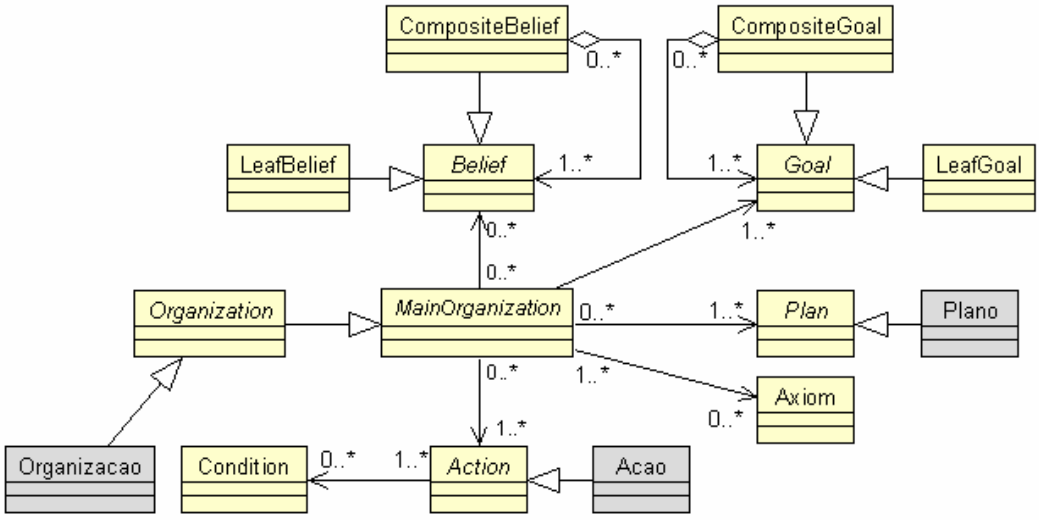


Figura 6 - Transformação de MAS-ML para UML de uma organização

Para a transformação do ambiente, seja ele passivo ou ativo, e do papel de objeto, basta estender as classes abstratas *Environment* e *ObjectRole*, respectivamente. Sendo que caso o ambiente seja ativo, será necessário também instanciar suas propriedades, de forma semelhante como é feita com o agente.

O 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* e *Duty*. Os protocolos são definidos através

da criação de classes concretas que implementem o comportamento definido nos protocolos e que estendem a classe *Protocol*. É importante ressaltar que os relacionamentos criados e representados na modelagem MAS-ML são traduzidos para o modelo UML através de chamadas de métodos dos construtores e implementações dos mesmos.

Pela Figura 7 é possível visualizar as classes envolvidas na transformação do papel de agente da Tabela 2 em classes OO. O UML XMI apresentado no Quadro 9 representa esta transformação, apresentando inclusive as instanciações feitas para representar os direitos, deveres e mensagens. Algumas informações do XMI foram suprimidas para não torná-lo muito extenso.

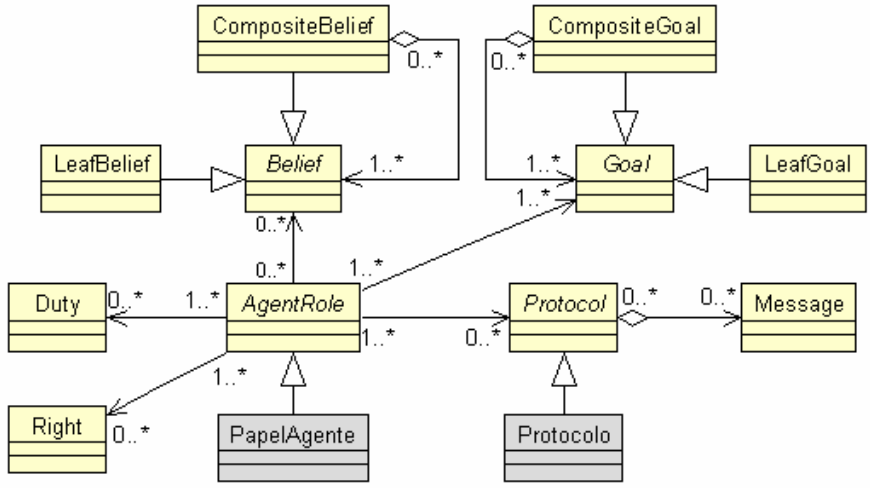
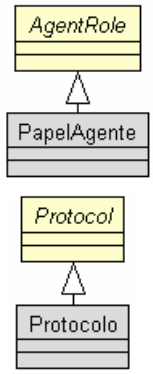


Figura 7 - Transformação de MAS-ML para UML de um papel de agente

```
<UML:Class isAbstract="false" name="Message" xmi.id="F.82"/>
<UML:Class isAbstract="true" name="Belief" xmi.id="F.307"/>
<UML:Class isAbstract="false" name="CompositeBelief" xmi.id="F.347"/>
<UML:Class isAbstract="false" name="LeafBelief" xmi.id="F.381"/>
<UML:Class isAbstract="true" name="Goal" xmi.id="F.398"/>
<UML:Class isAbstract="false" name="CompositeGoal" xmi.id="F.507"/>
<UML:Class isAbstract="false" name="LeafGoal" xmi.id="F.541"/>
<UML:Class isAbstract="true" name="Plan" xmi.id="F.557"/>
<UML:Class isAbstract="false" name="Duty" xmi.id="F.1002"/>
<UML:Class isAbstract="false" name="Right" xmi.id="F.1021"/>
<UML:Class isAbstract="true" name="AgentRole" xmi.id="F.1040"/>
<UML:Class isAbstract="true" name="Protocol" xmi.id="F.1190"/>
<UML:Class isAbstract="false" name="remetente" xmi.id="A.2">...</UML:Class>
<UML:Class isAbstract="false" name="destinatario" xmi.id="A.30">...</UML:Class>
<UML:Class isAbstract="false" name="Protocolo" xmi.id="A.77">
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="A.82"/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Class isAbstract="false" name="PapelAgente" xmi.id="A.58">
  <UML:Classifier.feature>
    <UML:Operation name="AgentRole" xmi.id="A.59">...</UML:Operation>
    <UML:Method xmi.id="A.62">
      <UML:Method.body>
        <UML:ProcedureExpression body="
          LeafGoal objectGoal = new LeafGoal ("tipo","nome","valor_inicial");
          LeafBelief objectBelief = new LeafBelief ("tipo","nome",
"valor_inicial");
          duties = new Vector ();  this.duties.add (new Duty ("acao"));
          rights = new Vector ();   this.rights.add (new Right ("acao2"));
          Protocol objectProtocol = new Protocolo();
```

```
          Message objectMessage = new Message("titulo","conteudo","remetente",
"destinatário");
           objectProtocol.setMessage (objectMessage); " language="java"
xmi.id="A.63"/>
      </UML:Method.body>
      <UML:Method.specification>
        <UML:Operation xmi.idref="A.59"/>
      </UML:Method.specification>
    </UML:Method>
  </UML:Classifier.feature>
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref="A.64"/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Generalization xmi.id="A.64">
  <UML:Generalization.child>
    <UML:Class xmi.idref="A.58"/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref="F.1040"/>
  </UML:Generalization.parent>
</UML:Generalization>
<UML:Generalization xmi.id="A.82">
  <UML:Generalization.child>
    <UML:Class xmi.idref="A.77"/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref="F.1190"/>
  </UML:Generalization.parent>
</UML:Generalization>
```



Quadro 9 - Trecho do UML XMI usado para representar um papel de agente

Uma vez realizada as transformações das abstrações de MAS-ML em classes OO e gerado o arquivo UML XMI referente à aplicação, temos no formato XMI a representação da aplicação. O arquivo XMI representa em UML um modelo específico de plataforma (PSM), baseado no framework ASF.

4.3.3. Classificação da transformação modelo-em-modelo

A transformação modelo-em-modelo detalhada nas Sub-Seções 4.3.1 e 4.3.2 pode ser classificada como uma abordagem baseada em transformação gráfica [52]. Este tipo de abordagem utiliza regras de transformação gráficas (padrões de transformação gráficos) calcados numa sintaxe concreta das respectivas linguagens originais e nas que serão convertidas. Esta transformação entre modelos utiliza atributos, gráficos intitulados, que é um tipo gráfico especificamente desenhado para representar modelos UML.

Em [52], é apresentado um diagrama baseado em *features* que representa as possíveis características de uma abordagem de transformação entre modelos. Neste diagrama temos um conjunto de oito *features* que representam os principais pontos de variação. A seguir descrevemos o conjunto de *features* e como estas *features* estão relacionadas como nossa proposta:

- Regras de transformação – é composta por duas partes, sendo uma direcionada ao modelo original e outra ao modelo resultante. Ambas as partes

podem ser representadas utilizando uma composição dos seguintes conceitos: variáveis, padrões (fragmentos de zero ou mais variáveis) e lógica (expressa cálculos e regras dos elementos presentes no modelo). Outros aspectos das regras de transformação são: separação sintática (ocorre quando a sintaxe original é diferente da final), bidirecionalidade (quando a regra pode ser aplicada nas duas direções), parametrização da regra (as regras possuem parâmetros de controle adicionais) e estruturas intermediárias (geração de modelos intermediários).

Na abordagem proposta, as regras (ou padrões) estão representadas de forma textual. As regras transformam o MAS-ML XMI (entrada) e geram o UML XMI (saída). Essas regras são separadas sintaticamente, não estão parametrizadas e não produzem estruturas intermediárias.

- Aplicação do escopo das regras – permite à transformação restringir partes de um modelo que participam na transformação. Neste trabalho, cada regra é aplicada em um escopo específico. Em nossa transformação existem 14 escopos diferentes, correspondendo às 10 abstrações que podem ser representadas em modelos MAS-ML. Aplicando essas regras, o modelo original é transformado em um modelo final.

- Relacionamento entre o Original e o Final – algumas abordagens pregam a criação de um novo modelo final (resultante), que deve ser separado do original. Ainda assim, temos outras abordagens que permitem que o modelo final seja um novo modelo ou um já existente. Em nossa abordagem, parte do modelo final já existe e é estendido através da aplicação das regras de transformações ao modelo original. Conseqüentemente, o modelo final, que representa o framework ASF, é estendido de acordo com as características da aplicação presentes no modelo original.

- Estratégia para aplicação da regra – a estratégia de uma regra determina a sua aplicação quanto tem-se diferentes formas de aplicação em um determinado escopo. Um estratégia pode ser determinística, não-determinística ou até mesmo interativa. Em nossa proposta, cada regra é aplicada em partes específicas do modelo original explicitamente determinadas pelo escopo da regra.

- Programação das regras – mecanismos de programação determinam a ordem de utilização de cada regra. Em nossa transformação o usuário não controla a programação das regras. A programação é pré-definida pela ferramenta. A ferramenta utiliza as regras quando necessário, ou seja, quando o tipo de abstração associada à regra é representada no modelo original. Se uma

determinada abstração não está presente no modelo original, a regra de transformação relacionada a esta abstração não é utilizada. Por outro lado, se um determinado tipo de abstração aparece várias vezes no modelo, a regra é utilizada várias vezes.

- Organização de regras – está relacionada com a composição e estruturação de diversas regras de transformação. Podem ser usados mecanismos de modularização (empacotamento de regras em módulos), mecanismos de reutilização e uma estrutura organizacional para organizar a execução das regras. Na transformação apresentada nesta Seção, as regras estão organizadas de acordo com o modelo de entrada, isto é, são chamadas de acordo com as abstrações existentes no modelo original.

- Rastreamento – transformações podem registrar os *logs* dos mapeamentos executados entre os elementos iniciais e os finais. Estes logs podem ser úteis na análise de impacto de performance, sincronização entre modelos, depuração baseada em modelos. A ferramenta proposta neste trabalho não registra os *logs* dos mapeamentos e não apresenta nenhum mecanismo de rastreamento.

- Direcionalidade – transformações podem ser classificadas como unidirecional ou bidirecional. Transformações unidirecionais só podem ser executadas em uma direção, onde em cada caso um modelo final é computado (ou atualizado) baseado em um modelo original. Transformações bidirecionais podem ser executadas em ambas as direções, isto é, a partir de um modelo final é possível re-gerar o modelo original. Atualmente, a transformação proposta neste trabalho é unidirecional, ou seja, as regras são executadas em somente uma direção. As regras criadas transformam o MAS-ML XMI em UML XMI, porém o inverso não é realizado.

4.4. Modelo PSM

O UML XMI gerado no segundo passo da etapa anterior reflete o modelo específico de plataforma da aplicação (PSM). Utilizando-se de ferramentas que tenham a funcionalidade de importar arquivos UML 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 [53], Rational Rose [39] e Poseidon [54].

O modelo UML gerado nesta etapa é uma diagrama de classes UML que contém as classes do framework ASF e as classes correspondentes a aplicação que instancia o framework. Todas as entidades, propriedades e os relacionamentos entre as entidades modeladas nos três diagramas estruturais 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 figura abaixo (Figura 8) estão representadas todas as classes do framework e a instanciação de algumas classes (cor escura) de acordo com os exemplos dados durante esta seção.

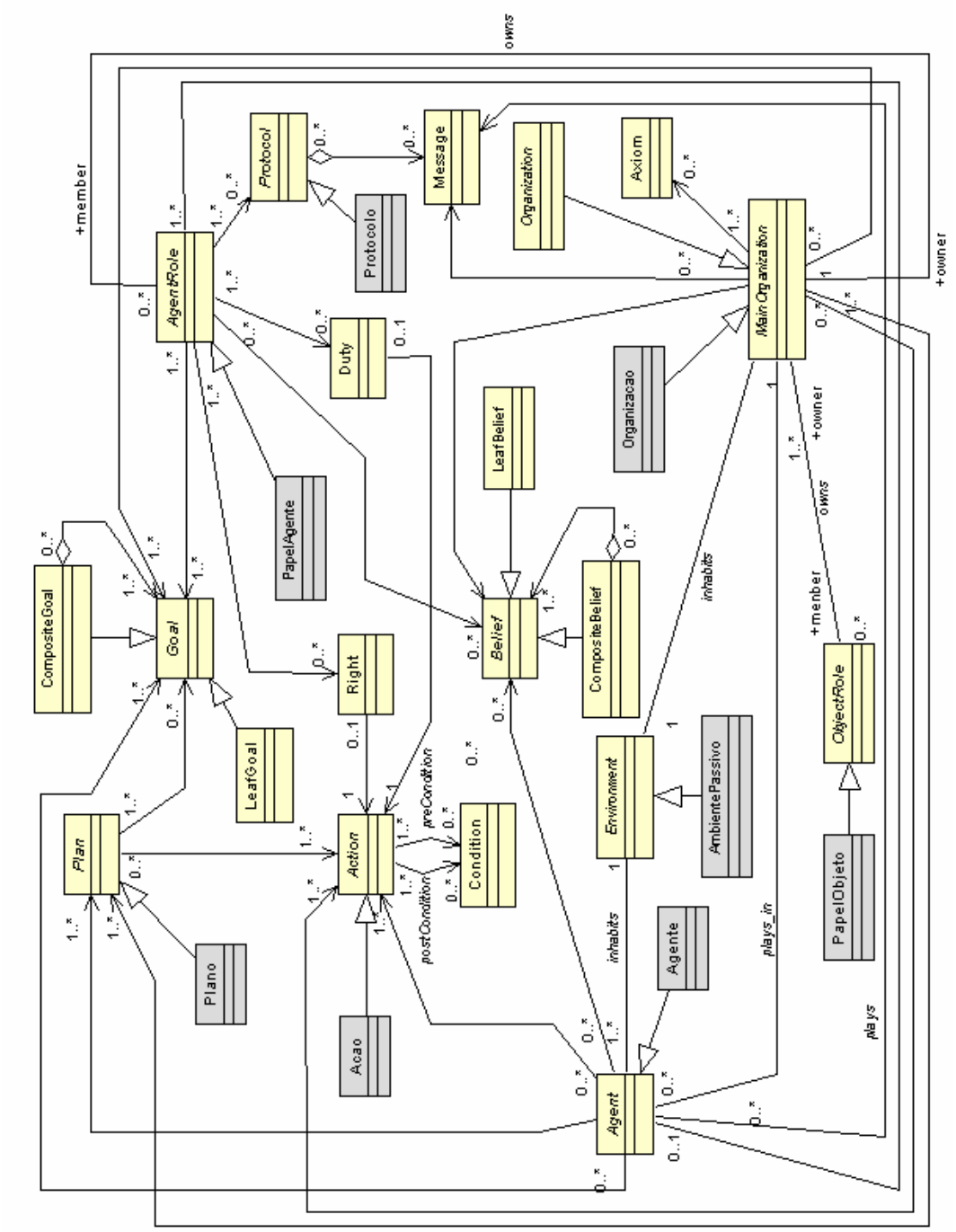


Figura 8 – Modelo UML com instanciações geradas com a transformação

4.5. Transformação do modelo 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 à última etapa de MDA que transforma o modelo PSM em código. A geração de código a partir de modelos UML já é provida por ferramentas que importam o XMI e geram modelos UML. Portanto, o

mapeamento entre os modelos UML descritos em XMI e o código fonte dependerá da interpretação utilizada pela ferramenta empregada. A linguagem utilizada para representar o código gerado a partir do UML XMI dependerá também da ferramenta utilizada.

Na Seção 6 deste trabalho é apresentada uma ferramenta desenvolvida para dar apoio ao processo de desenvolvimento proposto e que utiliza o framework AndroMDA [55] para realizar este mapeamento. Esta ferramenta é capaz de gerar código na linguagem Java. A seguir estão descritos os códigos gerados por esta ferramenta relativo a classe *AgentRole* (Quadro 10) pertencente ao framework e a classe "PapelAgente" (Quadro 11), que é uma instanciação da classe *AgentRole* e foi gerada de acordo com o trecho de UML XMI apresentado no Quadro 9.

```
public abstract class AgentRole{
   protected Agent agentPlayingRole = null;
   protected Organization organizationPlayingRole = null;
   protected MainOrganization owner = null;
   protected Vector goals = new Vector();
   protected Vector beliefs = new Vector();
   protected Vector duties = new Vector();
   protected Vector rights = new Vector();
   protected Vector protocols = new Vector();
   private boolean threadStopped = false;
   private boolean threadSuspended = false;
   public Agent getAgentPlayingRole(  ){
      return this.agentPlayingRole;
   }
   public MainOrganization getOrganizationPlayingRole(  ){
      return this.organizationPlayingRole;
   }
   public String getRoleName(  ){
      return this.name;
   }
   public MainOrganization getOwner(  ){
      return this.owner;
   }
   public Vector getGoals(  ){
      return this.goals;
   }…
   public void setAgent( Agent newAgent ){
      this.agentPlayingRole = newAgent;
   }
   public void setOrganization( Organization newOrganization ){
      this.organizationPlayingRole = newOrganization;
   }
   public void setRoleName( String name ){
      this.name = name;
   }
   public void setOwner( MainOrganization newOwner ){
      this.owner = newOwner;
   }
   public void setGoal( Goal newGoal ){
      goals.add (newGoal);
   }…
   public boolean threadStopped(  ){
      return threadStopped;
   }
   public void stopThread(  ){
      threadStopped = true;
   }
   public boolean threadSuspended(  ){
      return threadSuspended;
   }
   public void supendThread(  ){
```

```
      threadSuspended = true;
   }
   public void resumeThread(  ){
      threadSuspended = false;
   }
   public void destroy(  ){
      setAgent(null);
      setOrganization(null);
      MainOrganization organization = getOwner();
      Vector vRoles = organization.getAgentRoles();
      Enumeration enumvRoles = vRoles.elements();
      while (enumvRoles.hasMoreElements()){
         AgentRole roleAux = (AgentRole)enumvRoles.nextElement();
         if (roleAux == this)
            vRoles.remove(roleAux);
      }
   }
}
```

Quadro 10 – Código gerado a partir do UML XMI referente a classe AgentRole do framework ASF

```
public class PapelAgente extends AgentRole{
   public PapelAgente( MainOrganization owner ){
      super.setOwner( owner );
      owner.setAgentRole( this );
      LeafGoal objectGoal = null;
      goals = new Vector ();
      objectGoal = new LeafGoal ("tipo","nome","valor_inicial");
      this.goals.add (objectGoal);
      LeafBelief objectBelief = null;
      beliefs = new Vector ();
      objectBelief = new LeafBelief ("tipo","nome", "valor_inicial");
      this.beliefs.add (objectBelief);
      duties = new Vector ();
      rights = new Vector ();
      this.duties.add (new Duty ("acao"));
      this.rights.add (new Right ("acao2"));
      Message objectMessage = null;
      Protocol objectProtocol = null;
      protocols = new Vector ();
      objectProtocol = new application.nome_protocolo();
      this.protocols.add (objectProtocol);
      objectMessage      =      new      Message("titulo","conteudo","remetente",
"destinatario");
      objectProtocol.setMessage (objectMessage);
   }
}
```

Quadro 11 – Código gerado a partir do UML XMI referente a classe Papel Agente da aplicação

A transformação apresentada nesta seção pode ser classificada como uma abordagem baseada em templates, segundo [52]. Ferramentas que geram código a partir de arquivos UML XMI usam transformações baseadas em templates. A ferramenta AndroMDA [55], que utilizamos para a geração de código como apresentado na Seção 6, é um exemplos deste tipo de ferramenta.