Model Oriented Approach to Code Generation for Normative Multi-Agent Systems

Robert M. R. Júnior, Emmanuel S. S. Freire e Mariela I. Cortés
Grupo de Engenharia e Sistemas Inteligentes (GESSI)
Departamento de Computação – Universidade Estadual do Ceará (UECE)
Fortaleza, Brasil
{robstermarinho, savio.essf}@gmail.com, mariela@larces.uece.br

***Abstract*—The increasing complexity of the normative multi-agent systems (MAS) development represents a challenge to software engineering. Model driven approach promotes a fast and consistent software development through the use of software models. In order to reduce the semantic gap that exists between modeling and implementation levels and surround the natural complexity associated to Normative MAS development, this work proposes the use of a model driven approach to develop Normative MAS. A template-based approach was used to automate the code generation process from NorMAS-ML models to the specific platform (JADE), it was named JAMDER 2.0 that contains all resources of JADE and adds new entities to adapt the concepts of each other.**

Keywords— Multi-Agent Systems; Norms; Model Driven Architecture; NorMAS-ML; JAMDER

I. INTRODUÇÃO

Sistemas Multi-Agente (SMA) normativos envolvem uma grande variedade de entidades, com isso, aumenta-se bastante a complexidade no processo de desenvolvimento [6]. O principal fator é a diferença semântica entre a fase de detalhamento do projeto ao longo de um conjunto de modelos, e a fase de implementação, que tem por objetivo a codificação do sistema. Nesse contexto, são necessárias linguagens de modelagem e programação que ajudem os desenvolvedores na construção de SMAs normativos e ferramentas que permitam a transição sistemática entre a modelagem e fases de implementação. Este artigo apresenta uma abordagem baseada na arquitetura orientada a modelos (MDA), a qual utiliza transformações e técnicas de geração de código a partir de artefatos de modelagem gerados pela linguagem de modelagem NorMAS-ML [6]. Diagramas da linguagem são gerados através do ambiente de modelagem MAS-ML Tool [7]. Como plataforma de implementação dessas entidades, destacamos o framework JAMDER 2.0 [10], definida como uma extensão do framework JAMDER (JADE to MAS- ML 2.0 *Development Resource*) [8]. Este trabalho está organizado da seguinte maneira: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a abordagem MDA. A Seção 4 aborda o framework JAMDER 2.0 e os *templates Acceleo* para JAMDER 2.0. A seção 5 apresenta um estudo de caso e a seção 6 apresenta as conclusões e trabalhos futuros.

II. TRABALHOS RELACIONADOS

Nesta seção, analisamos os principais trabalhos que abordam a geração de código para SMA normativos.

De Maria [4] propõe a geração de código baseado na arquitetura MDA para SMA utilizando a linguagem de modelagem MAS-ML e o *framework* de implementação ASF que contempla as entidades tipicamente encontradas em SMAs. Entretanto, possui suporte limitado aos conceitos normativos, pois a ferramenta é baseada em MAS-ML [11].

TAOM4E [13] é um ambiente de modelagem orientado a agente e suporta o desenvolvimento orientado a modelos. A ferramenta é um *plug-in* para a plataforma Eclipse, no entanto permite a geração de código apenas para agentes BDI (*belief-desire-intention*).

SMA *modeler* [12] é um metamodelo que possui independência com diferentes abordagens de desenvolvimento e plataformas de implementação e possibilita criar modelos de SMA. Porém este metamodelo não contempla todas as entidades de um SMA normativo e não possui uma ferramenta de modelagem.

Considerando a necessidade de uma abordagem de geração de código, a abordagem de De Maria [4] foi escolhida porque (i) sua geração de código é baseada na arquitetura MDA para SMA e (ii) utiliza a linguagem de modelagem MAS-ML que serviu como base para a definição da linguagem NorMAS-ML.

III. DESENVOLVIMENTO ORIENTADO A MODELOS

No desenvolvimento orientado a modelos (MDD), os diagramas da fase de modelagem possuem uma grande importância, pois qualquer modificação nos modelos conceituais reflete automaticamente no código gerado facilitando a manutenção e evolução dos sistemas além de trazer um foco maior na modelagem ao invés do código [5].

A arquitetura dirigida por modelos ou *Model Driven Architecture* (MDA) é uma iniciativa da OMG [9] com o intuito de formalizar os conceitos de MDD em um padrão para ser adotado pela comunidade e indústria de desenvolvimento de software. Com isso, a OMG propôs um processo de transformação, projetado para ser aplicado a diferentes linguagens de modelagem.

O código gerado a partir da transformação do modelo da aplicação é estabelecido pelo conceito ISM (*Implementation Specific Model*) dentro desta arquitetura. Neste contexto, podemos destacar o *plug-in* Acceleo [1] que permite a geração automática de código a partir de um metamodelo definido pelo

usuário que esteja de acordo com o EMF (*Eclipse Modeling Framework*). Como vantagens, o *plug-in* possui integração direta com o Eclipse e tem a possibilidade de extensão.

IV. Abordagem Baseada em Modelos para Sistemas Multi-Agente Normativos

No contexto da abordagem MDA, um suporte automatizado para a geração de código a partir de modelos de NorMAS-ML é proposto. A ferramenta MAS-ML Tool [7], desenvolvido como *plugin* para o Eclipse, permite a modelagem do diagrama de normas com base no metamodelo de NorMAS-ML. Esta ferramenta gera o arquivo masml. (formato XMI) que armazena a estrutura de dados das entidades e os aspectos estruturais e comportamentais definidos em NorMAS-ML. Estes arquivos representam a entrada para o processo de transformação a partir dos modelos de código.

Como o objetivo de fornecer a infraestrutura adequada voltada para a implementação de entidades de um SMA normativo de acordo com a linguagem NorMAS-ML, o framework JAMDER 2.0 é proposto.

A. *Jamder 2.0*

JAMDER 2.0 [10] é uma extensão do framework JAMDER [8] que incorpora os recursos oferecidos em nível de modelagem por NorMAS-ML, dentre eles, especificamente, as normas e suas propriedades. A extensão foi realizada por meio da inclusão de um conjunto de classes para representar os conceitos normativos. Essas classes são descritas a seguir: (i) para representar a entidade norma foi criada a classe *jamder.norms.Norm* que define as seguintes propriedades: o identificador, o tipo da norma, a entidade restringida, o contexto, a ação e a lista de restrições de ativação; (ii) a classe *jamder.norms.NormResource* representa qualquer tipo de recurso que será restringido; (iii) as ações vinculadas às normas foram representadas pela classe principal *jamder.norms.NormAction* e suas duas subclasses que definem operações do sistema: *AtomicAction* e *CompositeAction*; (iv) a classe *jamder.norms.NormConstraint* foi criada para a representação das restrições de ativação de uma norma, com suas respectivas subclasses que contemplam cada tipo de restrição: *jamder.norms.Before*, *jamder.norms.After*, *jamder.norms.Between* e *jamder.norms.IfConditional*. Ela está diretamente ligada à classe *jamder.norms.Date* responsável por definir uma data.

Outras classes existentes em JAMDER sofreram modificações para contemplar as propriedades definidas em NorMAS-ML. Essas modificações juntamente com as novas classes formam o framework JAMDER 2.0 que 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

B. *Geração de código*

Para definir o processo de transformação para a geração de código utilizamos o plugin Acceleo [1]. O plugin permite desenvolvimento incremental em que o código pode ser gerado, modificado e reutilizado. Para formalizar a geração de código em Acceleo, é necessário estabelecer um template para cada entidade através da linguagem MTL (*Model Transformation Language*) [9]. Quando o template é executado no Eclipse é necessário especificar o modelo NorMAS-ML (arquivos .masml no formato XMI) e a pasta de saída para as classes (classes geradas em JAMDER 2.0). Dessa forma, a estrutura de cada entidade no diagrama de modelagem é verificada pelo template.

Uma norma é uma instância de uma classe que herda a classe *Norm* definida em JAMDER 2.0 e seu *template* possui uma regra de definição do construtor que contempla os vários casos de modelagem envolvendo a entidade definida como contexto e a entidade restringida segundo a especificação feita na modelagem. A Figura 1 apresenta parte da estrutura do *template* para a entidade *Norm*.

```
[comment encoding = UTF-8 /]
[module Norm('masml')]
[template public generateElement(c : NormClass)]
[comment @main/]
[file (c.name + '.java', false, 'UTF-8')]
import java.util.Hashtable;
import jamder.Organization;
import jamder.norms.*;
import jamder.roles.AgentRole;

public class [c.name/] extends Norm{
        [if ((c.normContext.organizationClass->size() >
        0) and
             (c.normRestrict.agentRoleClass->size() >
        0))]
        public [c.name/] (String name, NormType
normType, AgentRole       restrictAgentRoleClass,
Organization contextOrganizationClass, NormAction
        action, Hashtable<String, NormConstraint>
normConstraint){
                super(name, normType,
restrictAgentRoleClass, contextOrganizationClass,
        action, normConstraint);
}
        [else][if ((c.normContext.organizationClass-
>size() > 0) and (c.normRestrict.organizationClass-
>size() > 0)) ]
```

Fig. 1. Parte do template da classe Norm.

O template para gerar a entidade ambiente é responsável por criar uma instância de uma classe que herda a classe ambiente de JAMDER 2.0 e analisa, segundo a modelagem, todos os relacionamentos que o ambiente possui, a fim de instanciar em seu construtor as entidades relacionadas na seguinte ordem: organizações, agentes, papéis de agente e por fim, as normas e suas propriedades. Outro template de JAMDER modificado foi o da entidade agente. JAMDER considera cinco tipos de agentes de acordo com suas arquiteturas internas. Sabendo que todos os agentes têm como propriedades comuns estar em um ambiente e executar pelo menos um papel nas organizações de que fazem parte, a classe *jamder.agents.GenericAgent* é a classe que representa as propriedades e demais atributos comuns entre as três ramificações da hierarquia de agentes especificada em JAMDER 2.0. Dessa forma o template especifica os dois casos de herança: (i) quando o agente herda de outro agente; (ii) quando o agente não herda de outro agente, então nesse caso, ele herdará de *GenericAgent*.

V. Estudo de Caso

Com o objetivo de ilustrar a geração de código, este estudo de caso aborda a criação de um SMA normativo responsável pela

gestão de submissões de artigos. Esses sistemas 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, a partir da qual, os avaliadores iniciam o processo de revisão. Após o término do período de revisão, os organizadores devem divulgar os resultados.

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*. Para o sistema de gestão de submissão de artigos foram definidas as seguintes normas: (i) N1: Os revisores estão proibidos de revisar seus próprios artigos; (ii) N2 (Punição para a violação da N1): Os revisores que violarem N1 devem ter seu papel cancelado.

A. *Geração de código*[1]

Cada entidade do sistema juntamente com as normas foram modeladas utilizando o diagrama de normas da MAS-ML Tool. A partir dessa modelagem segue a geração de código:

1) Normas: possuem uma estrutura simples de classe que contém apenas a chamada ao construtor. Os parâmetros do construtor serão todos instanciados dentro da classe Ambiente. A Figura 2 exibe o código gerado para as normas N1 e N2.

```
import java.util.Hashtable;
import jamder.Organization;
import jamder.norms.*;
import jamder.roles.AgentRole;
public class N1 extends Norm{
                  public N1 ( 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 N2 extends Norm{
         public N2 (String name, NormType normType, AgentRole
restrictAgentRoleClass, Organization
contextOrganizationClass, NormAction action,Hashtable<String,
NormConstraint> normConstraint ){
                  super(name, normType,
restrictAgentRoleClass, contextOrganizationClass, action,
normConstraint);
         }}
```

Fig. 2. Código gerado para as normas N1 e N2.

2) Ambiente: É necessário identificar todas as propriedades que compõe cada uma das normas. Essas propriedades serão instanciadas dentro da classe do ambiente antes mesmo de instanciar a entidade Norma.

VI. Conclusão e Trabalhos Futuros

Neste trabalho, descrevemos uma abordagem dirigida a modelos para desenvolver SMAs Normativos através de geração automática de código a partir de modelos, com base em artefatos gerados pela MAS-ML Tool. O processo de geração automática de código é realizado usando *templates Acceleo* que são responsáveis por criar classes em JAMDER 2.0. Uma das vantagens da geração de código é encapsular o processo de transição da fase de concepção para a fase de aplicação, aumentando a produtividade e reduzindo a ocorrência de erros nas atividades de prototipagem. A combinação destas três ferramentas (JAMDER 2.0, NorMAS-ML e *Acceleo*) fornece um desenvolvimento mais fácil para SMAs normativos, aproveitando o uso da plataforma Eclipse.

Como trabalho futuro pode ser abordada a evolução da linguagem NorMAS-ML e do framework JAMDER 2.0 para suportar a modelagem e a geração de código das entidades considerando os elementos dinâmicos das normas.

[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.

References

[1] ACCELEO, "Acceleo OpenSource"; disponível em: http://www.acceleo.org/. Acessado em 15 de janeiro de 2013.

[2] Beydeda, S., Book M., E Gruhn, V. (2005) "Model-driven Software Development." Birkhäuser,

[3] Blois, M., Lucena, C. (2004) "Multi-Agent Systems And The Semantic Web – The SemanticCore Agent-Based Abstraction Layer." In: ICEIS - International Conference on Enterprise Information Systems, 2004, Porto. Proceedings of Sixth International Conference on Enterprise Information Systems ICEIS 2004. Porto: INSTICC, 2004. p. 263-270.

[4] De Maria, B. A. (2004) "Usando a abordagem MDA no desenvolvimento de sistemas multi-agentes. " Dissertação de Mestrado – Pontífica Universidade Católica do Rio de Janeiro.

[5] France, R.; Rumpe, B; (2007) "Model-Driven Development of Complex Software: A Research Roadmap" In: Future of Software Engineering (FOSE'07) co-located with ICSE'07, Minnesota, EUA.

[6] 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.

[7] 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), 2012, Natal. Proceedings of III Workshop on Autonomous Sotware Systems.

[8] 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.

[9] OMG. "Object Management Group." Disponível em: http://www.omg.org. Acessado em 15 de janeiro de 2013.

[10] Rocha Jr., R. M. ; Freire, E. S. S. ; Cortés, M. I. (2013) "Estendendo o Framework JAMDER para Suporte à Implementação de Sistemas Multi-Agente Normativos ". In: IX Simpósio Brasileiro de Sistemas de Informação (SBSI), 2013, João Pessoa. Anais do IX Simpósio Brasileiro de Sistemas de Informação (SBSI), 2013.

[11] Silva, V. T.; Choren, R.; Lucena, C. J. P. (2007) "MAS-ML: A Multi-Agent System Modeling Language." Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA); In: Companion of the 18th annual ACM SIGPLAN Conference on Object-oriented programming, systems, languages, and applications; Anaheim, CA, USA, ACM Press, pp. 304-305.

[12] Santos, D. R. (2008) "Um Metamodelo para a Representação Interna de Agentes de Software. Dissertação de Mestrado." Porto Alegre: PUC.

[13] TAOM4E; "Tool for Agent Oriented Modeling." Disponível em: http://selab.fbk.eu/taom/

[14] Zambonelli, F.; Jennings, N. R.; Wooldridge, M. J. (2001) "Organisational Rules as an Abstraction for the Analysis and Design of Multi -Agent Systems." In: International Journal of Software Engineering and Knowledge Engineering, Volume 11, Number 3, p. 303-328.