Modelagem de Organizações de Agentes Inteligentes: uma Extensão da MAS-ML *Tool*

Enyo J. T. Gonçalves[1, 2], Kleinner Farias[3], Mariela I. Cortés[2]
Viviane Torres da Silva[4], Robson G. F. Feitosa[2]

[1] Universidade Federal do Ceará, Quixadá-CE – Brasil

[2] Universidade Estadual do Ceará, Fortaleza - CE – Brasil

[3] Pontifícia Universidade Católica do Rio de Janeiro, Rio de janeiro - RJ – Brasil

[4] Universidade Federal Fluminense, Niterói - RJ – Brasil

enyotavares@uece.br,kfarias@inf.puc-rio.br, mariela@larces.uece.br, viviane.silva@ic.uff.br, robsonf@gmail.com

Resumo. *É particularmente desafiante para os projetistas de software criar e validar manualmente diagramas de organizações de agentes inteligentes, ao mesmo tempo em que importantes questões de qualidade, tais como compreensibilidade e consistência, sejam garantidas. Este artigo ataca esta problemática através da adição do diagrama de organização à ferramenta MAS-ML tool, em conformidade com MAS-ML 2.0. Adicionalmente, o diagrama de classes da ferramenta é adaptado à nova versão da linguagem de modelagem.*

1. Introdução

A indústria de software e a academia têm cada vez mais desenvolvido pesquisas e fornecido tecnologias no intuito de atender à demanda da construção de sistemas de software cada vez mais complexos. Neste cenário, Sistemas Multi-Agentes (SMAs) surgem como uma abordagem promissora na tentativa de melhor gerenciar esta complexidade. De acordo com [Jennings 2000], SMAs podem ser entendidos como sociedades de agentes em que entidades autônomas e heterogêneas podem trabalhar de forma conjunta para fins similares ou totalmente distintos. SMAs têm se tornado um poderoso paradigma de engenharia de software [Mubarak 2008] e têm sido utilizados com sucesso para o desenvolvimento de diferentes tipos de sistemas de software, tanto na academia quanto na indústria [Lind 2001] [Wooldridge 2001].

Diante deste cenário, linguagens de modelagem para SMAs desempenham um papel central dentro do processo de desenvolvimento. Em especial neste artigo, damos ênfase à MAS-ML (*Multi-Agent System Modeling Language*) [Silva et al. 2004]. A escolha da linguagem MAS-ML foi motivada pelo (1) seu reconhecimento diante da necessidade de modelar SMAs em projetos reais [C. Nunes et al. 2009] [I. Nunes et al. 2009], e (2) por apresentar as definições necessárias para modelar os conceitos e abstrações apresentadas no paradigma de SMAs. Porém, algumas limitações da linguagem foram observadas na modelagem de agentes com arquiteturas internas diversas. Sendo assim, a linguagem MAS-ML foi evoluída para suprir estas deficiências, originando a MAS-ML 2.0 [Gonçalves et al. 2009] [Gonçalves et al. 2010].

O presente artigo apresenta uma evolução da ferramenta MAS-ML *tool* [Farias et al. 2009] relacionada à: (i) adequação do diagrama de classes (já presente na ferramenta) à MAS-ML 2.0; e (ii) adição do diagrama de organização à MAS-ML *tool* em conformidade com MAS-ML 2.0. O artigo é organizado como segue: Na Seção 2 é apresentado o referencial teórico em relação à MAS-ML *tool* e MAS-ML 2.0. Na Seção

3, a evolução da ferramenta é apresentada. Na Seção 4, um estudo de caso é ilustrado. Na Seção 5, os trabalhos relacionados são confrontados com as contribuições deste artigo. E por fim, na Seção 6, são apresentadas as conclusões e os trabalhos futuros.

2. Referencial Teórico

Nesta seção é apresentado o referencial teórico em relação aos conceitos de MAS-ML tool e da linguagem de modelagem MAS-ML em sua versão 2.0.

2.1. Background: MAS-ML *tool*

A ferramenta MAS-ML *tool* foi desenvolvida como um *plug-in* da plataforma Eclipse [Eclipse 2009]. Isso implica que os usuários podem trabalhar com modelagem de SMAs ao mesmo tempo em que fazem uso dos recursos oferecidos pela plataforma Eclipse. Dado que muitas plataformas de agentes são implementadas em Java, tais como Jade [Bellifemine, Caire, e Greenwood 2007], Jadex [Pokahr, Braubach e Lamersdorf 2003], Jason [Bordini, Wooldridge e Hübner 2007], o uso da plataforma Eclipse também facilita uma possível geração de código dentro do mesmo ambiente de desenvolvimento.

O desenvolvimento da ferramenta seguiu uma abordagem dirigida por modelos, onde o modelo central é o próprio metamodelo da linguagem MAS-ML, na sua versão inicial. Os componentes principais da MAS-ML *tool* são os seguintes:

(A) **Package Explorer** - Tem como funcionalidade central permitir a organização dos arquivos em uma estrutura de árvore, facilitando o gerenciamento e manipulação;

(B) **Modeling View** - Permite aos desenvolvedores visualizar e editar os modelos de forma interativa;

(C) **Nodes Palette** - Os construtores que podem fazer parte dos diagramas encontram-se na *nodes palette,* cujas instâncias são visualizadas na *modeling view*.

(D) **Relationship Palette** - Os relacionamentos que podem ser estabelecidos entre os construtores presentes na nodes palette são disponibilizados na *relationship palette*.

(E) **Properties View** - Permite manipular com precisão as propriedades dos modelos.

(F) **Problems View** - Apresenta eventuais inconsistências no modelo. Esta funcionalidade é particularmente importante para viabilizar o uso da modelagem de SMAs dentro do contexto do desenvolvimento dirigido por modelos.

(G) **Outline View** - Uma vez que um modelo tenha sido criado, a visualização da distribuição dos elementos presentes no mesmo é realizada através da *outline view*.

2.2. MAS-ML 2.0

MAS-ML 2.0 [Gonçalves et al. 2009] [Gonçalves et al. 2010] é uma extensão da linguagem de modelagem MAS-ML visando adequá-la a modelagem de: (i) arquiteturas internas de agentes reativos simples; (ii) agentes reativos baseados em conhecimento; e (iii) agentes baseados em objetivo com planejamento e agentes baseados em utilidade.

Em termos práticos, a extensão da linguagem envolveu a evolução do metamodelo da MAS-ML com o objetivo de inserir novas meta-classes e especificar como os relacionamentos, previamente definidos, interagem com as mesmas. A citada extensão envolveu a criação de duas meta-classes: *AgentPerceptionFunction*, que representa as percepções do agente; e *AgentPlanningStrategy*, que representa o planejamento do agente. Ambas especializam a meta-classe *BehavioralFeature* da UML. Além das duas meta-classes, foram criados quatro estereótipos: *formulate-goal-*

function, para representar a função de formulação de objetivo; *formulate-problem-function*, para representar a função de formulação de problema; *next-function*, para representar a função próximo; e *utility-function*, para representar a função utilidade [Gonçalves et al. 2009] [Gonçalves et al. 2010].

A partir dos novos elementos no metamodelo, a representação do agente nos diagramas de MAS-ML ganhou quatro variantes gráficas, onde cada uma representa cada uma das arquiteturas internas citadas anteriormente. Conseqüentemente, o elemento papel de agente passou a ter três representações: (i) a representação inicial de MAS-ML foi mantida; (ii) uma representação sem objetivos, associada a agentes reativos baseados em conhecimento; e (iii) uma representação sem objetivos nem crenças, associada a agentes reativos simples.

Em relação aos diagramas, o metamodelo de MAS-ML na sua concepção original [Silva *et al.* 2004] contempla os diagramas de Classes, Organização, Papéis, Seqüência e Atividades. MAS-ML 2.0 manteve as representações existentes, no entanto, esses diagramas foram alterados adequando o metamodelo original da linguagem para a modelagem das diferentes arquiteturas internas de agentes.

3. Evolução da MAS-ML *tool*

A versão da ferramenta utilizada como base para o presente trabalho disponibiliza exclusivamente o diagrama de classes em conformidade com a versão original de MAS-ML. Considerando a evolução da linguagem de modelagem torna-se necessária a evolução da ferramenta de apoio de forma a propiciar a correta geração do diagrama de classes em consistência com a nova especificação da linguagem. Adicionalmente, a ferramenta é enriquecida incorporando apoio à criação do diagrama de organização MAS-ML 2.0.

A evolução da ferramenta segue a abordagem dirigida por modelos que foi utilizada originalmente para seu desenvolvimento. Neste caso, o metamodelo da linguagem MAS-ML 2.0 é usado como guia para as alterações no metamodelo da ferramenta [Farias et al. 2009], o qual é utilizado como base da implementação. O desenvolvimento da ferramenta se baseou em uma abordagem generativa utilizando o GMF [GMF 2009] e pode ser resumido em 6 etapas descritas como segue:

1. **Extensão do Modelo de Domínio.** Ao partir da especificação original do metamodelo usando o EMOF[1] (*Essencial Meta-Object Facility*), os estereótipos apresentados na seção anterior foram adicionados à *ActionClass* através de uma semântica chamada *ActionSemantics*. Esta semântica representa as opções 0- sem estereótipo, 1- next-function, 2- utility-function, 3- formulate-problem-function e 4- formulate-goal-function.

2. **Extensão do Modelo Gráfico.** O Modelo Gráfico contém as definições das entidades e suas propriedades, e dos relacionamentos com base no metamodelo da linguagem. No contexto da evolução, as novas meta-classes apresentadas na seção anterior foram adicionadas na "Definição do Modelo Gráfico" da ferramenta, criando os relacionamentos necessarios. As multiplicidades dos relacionamentos entre as entidades *goal*, *belief* e *plan* e a entidade *AgentClass* foram alteradas de 1 para 1..* para 1 para 0..*. As entidades *perception* e *planning* possuem um relacionamento com multiplicidade 1 para 0..*.

[1] O EMOF é parte integrante do plug-in EMF (Eclipse Metamodel Framework) que consiste em um framework de modelagem e de geração de código para construir aplicações baseadas em modelos [EMF 2009]

3. **Extensão do Modelo de Ferramenta.** Esta etapa especifica quais elementos fazem parte da paleta da ferramenta. Para isso, essa etapa recebe como entrada o modelo de domínio e o modelo gráfico determinados anteriormente. Adicionalmente, para o diagrama de classes foram criados novos elementos na paleta da ferramenta para criar percepções e planejamento. Na representação do modelo de ferramenta do diagrama de Organização, foram adicionados os elementos *Perception*, *Planning*, *AgentRoleClass*, *ObjectRoleClass*, *Duty*, *Right* e *Protocol* e os relacionamentos *inhabit*, *ownership* e *play*.

4. **Extensão da Definição do Modelo Gráfico.** Esta etapa está relacionada com a representação dos elementos no respectivo diagrama. O modelo Gráfico, que tem como entrada o modelo de domínio, é utilizado na combinação para gerar o modelo de mapeamento. Para esta etapa, foram criados compartimentos para a representação das percepções e do planejamento, além da representação dos nós (*nodes*) provenientes da etapa anterior.

   Na representação do diagrama de Organização, na definição do modelo gráfico, foram adicionados os elementos *Planning, Perception, AgentRoleClass, ObjectRoleClass, Duty, Right e Protocol*, compartimentos para *Planning, Perception, Duty, Right* e *Protocol* e os relacionamentos *inhabit, ownership* e *play*.

5. **Extensão do Modelo de Mapeamento.** Este modelo é criado a partir de um mapeamento entre os modelos: modelo de domínio, modelo gráfico, e o modelo de ferramenta. O mapeamento gerado pode ser usado como entrada em um processo de transformação no intuito de criar um modelo específico de plataforma. Novas regras de validação foram implementadas (exemplos no Quadro 1) através de restrições em OCL (*Object Constraint Language*) a partir da multiplicidade das meta-classes presentes na "Definição do Modelo Gráfico" da ferramenta.

Quadro 1 – Regras de validação dos modelos.

Regra	Propósito e Definição em OCL
Regra 1	Se o agente possui plano, então ele possui objetivo, crença e ação.
	self.ownedPlan->isEmpty() = false implies self.ownedGoal->isEmpty() = false and self.ownedBelief->isEmpty() = false and self.owendAction->isEmpty() = false and self.ownedPlan->isEmpty() = false
Regra 2	Se o agente possui plano, então não possui percepção.
	self.ownedPlan->isEmpty() = false implies self.ownedPerception->isEmpty() = true and self.ownedPlan->isEmpty() = false

6. **Geração da Ferramenta.** Finalmente, seguindo a abordagem generativa [Czarnecki e Eisenecker 2000], e feita a geração do código da ferramenta a partir do modelo específico de plataforma estendido na etapa anterior.

No diagrama de classes, o agente passou a ter dois compartimentos extras para abrigar as percepções do agente e o planejamento, onde uma ação pode ter um estereótipo associado para representar a *função próximo*, formulação de objetivo, formulação de problema e utilidade. A representação do agente é apresentada na Figura 1, e o diagrama de classes do SMA para o TAC-SCM[2] [Sadeh et al. 2003] é apresentado na Figura 4. Ambos diagramas foram gerados a partir da ferramenta MAS-ML *tool* após a extensão.

[2] O TAC-SCM (*Trading Agent Competion - Supply Chain Management*) é um ambiente que possibilita a realização de leilões simultâneos, para testar técnicas, algoritmos e heurísticas em agentes de negociação.

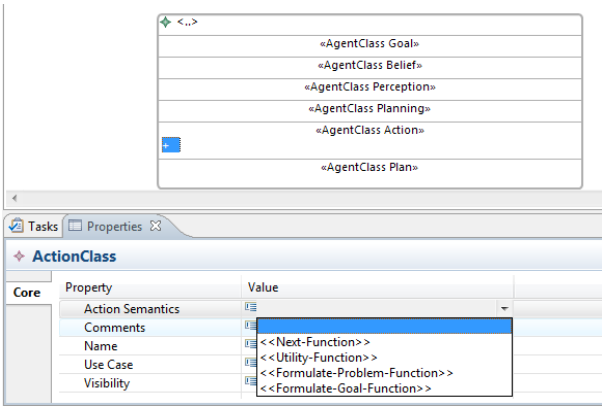

Figura 1. Nova representação do agente na ferramenta MAS-ML *tool*.

A estrutura criada para o diagrama de classes foi aproveitada para a criação do diagrama de organização, pois grande parte das entidades é comum aos dois diagramas. Os papéis de agente foram representados de acordo com a extensão proposta por Gonçalves et al. (2010), ilustrada na Figura 2 a partir da ferramenta MAS-ML *tool*.

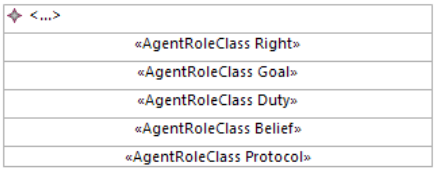

Figura 2. Representação do papel de agente na ferramenta MAS-ML tool.

Os relacionamentos *posse* e *exerce* definidos em MAS-ML, os quais fazem parte do diagrama de organização, foram adicionados. O relacionamento *habita* teve sua semântica alterada para permitir que agentes, papéis de agente e organizações possam ser considerados. Na Figura 3, o diagrama de Organização para TAC-SCM.

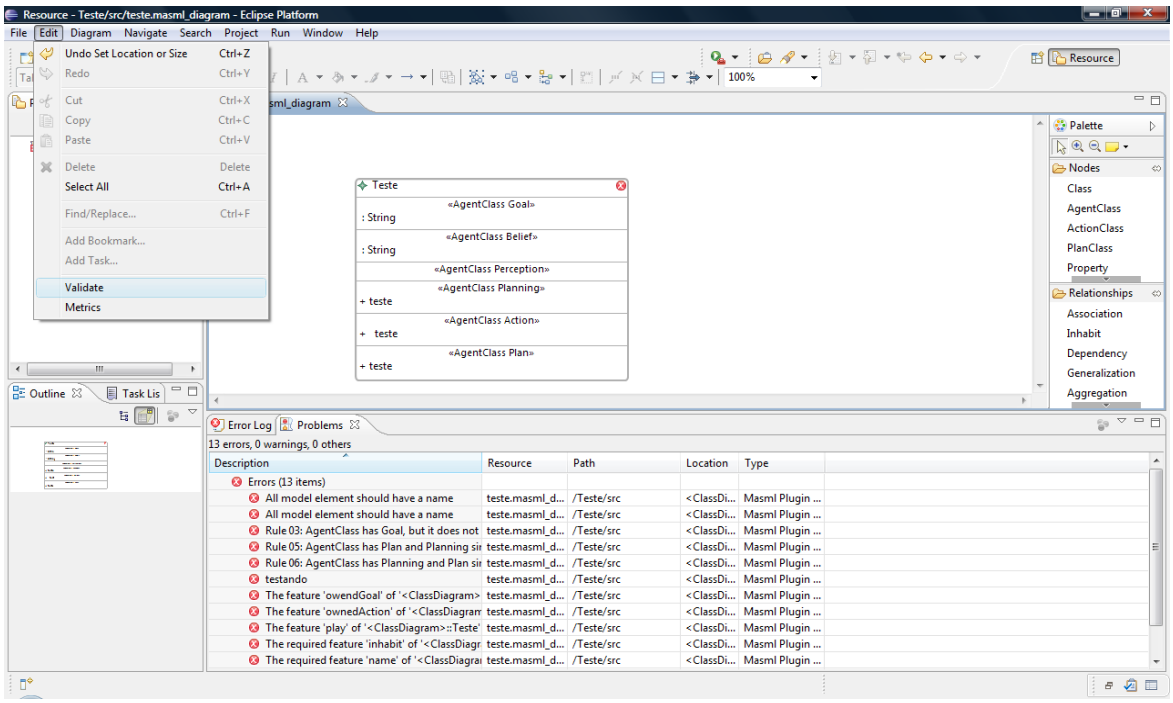

Figura 3. Validação na ferramenta MAS-ML tool.

Os modelos gerados a partir da ferramenta podem ser validados aplicando as restrições OCL implementadas, as quais podem ser acessadas através do menu *edit* e da

opção *validate*. Caso alguma regra seja violada, o elemento que apresenta problema é assinalado e o detalhamento dos problemas é apresentado através do recurso *problems* do eclipse. A Figura 5 apresenta o recurso de validação no Eclipse, o elemento *Teste* do diagrama que apresenta problemas detectados pela validação e o respectivo relatório.

A extensão do diagrama de classes da ferramenta MAS-ML tool envolveu: a criação de duas novas metaclasses, a alteração da multiplicidade de alguns relacionamentos, criação de novos compartimentos, bem como a representação das novas entidades na paleta e criação de novas restrições OCL.

Apoio à geração do diagrama de Organização foi incorporado na ferramenta em consistência com as alterações implementadas para o diagrama de classes. Foi necessário remover a representação dos relacionamentos associação, dependência, generalização, agregação e composição, pois segundo Silva (2004) os relacionamentos presentes neste diagrama são posse (*ownership*), exerce (*play*) e habita (*inhabit*). O relacionamento *inhabit* foi mantido e os relacionamentos *ownership* e *play* foram adicionados na representação deste diagrama. Os papéis de agente e de objeto, que não eram modelados no diagrama de classes, foram adicionados. Vale ressaltar que os elementos adicionados já estavam presentes no modelo de domínio e no modelo gráfico, porém não eram utilizados na representação inicial.

4. Estudo de caso

Nesta seção é apresentada a modelagem de um SMA para TAC-SCM (*Trading Agent Competition - Supply Chain Management*) utilizando MAS-ML tool. Inicialmente o TAC-SCM será descrito e em seguida a modelagem será apresentada.

4.1. TAC-SCM

Cadeias de Suprimento são ambientes altamente dinâmicos, estocásticos e estratégicos [Arunachalam 2004]. O TAC SCM foi projetado para capturar os desafios presentes em um ambiente integrado de aquisição de matéria-prima, produção de bens e oferta para clientes. O jogo descreve o cenário de uma cadeia de suprimentos para a montagem de computadores pessoais, consistindo de uma fábrica de computadores, fornecedores que provêem componentes para a montagem destes computadores e clientes que demandam computadores prontos [Sadeh et al. 2003].

O jogo consiste em uma seqüência de dias simulados ou rodadas em que os agentes precisam realizar tarefas para gerenciar a cadeia de suprimento. Os agentes têm uma conta no banco com saldo inicial igual a zero. A cada dia, clientes lançam pedidos de orçamentos e selecionam os orçamentos submetidos com base na data de entrega e no preço de oferta. Os agentes são limitados pela capacidade de produção de suas linhas de montagem e têm que obter componentes de um conjunto de fornecedores. A demanda de clientes vem na forma de pedidos de orçamento para diferentes tipos de computadores pessoais. O jogo começa quando um ou mais agentes se conectam ao jogo. O jogo simula fornecedores e clientes, provê banco, produção e serviço de estocagem de mercadoria para agentes individuais. Ao final, o agente com maior soma em dinheiro no banco é declarado vencedor [Collins et al. 2006].

4.2. Modelagem do SMA para TAC-SCM com MAS-ML tool

Um único SMA pode conter agentes com diferentes arquiteturas internas. Nesse contexto Weiss (1999) descreve que é possível balancear o comportamento dos agentes de um SMA em relação a pró-atividade e reatividade. Levamos em consideração na

escolha da arquitetura interna de cada agente que devemos escolher a arquitetura do agente em relação à função que ele irá desempenhar no SMA. Consequentemente o SMA envolve os seguintes agentes (Figura 4):

AgenteVendedor é um agente reativo simples para ofertar computadores aos clientes e receber o pagamento.

AgenteComprador é um agente reativo baseado em conhecimento para decidir quando realizar novos lances, o valor do lance e o realizar pagamento.

AgenteGerente é um agente baseado em utilidade [Russell e Norvig 2004], que deverá encontrar uma melhor maneira de alocação dos recursos, frente a demanda corrente, para maximizar o lucro e maximizar as vendas.

AgenteProdução é um agente baseado em objetivo guiado por planejamento [Russell e Norvig 2004]. Este agente é responsável por gerenciar o estoque e montar os computadores frente à demanda.

AgenteEntrega é um agente baseado em objetivo guiado por plano. Este agente é responsável por entregar os produtos do estoque aos clientes.

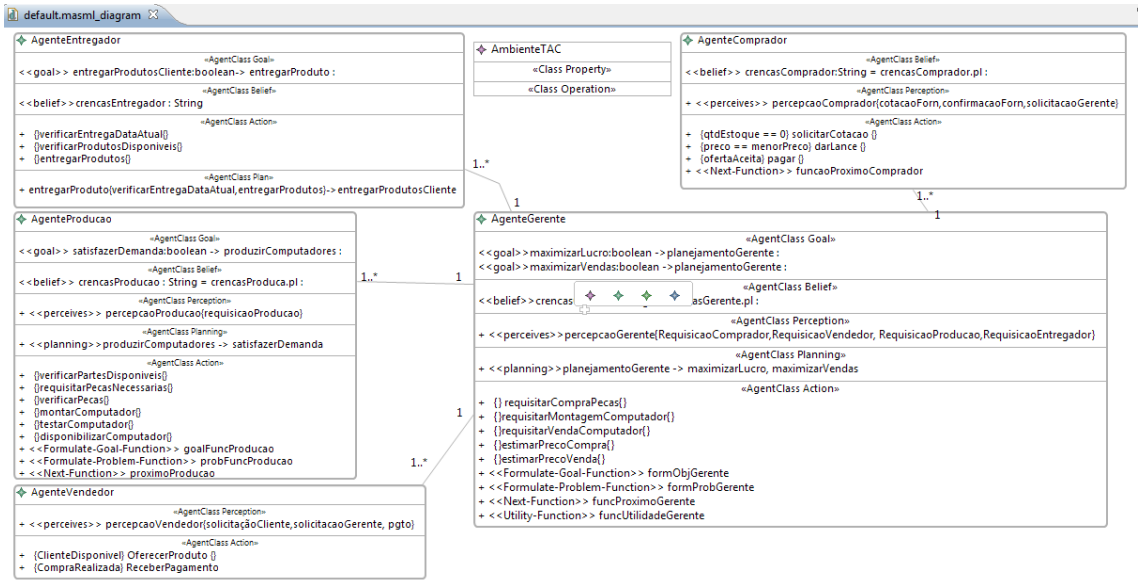

Figura 4. Diagrama de classes para TAC-SCM.

Na Figura 5 os agentes e seus relacionamentos descritos são representados através do diagrama de Organização criado através da nova versão de MAS-ML tool.

5. Trabalhos Relacionados

Considerando um escopo amplo em relação às ferramentas de apoio para a modelagem de SMAs, a ferramenta *Prometheus Design Tool* (PDT) e *AgentTool* [AgentTool 2010] podem ser citadas. A ferramenta PDT foi idealizada com o objetivo de dar suporte à metodologia Prometheus [Padgham, Thangarajah e Winikoff 2008], enquanto que *AgentTool* está associada à metodologia de desenvolvimento de SMAs O-MaSe (*Organization-based Multiagent Systems Engineering*) [Garcia-Ojeda, DeLoach e Robby 2009]. Uma questão essencial é que as ferramentas diferem-se na linguagem de modelagem por elas suportadas, assim, as vantagens e limitações dessas linguagens são propagadas para as ferramentas que as implementam.

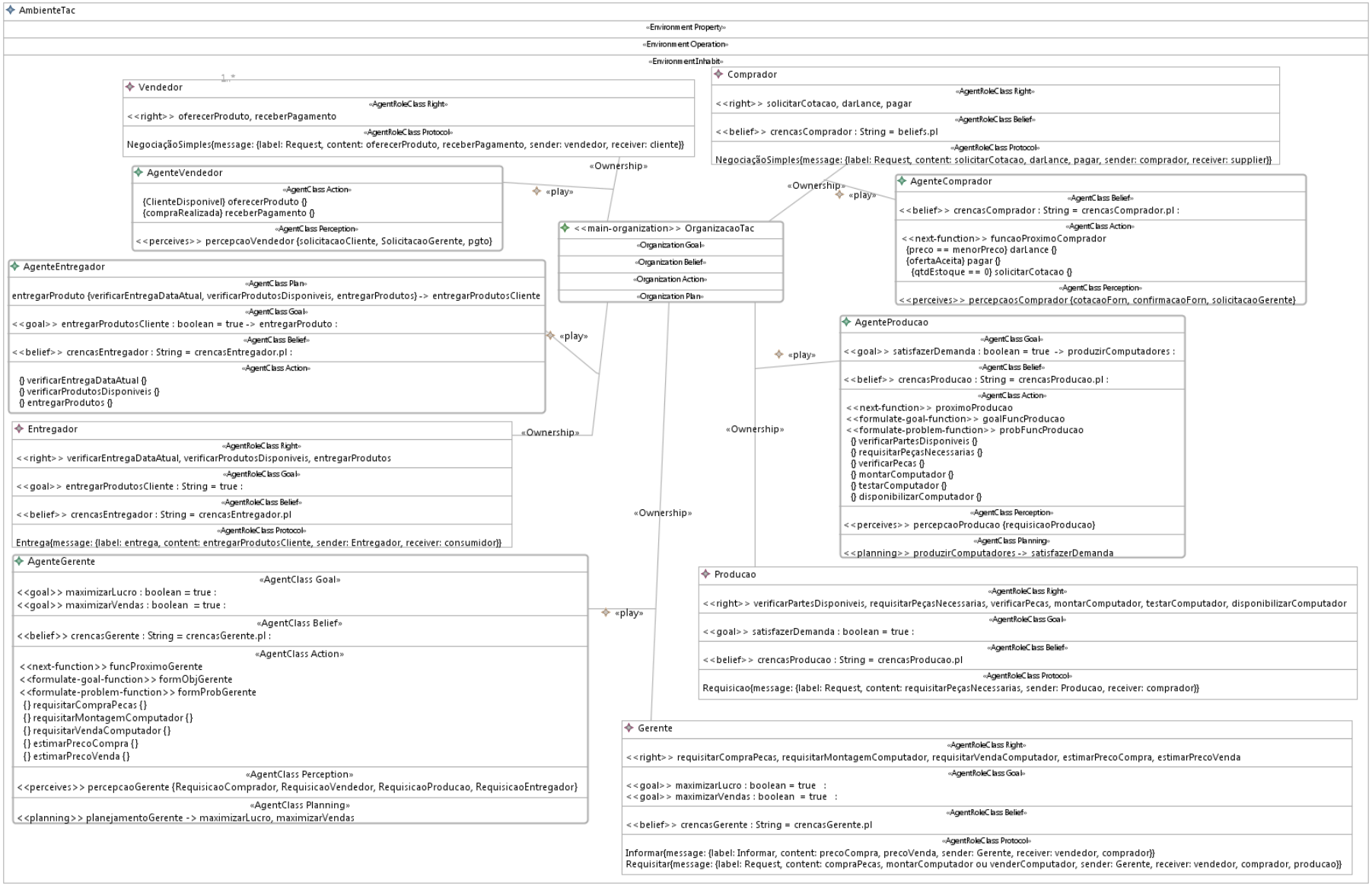

Figura 5. Diagrama de Organização para o estudo de caso do TAC-SCM.

VisualAgent [De Maria et al. 2005] é um ambiente de desenvolvimento de software que visa auxiliar desenvolvedores na especificação, projeto e implementação de SMAs. Ela também propõe uma abordagem dirigida por modelo para o desenvolvimento de SMAs fazendo uso do ASF (*Agent Society Framework*) [Silva, Cortés e Lucena 2004], o que representa um ponto relevante da abordagem. Similarmente, MAS-ML *tool* [Farias et al. 2009] é um ambiente de modelagem específico de domínio que atende à modelagem de MAS de acordo com a especificação do metamodelo de MAS-ML. Adicionalmente, MAS-ML *tool* fornece mecanismos para realizar verificação de regras de boa forma dos modelos em relação ao metamodelo da MAS-ML. A ausência desta funcionalidade em *VisualAgent* pode comprometer a qualidade dos modelos e código gerados. Além disso, a falta de documentação e acesso ao código fonte dificulta sua evolução.

6. Conclusões e Trabalhos Futuros

Este artigo apresenta a evolução de uma ferramenta, que consiste em uma prova conceitual da linguagem de modelagem MAS-ML em sua versão 2.0, a qual adiciona recursos para modelar agentes racionais de maneira adequada, propiciando um melhor nível de abstração para representar características internas de SMAs. Partindo do princípio que modelagem é uma atividade essencial dentro do processo de desenvolvimento de software, a existencia de ferramentas de apoio à esta atividade se torna de fundamental importância. No presente trabalho é apresentada a evolução de um ambiente de modelagem para SMAs com foco no diagrama de classes e organizações da MAS-ML 2.0. Com o ambiente é possível, dentre outras atividades, realizar a modelagem, validar os modelos criados e realizar sua persistência.

Como trabalhos futuros alguns melhoramentos podem ser realizados em relação à representação gráfica dos construtores. Adicionalmente, o suporte para a modelagem dos demais diagramas estáticos da MAS-ML 2.0, bem como os diagramas dinâmicos que não foram contemplados neste trabalho é sugerido.

Referências

agentTool, http://agenttool.cis.ksu.edu/, acessado em 15 de Junho de 2010.

Arunachalam, R, (2004). The 2003 supply chain management trading agent competition. In: Third International Joint Conference on Autonomous Agents & Multi Agent Systems. p. 113–120.

Bellifemine, F. L.; Caire, G.; Greenwood, D. (2007) Developing Multi-Agent Systems with JADE. [S.l.]: Wiley (Wiley Series in Agent Technology).

Bordini, R.; Wooldridge, M.; Hübner, J. (2007). Programming Multi-Agent Systems in AgentSpeak using Jason, John Wiley & Sons.

Collins, J.; Arunachalam, R.; Sadeh, N.; Eriksson, J.; Finne, N.; Janson, S., (2006). The Supply Chain Management Game for the 2007 Trading Agent Competition. Available in http://www.sics.se/tac/tac07scmspec.pdf.

Czarnecki, K.; Eisenecker, U. (2000). Generative Programming - Methods, Tools, and Applications, Adison-Wesley, June 2000.

De Maria, B. A.; Silva, V. T.; Lucena, C.J.P.; Choren, R. (2005). VisualAgent: A Software Development Environment for Multi-Agent Systems. Proceedings of the 19º Simpósio Brasileiro de Engenharia de Software, Tool Track, Brazil.

Eclipse Platform (2009). www.eclipse.org, acessado em 15 de Julho de 2009.

Farias, K.; Nunes, I.; Silva, V. T. da; Lucena, C. J. P. de (2009). MAS-ML Tool: Um Ambiente de Modelagem de Sistemas Multi-Agentes. Fifth Workshop on Software Engineering for Agent-oriented Systems (SEAS@SBES 09), Brazil.

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

GMF (2009). www.eclipse.org/modeling/gmf/, Acessado em 15 de Julho de 2009.

Gonçalves, E. J. T.; Campos, G. L.; Cortés, M. I.; Silva, V. T. da (2009). Modelagem de Agentes Reativos utilizando MAS-ML. Fifth Workshop on Software Engineering for Agent-oriented Systems (SEAS@SBES 09), Brazil.

Gonçalves, E. J. T.; Cortés, M. I.; Campos, G. L.; Silva, V. T. (2010). Extending MAS-ML Tt Model Proactive and Reactive Sotware Agents. 12th International Conference on Enterprise Information System (ICEIS 2010), Funchal, Portugal.

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

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

Mubarak, H. (2008), Developing Flexible Software Using Agent-Oriented Software Engineering, IEEE Software, Sep/Oct, IEEE Computer Society, pp. 12-15.

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

Nunes, I.; Nunes, C.; Kulesza, U.; Lucena, C. (2009) Developing and Evolving a Multi-Agent System Product Line: An Exploratory Study. In Agent-Oriented Software Engineering IX: Springer-Verlag, v. 5386, p. 228-242.

Padgham, L.; Thangarajah, J.; Winikoff, M. (2008) Prometheus Design Tool, in 23th AAAI Conference on Artificial Intelligence, Chicago, EUA, pp.1882-1883.

Pokahr, A.; Braubach, L.; Lamersdorf, W. (2003). Jadex: Implementing a BDI-Infrastructure for JADE Agents. EXP - In Search of Innovation (Special Issue on JADE), vol. 3, no. 3 , Telecom Italia Lab, Turin, Italy, S. 76-85.

Sadeh, N.; Arunachalam, R.; Erikson, J.; Finne, N.; Janson, S., (2003). A supply-chain trading competition. AI Magazine, v. 24, n. 1, p. 92–94.

Silva, V. T.; Choren, R.; Lucena, C. J. P. de (2007). MAS-ML: A Multi-Agent System Modeling Language. Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), ACM Press, pp. 304-305.

Silva, V. T.; Cortés, M.; Lucena, C. (2004). An Object-Oriented Framework for Implementing Agent Societies. MCC32/04. Technical Report, PUC-Rio. Brasil.

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

Weiss, G. (1999). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Massachusetts.

Wooldridge, M.; Ciancarini, P. (2001), Agent-Oriented Software Engineering: the State of the Art, In Agent-Oriented Soft. Eng., LNCS1957,Springer, p. 1-28.