Estendendo o Framework JAMDER para Suporte à Implementação de Sistemas Multi-Agente Normativos

Robert M. R. Júnior, Emmanuel S. S. Freire, Mariela I. Cortés

Grupo de Engenharia e Sistemas Inteligentes (GESSI)
Departamento de Computação – Universidade Estadual do Ceará (UECE)
Avenida Paranjana, 1700 – 60740-000 – Fortaleza – CE – Brazil

{robstermarinho, savio.essf}@gmail.com, mariela@larces.uece.br

Abstract. *In complex systems development is evident a significant difficulty resulting by semantic gap, which is characterized by semantic and conceptual distinction between two descriptions generated and their representations. This difference has a negative impact on the developer productivity and probably, the quality of the generated code. This work aims to bring the concepts of modeling and implementation, especially for normative multi-agent systems, through the extension of JAMDER framework, including the norm concepts and its properties. The main benefit of the proposed mapping is to reduce the semantic gap between the modeling project and its implementation. To illustrate the result of this approach a case study is presented that involves a normative multi-agent system to paper submission.*

Resumo. *No desenvolvimento de sistemas complexos é notória uma dificuldade significativa, resultante do gap semântico, caracterizado pela distinção conceitual e semântica entre duas descrições e suas respectivas representações. Esta diferença traz um impacto negativo na produtividade do desenvolvedor e possivelmente na qualidade do código gerado. Este trabalho tem como objetivo aproximar os conceitos de modelagem e implementação, especificamente para sistemas multi-agente normativos, através da extensão do framework JAMDER. O principal benefício do mapeamento proposto é reduzir o gap semântico entre a modelagem do projeto e sua implementação. Para ilustrar o resultado da proposta é apresentado um estudo de caso que envolve um sistema multi-agente normativo para submissão de artigos.*

1. Introdução

Os sistemas centrados em agente vêm sendo amplamente utilizados pela comunidade científica para o desenvolvimento de sistemas computacionais complexos [Casillo, 2008]. Entretanto, a implementação desses sistemas requer um esforço de engenharia no intuito de dar suporte adequado às diversas atividades de desenvolvimento.

Segundo Russell e Norvig (2004), um agente de software é uma entidade capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores. No caso de vários agentes cooperando ou disputando entre si, inseridos em um mesmo ambiente e trocando informações, chamamos esse sistema de multi-agente (SMAs). Dentro desse contexto, um SMA pode ser governado por normas

que regulam o comportamento das entidades inseridas nesse sistema. Com isso, aumenta-se mais a complexidade para o seu desenvolvimento [Freire et al., 2012].

Um fator significativo para a dificuldade no desenvolvimento de sistemas complexos é a diferença conceitual e semântica entre a fase de detalhamento do projeto ou domínio do problema, que representa a documentação ao longo de um conjunto de modelos, e a fase de implementação/solução, que tem por objetivo a codificação do sistema. Assim, é necessário um conjunto de métodos e ferramentas adequado, no intuito de dar suporte às atividades de desenvolvimento, que visam reduzir o *gap* semântico entre estes dois níveis de abstração: artefatos de modelagem e código.

Considerando como requisito a existência de um metamodelo que contemple as entidades integrantes de um SMA normativo, a linguagem de modelagem NorMAS-ML (*Normative Multi-Agent System Modeling Language*) [Freire et al., 2012] dá suporte à modelagem de agente, papel de agente, papel de objeto, objeto, organização e ambiente, juntamente com os elementos estáticos que compõem uma norma. Diagramas da linguagem podem ser gerados através da ferramenta MAS-ML *Tool* [Freire, Rocha Junior e Cortés, 2012].

No contexto de implementação, o *framework* JAMDER (JADE to MAS-ML 2.0 *Development Resource*) [Lopes et al., 2012] representa uma extensão do *framework* JADE (*Java Agent Development*) com as seguintes características: (i) possui uma plataforma na linguagem Java, (ii) dá suporte a sistemas distribuídos, (iii) é gratuito e (iv) permite a implementação dos elementos típicos de um SMA. No entanto, JAMDER não permite a implementação de todos os elementos normativos.

O objetivo desse artigo consiste na extensão do *framework* JAMDER no intuito de fornecer a infraestrutura adequada para a implementação de sistemas multi-agente normativos de acordo com as definições contempladas em NorMAS-ML. Este trabalho está organizado da seguinte maneira: a Seção 2 apresenta alguns trabalhos relacionados. A Seção 3 corresponde ao referencial teórico, no qual é apresentada a linguagem NorMAS-ML e o *framework* JAMDER. Em seguida, na Seção 4, é descrita a extensão do *framework* JAMDER, proposta para implementar SMAs com normas. Um estudo de caso referente à extensão é apresentado na Seção 5 e, finalmente, as conclusões e trabalhos futuros são descritos na Seção 6.

2. Trabalhos Relacionados

Um conjunto de *frameworks* e plataformas tem sido desenvolvido para dar apoio ao desenvolvimento de SMAs. De forma geral, estes mecanismos estão associados a uma linguagem de programação para compor as entidades e fornecem um ambiente para a execução dos mesmos. A seguir são apresentados alguns dos mais utilizados *frameworks* de implementação para SMA.

JACK [Jack, 2012] é um *framework* em Java para o desenvolvimento de SMAs. Esse *framework* oferece alto desempenho, e uma maneira fácil de ser estendido para dar suporte a agentes BDI (*Belief-Desire-Intention*) e requisitos específicos das aplicações. A linguagem utilizada pelo JACK (JACK *Agent Language*) é construída a partir da linguagem Java podendo ser usada no desenvolvimento de agentes BDI e o seu comportamento [Nunes, 2007]. Entretanto, JACK não suporta a modelagem dos conceitos normativos e sua ferramenta IDE (*Integrated Development Environment*) não é gratuita.

JAM [Huber, 2012] foi desenvolvido com base em uma série de teorias sobre agentes e *frameworks* para agentes baseadas em BDI. Permite a implementação de agentes que possuem planos e objetivos. Entretanto, JAM não possui ferramenta de desenvolvimento, não permite a implementação de papel de agente, organização nem dos conceitos normativos.

JADE [Jade, 2012] é um *framework* implementado na linguagem Java, simplificando o desenvolvimento de SMAs através do *middleware* que está de acordo com as especificações FIPA (*Foundations of Intelligent Physical Agents)* e com um conjunto de ferramentas gráficas que suportam *debugging* [NUNES, 2007]. Este *framework* permite a implementação dos elementos típicos que compõem os SMAs. Entretanto, não possui suporte aos conceitos normativos.

Como JADE não suportava a implementação das diferentes arquiteturas de agente, Salmito et al. (2012) propôs sua extensão, denominada JAMDER. Toda a extensão foi baseada na linguagem de modelagem MAS-ML 2.0 [Gonçalves, 2009]. Com isso, JAMDER além de suportar a modelagem dos elementos típicos dos SMAs, passou a suportar os diferentes tipos de agente. Entretanto, JAMDER possui suporte limitado aos conceitos normativos. Com ele, é possível implementar agentes que possuem seu comportamento regulamentado por regras incluídas em papéis de agente, organizações e sub-organizações.

Jason normativo [Santos Neto, 2012] é uma extensão do *framework* Jason [Jason, 2012] baseada na linguagem *AgentSpeak* Normativo (extensão da linguagem *AgentSpeak*). Este *framework* possibilita a implementação de agentes normativos capazes de entender, seguir ou violar as normas contidas no ambiente. Entretanto, a linguagem é definida em linguagem de primeira ordem, o foco das normas é no comportamento dos agentes e não das outras entidades que compõem um SMA normativo, e não possui suporte a troca de papéis entre agentes.

A partir da análise apresentada e considerando a necessidade de um *framework* que possibilite a implementação de normas juntamente com as entidades previstas em um SMA, destacamos JAMDER por (i) possuir uma plataforma e uma linguagem JAVA, (ii) dar suporte a sistemas distribuídos, (iii) estar de acordo com os padrões FIPA, (iv) dar suporte às entidades típicas dos SMAs e (v) possuir interface gráfica e *plugins* para IDEs.

3. Referencial Teórico

3.1. NorMAS-ML

NorMAS-ML (*Normative Multi-Agent System Modeling Language*) [Freire et al., 2012] é uma extensão da linguagem de modelagem MAS-ML [Silva, 2004] que incorpora os conceitos de normas, de modo que as entidades em SMA possam ser modeladas considerando os seguintes elementos estáticos normativos definidos por Figueiredo e Silva (2010): (i) conceitos deônticos (obrigações, permissões e proibições); (ii) entidades envolvidas cujo comportamento pode ser regulado pelas normas; (iii) ações; (iv) restrições de ativação para as normas; (v) sanções que envolvem recompensas e punições no cumprimento e violação de uma norma; e (vi) o contexto que determina a área de aplicação da norma. Com isso, a linguagem possibilita representar normas que regulam tanto o comportamento das entidades como suas propriedades estruturais. Como, por exemplo, as ações a serem executadas por um agente.

O processo de extensão possibilitou a inclusão dos elementos normativos nas sintaxes abstrata e concreta de MAS-ML. Na sintaxe abstrata, foram definidas novas metaclasses e estereótipos. Na sintaxe concreta, as representações gráficas das entidades Organização e Papel de Agente foram alteradas devido à incorporação do conceito de norma. Adicionalmente, foram definidos novos elementos gráficos para representar as novas metaclasses e relacionamentos inseridos na sintaxe abstrata. NorMAS-ML apresenta um novo diagrama estático para a modelagem de normas e suas propriedades e reaproveita todos os diagramas definidos em MAS-ML.

Para a modelagem do diagrama de normas, Freire, Rocha Junior e Cortés (2012) evoluíram a ferramenta MAS-ML *tool*, que é um *plug-in* da plataforma Eclipse [Eclipse, 2012]. Isso implica que os usuários podem trabalhar com modelagem de SMAs ao mesmo tempo em que fazem uso dos recursos oferecidos pela plataforma Eclipse. Com esta ferramenta é possível à modelagem de todos os diagramas estáticos definidos em NorMAS-ML.

3.2. JAMDER

JAMDER (JADE to MAS-ML 2.0 *Development Resource*) [Lopes et al., 2012] é uma extensão do *framework* JADE [Jade, 2012] que incorpora novas classes a fim de implementar informações específicas presentes na modelagem de atributos e métodos. JAMDER utiliza todos os recursos da plataforma JADE para incorporar os novos tipos de agente definidos em MAS-ML 2.0 [Gonçalves et al., 2009].

Neste contexto, este *framework* contempla o mapeamento para identificar quais elementos do metamodelo conceitual de MAS-ML 2.0 correspondem às entidades de primeira classe na plataforma de implementação. Consequentemente, três hierarquias de agentes em nível de modelagem foram definidas em JAMDER, a saber: agentes com plano, agentes reflexivos (reativo simples e reativo baseado em conhecimento) e agentes cognitivos (baseado em objetivo e baseado em utilidade).

Além do conceito de agente, foram definidas representações de outras entidades que compõem os SMAs. Com isso, as representações específicas para a implementação de papel de agente, organização, ambiente, objeto e papel de objeto foram incluídas em JAMDER.

4. Extensão do Framework JAMDER e Mapeamento com NorMAS-ML

Esta seção apresenta a extensão do *framework* JAMDER baseada nos conceitos normativos apresentados em NorMAS-ML. A extensão é denominada JAMDER 2.0 (Figura 1).

4.1. Inclusão dos elementos normativos

O processo de extensão caracterizou-se pelo mapeamento entre os conceitos de modelagem e de implementação com o objetivo de identificar quais elementos do metamodelo conceitual referente às entidades de NorMAS-ML existiam em JAMDER. Com isso, foi identificada a necessidade de criar classes para representar os conceitos normativos. A Figura 1 apresenta o diagrama das classes que representa os conceitos normativos incluídos em JAMDER.

4.1.1. Norm

Em NorMAS-ML, uma norma é utilizada para restringir o comportamento de agentes, organizações e suborganizações durante um período de tempo, e definir as sanções

aplicadas quando violadas ou cumpridas [Figueiredo e Silva, 2010]. Para representar a entidade norma foi criada a classe *jamder.norms.Norm*. O construtor desta classe define os seguintes argumentos em ordem: (i) o identificador; (ii) o tipo da norma; (iii) a entidade restringida; (iv) o contexto; (v) a ação e (vi) a lista de restrições de ativação.

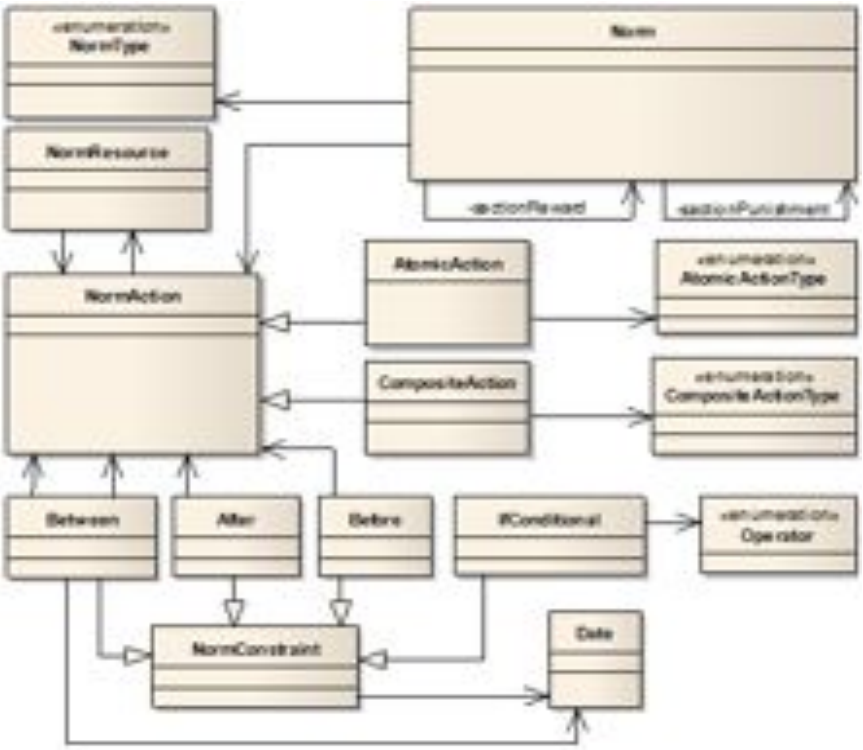

Figura 1. O diagrama de classes representando os conceitos normativos em JAMDER.

Algumas propriedades principais como *normType* definem o conceito deôntico da norma através de um *enumeration* do tipo booleano com os seguintes status: PERMISION, PROHIBITION e OBLIGATION. Adicionalmente, a norma possui atributos que definem as entidades envolvidas ou restringidas: *restrictOrganization*, *restrictAgentClass*, *restrictEnvironmentClass*, *restrictAgentRoleClass;* e atributos que definem o contexto ou domínio de aplicação: *contextOrganizationClass*, *contextEnvironmentClass*. Além disso, ela define o atributo *action* referente à ação vinculada a ela.

4.1.2. NormResource

Uma norma é definida para restringir o comportamento de determinado recurso [Freire et al., 2012]. Em NorMAS-ML, um recurso pode estar vinculado a: (i) características estruturais (objetivos, crenças, atributos), (ii) características comportamentais (métodos, ações, planos e protocolos), (iii) uma entidade (objeto, agente, organização, papel de agente ou ambiente), (iv) um relacionamento ou (v) uma mensagem.

Para contemplar cada um desses casos, o recurso é representado pela classe *jamder.norms.NormResource* com as seguintes propriedades detalhadas a seguir:

- ***property***: define como recurso um atributo, objetivo, crença ou relacionamento. Essa entidade é uma instância da classe *jamder.structural.Property* originária do JAMDER;

- ***method, action, plan, protocol***: esses atributos são utilizados quando se define como recurso uma das características comportamentais, respectivamente, um método, uma ação, um plano ou um protocolo. Cada uma delas são instâncias de classes originárias do JAMDER no *package jamder.behavioral*;
- ***object, agent, environment, agentRole***: atributos criados para definir um recurso caso ele seja uma entidade;
- ***message***: utilizado para definir um recurso do tipo mensagem.

Um recurso possui o construtor com apenas um parâmetro que preencherá um dos atributos listados, representado dessa forma o tipo do recurso.

4.1.3. NormAction

Segundo Freire et al. (2012), uma norma é definida para restringir a execução das entidades ou acesso aos recursos do sistema, por meio de um conjunto de ações vinculadas a ela, que controlam estes recursos. Estas ações são divididas em dois tipos em NorMAS-ML: *AtomicAction* e *CompositeAction*.

A metaclasse *AtomicAction* tem como objetivo mapear diretamente as operações do sistema modelado (excluir, atualizar, ler, criar e executar), enquanto a metaclasse *CompositeAction* mapeia as operações das entidades do sistema e mantém a hierarquia tanto de *AtomicActions* como *CompositeActions*.

As ações foram representadas pela classe principal, *jamder.norms.NormAction.* Esta classe principal possui como atributo a entidade *normResource* que define o recurso controlado pela ação. Este recurso é recebido como parâmetro pelo construtor de *NormAction*. Adicionalmente, essa classe possui uma lista de normas as quais a ação está vinculada e duas subclasses: *jamder.norms.AtomicAction* e *jamder.norms.CompositeAction*, as quais representam o tipo da ação.

4.1.4. NormConstraint

Em NorMAS-ML, a metaclasse *NormConstraint* é responsável por definir o período de restrição de uma norma, juntamente com as suas especializações: (i) *Before*, indica que uma norma está ativa antes da execução da ação e/ou da realização a partir de uma determinada data; (ii) *After*, indica que uma norma está ativa depois da execução da ação e/ou da realização a partir de uma determinada data; (iii) *Between*, indica que uma norma está ativa entre a execução de duas determinadas ações e/ou da realização entre duas datas definidas; (iv) *If*, ativa a norma de acordo com a comparação entre dois operandos ou por data. Um operando pode ser um objetivo, uma crença, um atributo ou um valor.

A classe *jamder.norms.NormConstraint* foi criada para a representação das restrições de ativação com suas respectivas subclasses que contemplam cada tipo de restrição: *jamder.norms.Before*, *jamder.norms.After, jamder.norms.Between e jamder.norms.IfConditiona*l. Ela está diretamente ligada à classe *jamder.norms.Date* responsável por definir uma data.

As classes *jamder.norms.Before* e *jamder.norms.After* possibilitam que seja definida uma ação ou uma data para ativar a restrição de ativação. A classe

jamder.norms.Between difere das anteriores pois possibilita que sejam definidas duas ações ou duas datas.

Por fim, a classe *jamder.norms.IfConditional* possui os atributos *firstProperty* e *secondProperty*, as instâncias da classe *jamder.structural.property,* que definem dois operandos, e o atributo *operator*, que define o operador. Para contemplar os operadores foi criada a *classe jamder.norms.Operator* como um *enumeration* do tipo booleano que define os seguintes operadores: LESS_THAN, GREATER_THAN, EQUAL_TO, LESS_OR_EQUAL_TO, GREATER_OR_EQUAL_TO. Consequentemente, a classe *jamder.norms.IfConditional* possibilita que seja definida uma restrição de ativação que possua um operador e uma data ou possua um operador e dois operandos.

4.2. Modificações das classes do JAMDER

Algumas classes existentes em JAMDER sofreram modificações para contemplar as propriedades definidas segundo a linguagem NormMAS-ML. A seguir a descrição dessas modificações:

- A lista de normas *contextNorms* foi adicionada às classes *jamder.Environment* e *jamder.Organization*. Ela representa as normas cujo ambiente ou organização é definido como contexto.
- A lista de normas *restrictNorms* foi adicionada às classes *jamder.Environment*, *jamder.Organization*, *jamder.agents.GenericAgent* e *jamder.roles.AgentRole*. Ela representa as normas cuja instância da classe é a entidade restringida.
- Os conceitos de direito (*right*) e dever (*duty*) são usados em JAMDER para definir as ações que podem e devem ser executadas por agentes. Entretanto, esses conceitos foram eliminados em NorMAS-ML pelo fato de serem considerados semanticamente equivalentes aos conceitos deônticos de permissão e obrigação das normas. Portanto, foi necessário retirar os conceitos de direito e dever presentes na classe *jamder.roles.AgentRole.*
- Os axiomas (*axiom*) caracterizam um conjunto de leis e regras que os agentes e suborganizações vinculadas a uma determinada organização devem obedecer (Silva, 2004). Entretanto, estes conceitos foram eliminados em NorMAS-ML, pois uma organização ou suborganização pode ser associada a uma norma ou a um conjunto de normas que por sua vez restringem o comportamento dos agentes ou das suborganizações que executam papéis nesta organização ou suborganização. Portanto, foi retirada a lista de axiomas da classe *jamder.Organization*.
- Finalmente, as seguintes classes que representavam as instâncias de *duty*, *right* e *axiom*, respectivamente, *jamder.behavioural.Duty, jamder.behavioural.Right e jamder.structural.Axiom* foram removidas juntamente com a classe *jamder.behavioural.DutyRightProperty*.

Com essas modificações nas classes existentes previamente em JAMDER e a criação de novas classes (definidas na subseção anterior), o *framework* JAMDER 2.0 possibilita a modelagem das propriedades e relacionamentos das entidades que compõem um sistema multi-agente normativo. O código completo de JAMDER 2.0 encontra-se em: https://sites.google.com/site/uecegessi/estudo-de-caso/jamder-2-0.

5. Estudo de Caso

Com o objetivo de ilustrar a utilização do *framework* JAMDER 2.0, foi utilizado o Sistema de Gestão de Submissão de Artigos. Este estudo de caso foi utilizado por Freire, Rocha Junior e Cortés (2012) para ilustrar a modelagem por meio do diagrama de normas da ferramenta MAS-ML *tool*. Este estudo de caso aborda a criação de um SMA normativo responsável pela gestão de submissões, no qual são identificados os agentes, seus papéis, a organização, o ambiente e as normas, através da modelagem pela ferramenta MAS-ML *tool*. Após a modelagem ser estabelecida, serão apresentadas as classes JAMDER 2.0 que representam essas entidades.

Os sistemas de gestão de submissão de artigos são utilizados para a seleção dos artigos que serão publicados em um evento científico. Para isso, os autores devem submeter seus artigos até uma data determinada (*deadline*), a partir da qual, os avaliadores (pelo menos três) iniciam o processo de revisão. Após o término do período de revisão, os organizadores do evento devem divulgar os resultados.

Os autores podem: (i) submeter seus artigos (completos ou curtos) até a data de submissão, (ii) visualizar a situação do seu artigo, (iii) visualizar revisões dos avaliadores. Os avaliadores podem: (i) submeter artigos, (ii) avaliar os artigos que foram indicados pelos organizadores do evento. Os organizadores podem (i) estender o período de submissão de artigos, (ii) escolher os avaliadores, (iii) divulgar o resultado das revisões.

5.1. Identificação das Entidades do Sistema de Gestão de Submissão de Artigos

No ambiente *Conference Management*, é possível identificar a organização principal *Conference* e o tipo de agente: *user agent*, que pode exercer os papéis: *author*, *speaker*, *organizer*, *conference chair*, *website manager* e *reviewer*. Estes papéis são definidos pela organização principal, juntamente com o papel de objeto *submitted*. As instâncias desse papel são exercidas pelas instâncias da classe *Paper*, que possui duas subclasses *ShortPaper* e *FullPaper*.

5.2. Definição das Normas para o Sistema de Gestão de Submissão de Artigos

Para o sistema de gestão de submissão de artigos[1] foram definidas as seguintes normas:

- N1: Os organizadores estão proibidos de submeter artigos.
- N2: Os revisores estão permitidos de submeter artigos.
- N3: Os revisores estão proibidos de revisar seus próprios artigos.
- N4 (Punição para a violação da N3): Os revisores que violarem N4 devem ter seu papel cancelado.
- N5 (Punição para a violação da N3): O presidente da conferência é obrigado a descartar o artigo.

[1] O estudo de caso está sendo parcialmente apresentado devido à limitação do número de páginas. Entretanto, a sua versão completa pode ser encontrada em: https://sites.google.com/site/uecegessi/estudo-de-caso/jamder-2-0.

A Figura 2 apresenta a modelagem das normas N3, N4 e N5 por meio do diagrama de normas de NorMAS-ML.

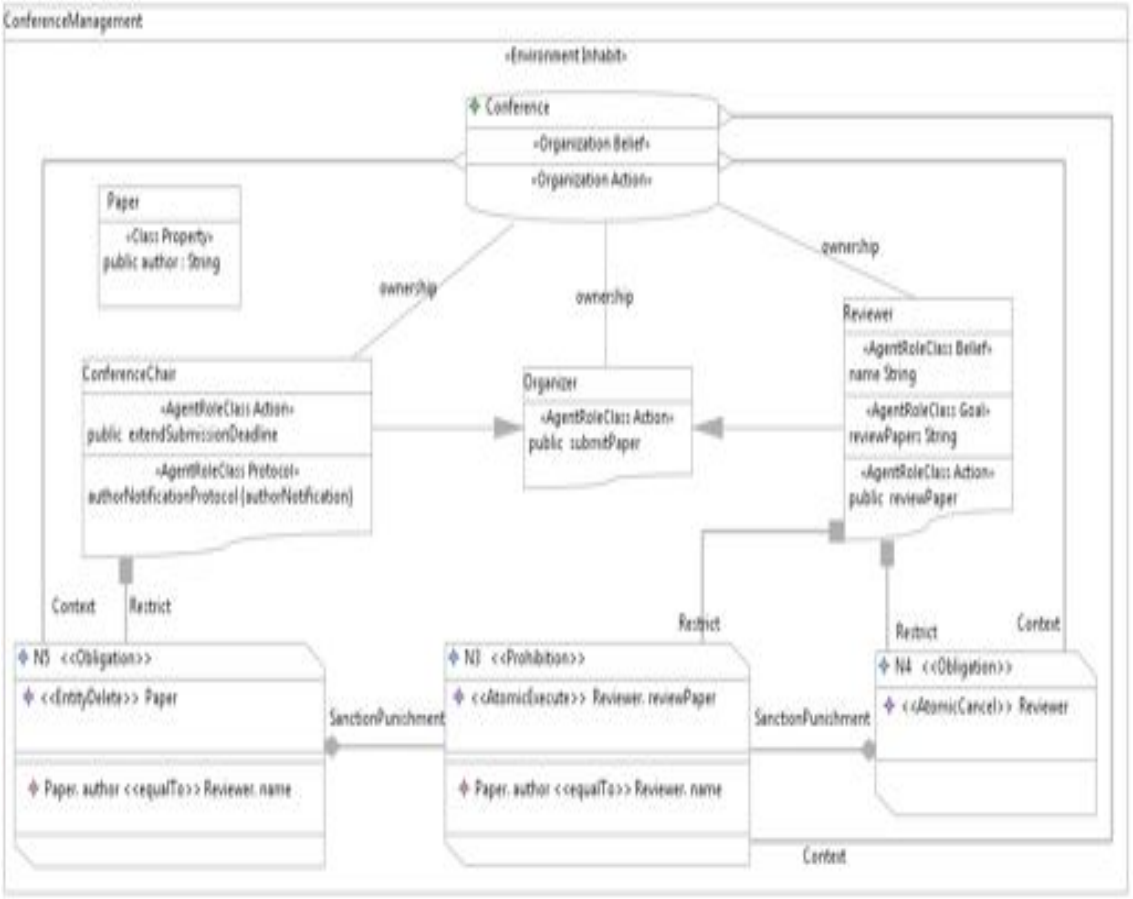

Figura 2. Modelagem das normas N3, N4 e N5 por meio do diagrama de normas.

5.3. Representação das Normas Utilizando JAMDER 2.0

Em JAMDER 2.0, cada uma das normas possui uma estrutura simples de classe que contém apenas a chamada do construtor. Os parâmetros que farão parte do construtor serão todos instanciados dentro da classe do ambiente *Conference Management*. A Figura 3 apresenta o código das normas N3, N4 e N5 do sistema.

Para a representação do ambiente *Conference Management*, é necessário identificar as características que compõe cada uma das normas. Essas características serão instanciadas dentro da classe de ambiente antes mesmo de instanciar a entidade norma.

A norma N3 determina que os revisores (entidades envolvidas) da organização *Conference* (contexto) estão proibidos (conceito deôntico) de revisar (ação) seus próprios artigos (restrição de ativação). Além disso, a norma N3 aplica duas punições (sanção) que são as normas N4 e N5. Se algum revisor violar N3, N4 obriga (conceito deôntico) que o dado revisor (entidade envolvida) da organização *Conference* (contexto) tenha o seu papel cancelado (ação) e N5 estabelece que o presidente da conferência (entidade envolvido) da organização *Conference* (contexto) é obrigado (conceito deôntico) a descartar o artigo (ação).

A Figura 4 apresenta o código parcial do ambiente *Conference Management* com as normas N3, N4 e N5.

```
import java.util.Hashtable;
import jamder.Organization;
import jamder.norms.*;
import jamder.roles.AgentRole;

public class N3 extends Norm{

        public N3 ( String name, NormType normType, AgentRole restrictAgentRoleClass,
Organization contextOrganizationClass, NormAction action, Hashtable<String, NormConstraint>
normConstraint, Hashtable<String, Norm> sactionPunishment){

        super(name, normType, restrictAgentRoleClass, contextOrganizationClass, action,
normConstraint);
        }
}
public class N4 extends Norm{

        public N4 (String name, NormType normType, AgentRole restrictAgentRoleClass,
Organization contextOrganizationClass, NormAction action,Hashtable<String, NormConstraint>
normConstraint ){

        super(name, normType, restrictAgentRoleClass, contextOrganizationClass, action,
normConstraint);
        }
}
public class N5 extends Norm{

        public N5 (String name, NormType normType, AgentRole restrictAgentRoleClass,
Organization contextOrganizationClass, NormAction action,Hashtable<String, NormConstraint>
normConstraint ){

        super(name, normType, restrictAgentRoleClass, contextOrganizationClass, action,
normConstraint);
        }
}
```

Figura 3. Representação das normas N3, N4 e N5 utilizando JAMDER 2.0.

6. Conclusão e Trabalhos Futuros

A diminuição do *gap* semântico entre as representações em fase de projeto e implementação contribui no aumento na produtividade dos desenvolvedores e na coerência entre o projeto das entidades e o código respectivo. Adicionalmente, a existência de um mapeamento entre as entidades em nível de modelagem e implementação garante a consistência e rastreabilidade entre os artefatos.

Este artigo apresenta a extensão do *framework* JAMDER através de um mapeamento entre modelagem e implementação que considera principalmente as normas e suas propriedades, devido à limitação da representação de conceitos normativos definidos pela linguagem de modelagem NorMAS-ML. O conjunto de classes associadas ao JAMDER propostas neste trabalho recebeu o nome de JAMDER 2.0, que contempla as entidades típicas de um SMA normativo como também as normas e suas propriedades.

Por fim, um estudo de caso baseado em um sistema de submissão de artigos foi utilizado para ilustrar a utilização do *framework* JAMDER 2.0, evidenciando sua aplicabilidade e adequação para o desenvolvimento de SMAs normativos.

Como trabalhos futuros, tem-se a necessidade de gerar código automaticamente do diagrama de normas baseado no *framework* JAMDER 2.0 utilizando a ferramenta MAS-ML *tool*.

```
import jamder.Environment;
import jamder.Organization;
import jamder.norms.AtomicAction;
import jamder.norms.AtomicActionType;
import jamder.norms.IfConditional;
import jamder.norms.Norm;
import jamder.norms.NormAction;
import jamder.norms.NormConstraint;
import jamder.norms.NormResource;
import jamder.norms.NormType;
import jamder.norms.Operator;
import jamder.roles.AgentRole;
import jamder.roles.ProactiveAgentRole;
import jamder.structural.Property;
import java.util.Hashtable;

public class ConferenceManagement extends Environment {
        public ConferenceManagement(String name, String host, String port) {
        super(name, host, port);
        //Organization
        Organization Conference = new Conference ("Conference", this, null);
        addOrganization("Conference", Conference);
        //AgentRole Played by Organization
        ProactiveAgentRole Organizer = new Organizer("Organizer", Conference, UserAgent);
        AgentRole Reviewer = new Reviewer("Reviewer", Conference, UserAgent);
        AgentRole ConferenceChair = new ConferenceChair("ConferenceChair", Conference,
UserAgent);
        addAgent("UserAgent", UserAgent);
        //Objects
        Object Paper = new Paper("Paper");
        addObject("Paper", Paper);

        //Norma N4
        NormResource resourceN4 = new NormResource(Reviewer);
        NormAction actionN4 = new AtomicAction(AtomicActionType.ATOMIC_CANCEL,
resourceN4);
        Norm N4 = new N4("N4",NormType.OBLIGATION, Reviewer, Conference, actionN4,
null);

        //Norma N5
        NormResource resourceN5 = new NormResource(Paper);
        NormAction actionN5 = new AtomicAction(AtomicActionType.ATOMIC_DELETE,
resourceN5);
        Norm N5 = new N5("N5",NormType.OBLIGATION, ConferenceChair, Conference, null,
null);

        //Norma N3
        NormResource resourceN3 = new NormResource(Reviewer.getAction("reviewPaper"));
        NormAction actionN3 = new AtomicAction(AtomicActionType.AtomicExecute,
resourceN3);

        Property first_property = new Property();
        Property second_property = new Property();
        first_property.setName(Paper.getAuthor());
        second_property.setName(Reviewer.getName());

        IfConditional constraintN3 = new IfConditional(Operator.EQUAL_TO,
first_property, second_property);
        Hashtable<String, NormConstraint> constraintsN3 = new Hashtable<String,
NormConstraint>();
        constraintsN3.put("contraintN3", constraintN3);
        Hashtable<String, Norm> sanctionsPunishmentN3 = new Hashtable<String, Norm>();
        sanctionsPunishmentN3.put("N4", N4);
        sanctionsPunishmentN3.put("N5", N5);
        Norm N3 = new N3("N3", NormType.PROHIBITION, Reviewer, Conference, actionN3,
constraintsN3, sanctionsPunishmentN3);
        Conference.addContextNorm("N3", N3);
        Reviewer.addRestrictNorm("N3", N3);

        }
```

Figura 4. Representação do ambiente *Conference Management* utilizando JAMDER 2.0.

Referências

Casillo, B. H. (2008) "Agentes auxiliando ambientes de engenharia de software centrado em processos", Dissertação de Mestrado. São José dos Campos: INPE.

Eclipse (2012), "Eclipse platform", http://www.eclipse.org/, Acessado em 12 de dezembro de 2012.

Figueiredo, K. e Silva, V. T. (2010) "*NormML: a modeling language to model norms*". In: 1st Workshop on Autonomous Software Systems. Salvador, Brazil.

Freire, E. S. S. ; Cortés, M. I. ; Goncalves, E. J. T. ; Lopes, Y. S. (2012) "*A Modeling Language for Normative Multi-Agent Systems*". In: 13th *International Workshop on Agent-Oriented Software Engineering* (AOSE@AAMAS), 2012, Valencia (Spain). Proceedings of the 13th *International Workshop on Agent-Oriented Software Engineering*.

Freire, E. S. S. ; Rocha Jr., R. M. ; Cortés, M. I. (2012) "Um Ambiente de Modelagem para Sistemas Multi-Agente Normativos". In: III Workshop on Autonomous Sotware Systems (Autosoft@CBSoft), 2012, Natal. Proceedings of III *Workshop on Autonomous Sotware Systems*.

Gonçalves, E. J. T. (2009) Modelagem de arquiteturas internas de agentes de software utilizando a linguagem MAS-ML 2.0. Dissertação de Mestrado. Universidade Estadual do Ceará. Centro de Ciência e Tecnologia. Fortaleza.

Huber, M. J. (2012) "JAM *Agent*". Disponível em: http://www.marcush.net/IRS, acessado em 12 de dezembro de 2012.

Jack (2012) "JACK *Agent Language*". Disponível em: http://www.agent-software.com.au/products/jack/, acessado em: 12 de dezembro de 2012.

Jade (2012) "*Java Agent Development Framework*", disponível em: http://jade.tilab.com/, acessado em: 12 de dezembro de 2012.

Jason (2012) "*Java-based interpreter for an extended version of AgentSpeak*". [S.l.], disponível em: http://jason.sourceforge.net/, acessado em: 12 de dezembro de 2012.

Lopes, Y. S. ; Goncalves, E. J. T. ; Cortés, M. I. ; Freire, E. S. S. (2012) "*A MDA Approach Using MAS-ML 2.0 and JAMDER*". In: 13th *International Workshop on Agent-Oriented Software Engineering* (AOSE@AAMAS), 2012, Valencia (Spain). Proceedings of the 13th *International Workshop on Agent-Oriented Software Engineering*.

Nunes, I. O. Implementação do Modelo e da Arquitetura BDI. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. Rio de Janeiro, 2007.

Russell, S. e Norvig, P. (2004) Inteligência Artificial: uma Abordagem Moderna, 2ª ed. Prentice-Hall: São Paulo.

Santos Neto, B. F. (2012) "Uma abordagem deontica para o desenvolvimento de agentes normativos autônomos", Tese de doutorado. Rio de Janeiro: PUC, Departamento de Informática.

Silva, V. T. (2004) "Uma linguagem de modelagem para sistemas multi-agentes baseada em um framework conceitual para agentes e objetos", Tese de doutorado. Rio de Janeiro: PUC, Departamento de Informática.