Desenvolvimento de agentes: Uma análise da utilização da metodologia Prometheus

Vanessa M. Berny [1] **and Diana F. Adamatti**[2] **and Antonio C. da Rocha Costa** [1]

[1]Programa de Pós-Graduação em Informática
Centro Politécnico - Universidade Católica de Pelotas - Brasil

[2]Centro de Ciências Computacionais - Universidade Federal do Rio Grande - Brasil

vmberny@gmail.com, dianaadamatti@furg.com.br, rocha@atlas.ucpel.tche.br

Abstract. *This paper presents an analysis to develop agents and which is the importance to use software engineering methodologies in this development. In this work, we used two approaches: in the first one, we chose a modelling methodology for software agents, called Prometheus. In the second one, we use "free"code, generating without a methodology.*

Resumo. *Este artigo apresenta uma análise para desenvolvimento de agentes e qual a importância da utilização de metodologias advindas da engenharia de software para este desenvolvimento. Foram utilizadas duas abordagens: na primeira, utilizou-se uma metodologia de modelagem para agentes de software, denominada Prometheus. Na segunda, o código foi gerado de forma "livre", sem utilização de nenhuma metodologia.*

1. Introdução

A escolha da abordagem baseada em Sistemas Multiagentes (SMA) para resolução de problemas tem sido um assunto muito explorada nas últimas décadas. Inúmeros trabalhos têm apresentado conceitualizações, formalizações, protocolos, técnicas e métodos para aplicação deste tipo de abordagem na concepção de softwares. Isso tem acontecido pelo fato da abordagem SMA possuir algumas características que viabilizam a resolução de problemas de outra forma que não a tradicional, adequando-se à problemas complexos [Pressman 2006] e de natureza descentralizada [Moulin and Chaib-Draa 1996].

A modelagem de um SMA é bastante complexa. Segundo Bastos [Bastos 1998], é possível modelar um agente utilizando uma abordagem Orientada a Objetos (OO). No entanto, persiste a problemática da modelagem da sociedade, pois, por não adequar-se ao modelo conceitual, a abordagem OO não viabiliza a modelagem de todos os aspectos envolvidos em um SMA [Iglesias et al. 1998, Wooldrige 1999, Kendall et al. 1996, Taveter 1999].

Várias metodologias, arquiteturas e ferramentas já foram desenvolvidas para facilitar o desenvolvimento de sistemas baseados em agentes. As técnicas criadas permitem a construção de aplicações desde as fases de análise e projeto até a implementação final do sistema.

O principal objetivo deste artigo é realizar uma análise da utilização de uma metodologia para modelagem de um SMA, baseado em um experimento. O artigo está organizado da seguinte forma: na seção 2 são apresentadas as algumas metodologias para

modelagem de SMA; na seção 3 apresenta o experimento, bem como a modelagem realizada utilizando a metodologia Prometheus e a análise entre os códigos gerados a partir da modelagem ”aberta” e a modelagem utilizando uma metodologia; e na seção 4 estão as conclusões deste artigo.

2. Metodologias de Desenvolvimento de Sistemas Multiagentes

Sob o ponto de vista da Engenharia de Software, a construção de software de alta qualidade de maneira produtiva é viabilizada por um conjunto de métodos, ferramentas e procedimentos. Desta forma, o caminho para a evolução no desenvolvimento de software passa por uma combinação de métodos abrangentes para todas as etapas de desenvolvimento do software, melhores ferramentas para automatizar estes métodos, blocos de construção mais poderosos para a implementação do software, melhores técnicas para a garantia da qualidade do software e uma filosofia de coordenação predominante, controle e administração [Pressman 2006].

Uma metodologia de desenvolvimento de um SMA deve capturar a flexibilidade, autonomia dos agentes, com variados graus de abstração, auxiliando o projetista nas tomadas de decisão relativas à análise, projeto e implementação [Rabelo 2007].

A distinção entre os requisitos envolvidos no projeto de um SMA e um sistema orientado a objetos (SOO) é evidenciada pelas próprias diferenças existentes entre agentes e objetos. Contudo, o conceito de agente apresenta semelhanças estruturais em relação ao conceito de objeto. O fato é que um agente é uma entidade que possui capacidades comportamentais e conhecimento privado, e que um objeto também possui esta mesma estrutura, demonstra a existência de características comuns entre estes conceitos [Amandi 1997].

Segundo [Shoham 1993], agentes podem ser vistos como objetos ativos com estados mentais, traçando semelhanças entre os dois conceitos, tais como a existência de troca de mensagens entre as entidades para a troca de informações e solicitação de serviços, e os conceitos de herança e agregação.

Existem distinções significativas entre agentes e objetos [Juchem and Bastos nico]:

- Agentes são autônomos. Objetos têm autonomia sobre seu estado interno, mas não exibem controle sobre seu comportamento. Assim sendo, objetos têm controle sobre como as coisas são feitas, mas não têm nenhum poder de decidir se determinada solicitação vai ou não ser atendida [Wooldrige 1999]. É preciso observar que nada impede a implementação da característica de autonomia em agentes utilizando-se técnicas OO, porém é necessário notar que a característica de autonomia não é um componente básico do paradigma OO.
- Os agentes têm comportamento autônomo flexível (reativo, pró-ativo, social). Os modelos existentes no paradigma OO não especificam nenhuma maneira de representar estes tipos de comportamento, embora isto possa ser implementado utilizando-se técnicas OO.

Existem algumas metodologias para análise e projeto de SMAs. A seguir, três metodologias serão apresentadas resumidamente.

A) Metodologia *TROPOS*

A metodologia *Tropos* [Bresciani et al. 2004] fornece suporte as atividades de análise e de projeto no desenvolvimento do sistema, desde a análise até a implementação do mesmo. Está dividida em cinco fases: fases inicial e final de requisitos, projetos arquitetural e detalhado, e implementação.

Para a análise de requisitos há duas etapas: as fases inicial e final. Na fase inicial são definidos os *stakeholders* do domínio, modelados como atores sociais, com dependências baseadas em objetivos, planos e fornecimento de recursos. Já na fase final, o modelo conceitual é estendido, inclui-se um novo ator que representa o sistema e as dependências com os outros atores do ambiente [Giunghigli et al. 2002].

As fases de projetos arquitetural e detalhado são relacionadas à especificação do sistema. A fase arquitetural define a arquitetura global do sistema em termos de subsistemas (atores), troca de dados e fluxo de controles (dependências). Também é realizado um mapeamento dos atores do sistema em um conjunto de agentes de software, cada um caracterizado com suas capacidades [Bresciani et al. 2004]. A fase de projeto detalhado especifica as habilidades dos agentes e suas interações. Cada agente é definido, mais especificamente em termos de entrada, saída, controle e outras informações relevantes para o sistema, já nesta fase é escolhida a plataforma de desenvolvimento.

A implementação se baseia na definição do projeto detalhado. Esta metodologia usa uma plataforma para agentes BDI (*Beliefs*, *Desires and Intentions* - Crenças, Desejos e Intenções), chamada *JACK* (*Jack Intelligent Agents*)[JACK 2008], para a implementação dos SMA. Os agentes em *Jack* são componentes autônomos que apresentam objetivos a serem alcançados ou eventos a serem tratados. Estes agentes são programados com um conjunto de planos, tornando-os capazes de alcançar seus objetivos.

A modelagem realizada nesta metodologia é bastante confusa e rebuscada, dificultando a fase do processo de desenvolvimento. A fase de projeto detalhado é orientada especificamente à plataforma *Jack* [Maria 2005].

B) Metodologia *GAIA*

A metodologia *Gaia* [Zambonelli et al. 2003] possui uma linguagem própria para a modelagem de SMA. O processo de desenvolvimento contém duas fases: análise e *design*. Esta metodologia tem início na fase de análise, visando coletar e organizar a especificação que servirá de base para a fase de *design*.

A fase de análise tem o objetivo de entender o sistema e decompô-lo em papéis que serão desempenhados na organização, através do modelo de papéis, e definir como os mesmos interagem, através do modelo de interação. Portanto, o modelo de papéis identifica os papéis existentes no sistema e o modelo de interação identifica um conjunto de definições de protocolos, um para cada tipo de interação entre os papéis. Esta fase de análise inclui identificar [Zambonelli et al. 2003]: as metas das organizações presentes no sistema e o comportamento esperado dos mesmos, o ambiente, os papéis iniciais, as interações iniciais e as regras que a organização deve seguir.

Os componentes gerados na fase de análise são utilizados como entrada para a fase de *design*. Esta fase pode ser decomposta em duas novas fases: a fase da elaboração da arquitetura, que inclui a definição da estrutura organizacional do sistema em termos de sua topologia e regime de controle e a identificação completa dos papéis e interações; e a fase de detalhamento que compreende a definição do modelo de agentes e a definição do

modelo dos serviços que os agentes devem oferecer para desempenhar seus papéis.

Esta metodologia precisa que o SMA seja estático, onde o número de agentes, seus comportamentos, habilidades e as inter-relações não mudam[Rabelo 2007].

C) Metodologia *PROMETHEUS*

A metodologia *Prometheus* [Padgham and Winikoff 2004] abrange desde a modelagem até a implementação de um SMA baseado na plataforma para agentes BDI. Esta metodologia é composta por três fases, onde os componentes produzidos são utilizados tanto na geração de esqueleto do código, como também para realização de testes.

A primeira fase corresponde à especificação do sistema e compreende em duas atividades: determinar o ambiente do sistema e determinar os objetivos e funcionalidades do mesmo. O ambiente do sistema é definido em termos de percepções (informações do ambiente) e ações. Além disso, também são definidos dados externos. As funcionalidades do sistema são definidas através da identificação de objetivos, da definição das funcionalidades necessárias para se alcançar esses objetivos e dos cenários de casos de uso [Dam and Winikoff 2003].

A fase seguinte é o projeto arquitetural que utiliza as saídas da fase anterior para determinar quais agentes existirão no sistema e como os mesmos irão interagir. Esta fase envolve três atividades: definição dos tipos de agentes, definição da estrutura do sistema e definição das interações entre os agentes [Padgham and Winikoff 2002].

O projeto detalhado é a última fase e é responsável por definir capacidades dos agentes, eventos internos, planos e uma estrutura de dados detalhada de cada tipo de agente identificado na fase anterior.

Atualmente existem duas ferramentas que utilizam o *Prometheus*. O *Jack Development Environment* [JDE 2008], que é um software comercial que inclui uma ferramenta de modelagem para a construção dos diagramas, resultando na geração de código na linguagem de programação *Jack* [JACK 2008]. Esta ferramenta dá suporte à metodologia *Prometheus* pelo fato dos conceitos utilizados pelo *Jack* corresponderem aos componentes gerados na fase de projeto detalhado da metodologia. A outra ferramenta é o *Prometheus Design Tool* [PDT 2008]. Esta ferramenta permite que o usuário crie e edite seu projeto utilizando os seguintes conceitos: verificar o projeto para um conjunto de possíveis inconsistências, gerar um conjunto de diagramas de acordo com a metodologia e gerar automaticamente a descrição do projeto, o que inclui descritores para cada entidade, um dicionário para o projeto e um relatório dos diagramas gerados anteriormente, além da geração de um pseudo-código em *Jack*. Mas recentemente foi implementado para o interpretador *Jason* [Bordini et al. 2007] um *plugin* [PD2JASON 2008] de geração de um pseudo-código à partir dos diagramas estruturados no PDT na Linguagem *AgentSpeak(L)*.

Para modelagem do experimento deste artigo, escolheu-se essa metodologia, visto que facilita o aprendizado de sistemas baseados em agentes, pois é bem flexível quanto ao tipo de agentes e possibilita a validação e geração de um esqueleto código [Drummond and Benzatti 2008].

3. Experimento: JogoMan

O JogoMan (acrônimo para Jogo dos Manaciais) [Adamatti 2007] é um jogo de papéis (RPG - Role Playing-Game) que tem por objetivo apresentar o problema de gerencia-

mento de água e solo em três diferentes cidades em um determinado estado, bem como o gerenciamento da pressão urbana desta região.

Este jogo é um sistema multiagentes que utiliza a arquitetura BDI para implementação de agentes com perfis comportamentais de diferentes tipos, que interagem com as pessoas de forma robusta e completa. Assim, é possível realizar uma simulação dos comportamentos destes agentes, com tomadas de decisões muito parecidos com os humanos.

A distribuição do espaço físico do JogoMan é bidimensional e apresenta 64 lotes, distribuídos nas três cidades. Cada lote, tem diferentes possibilidades de uso de solo, como agricultura ou floresta, e pertence a um jogador que possui o papel de proprietário. Quatorze jogadores desempenham diferentes papéis no jogo: 3 Prefeitos, 1 Administrador da Companhia de Água, 9 Proprietários Particulares e 1 Representante dos Sem Teto. Cada tipo de jogador tem as seguintes atribuições:

- Prefeito: cada cidade tem um prefeito, que pode ter um perfil comportamental diferente (social, econômico ou ambientalista). Os prefeitos podem construir infra-estruturas como: escolas, hospitais, postos de saúde e de polícia, além das redes de água e esgoto.
- A Proprietário Particular: cada um dos 9 proprietários possuem 5 lotes. Este jogador poderá usufruir de diferentes usos de solo, como floresta, agricultura, sítio, etc. Dependendo de seu perfil comportamental, poderá preocupar-se com retorno financeiro ou com a situação ambiental da região.
- Administrador da Companhia de Água: empresa pública que pode construir as infra-estruturas das redes de água potável e de saneamento em qualquer área nas três cidades.
- Representante dos Sem Teto: este papel tem uma função especial no jogo, pois é responsável pela alocação de um número determinado de novas famílias em cada ciclo de simulação. Estas novas famílias chegam a uma região (o que caracteriza a pressão urbana).

Conforme o tempo vai passando no jogo, vão ocorrendo as negociações entre os jogadores, sejam estes jogadores agentes de software (virtuais) ou pessoas. Todos os jogadores não têm conhecimento dos outros (reais ou virtuais). A idéia central deste jogo é a simulação destes agentes (jogadores virtuais) e jogadores reais com determinados papéis, de forma a mostrar que é possível a representação de jogadores virtuais com tomada de decisão bastante similar a de pessoas.

3.1. Modelagem do JogoMan utilizando Prometheus

Os diagramas modelados representam duas fases da metodologia Prometheus: desenvolvimento da arquitetura do sistema e desenvolvimento detalhada do sistema . Estes diagramas foram estruturados através do software *Prometheus Design Tool* (PDT).

Os componentes utilizados nos diagramas são apresentados na Figura 1 e têm a seguinte descrição:

- Agente: indivíduo externo que interage com o sistema;
- Ação: atividade realizada pelo agente;

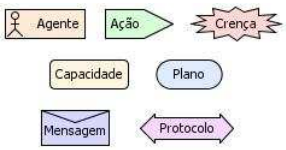

Figura 1. Componentes gráficos utlizados pelo PDT.

- Crença: informação que o agente armazena e atualiza internamente referente ao ambiente em que está inserido ou a ele próprio;
- Capacidade: o que o agente é capaz de realizar;
- Plano: conjunto de ações para se realizar um objetivo;
- Mensagem: os agentes se comunicam uns com os outros através de envio de mensagens;
- Protocolo: para envio de mensagens entre os agentes, é necessário definir como estas mensagens serão enviadas, através de um protocolo.

Na Figura 2 é apresentada a visão geral do sistema, que representa a fase de desenvolvimento da arquitetura. A Figura 2 apresenta os agentes envolvidos no jogo: **prefeito, proprietario, adm_agua-pura e o sem-teto**. Estes agentes possuem suas crenças individuais, representando os seus recursos financeiros, que são: **caixa_prefeito, caixa_proprietario, caixa_agua-pura e caixa_sem-teto** e, informações atuais sobre os lotes de cada agente, pois estes podem alterar o uso e manejo do solo destes lotes.

No caso do agente **prefeito**, este pode realizar as seguintes ações: **cria_escola, cria_hospital, cria_posto-saude, cria_posto-policia, cria_rede-agua, cria_rede-esgoto, altera_solo, altera_imposto**. Através do protocolo de interação com o agente adm_agua-pura, o agente prefeito pode negociar as infra-estruturas de redes de água e de esgoto.

Os agentes **prefeito, proprietario e sem-teto** podem interagir uns com os outros através de protocolos de **compra, venda e aluguel** de lotes, pois dependendo do comportamento de cada agente, estes poderão negociar seus lotes. Os agentes proprietario e sem-teto possuem a ação de **pagar impostos**, que lhes permiete reivindicar infra-estruturas ao agente prefeito.

A fase de desenvolvimento detalhado do sistema envolve definir uma visão individual de cada um dos agentes pertencentes ao experimento. Foram modelados diagramas para os quatro agentes existentes no JogoMan. Contudo, apenas a modelagem realizada no agente Prefeito será explanada, por restrição de espaço. Os diagramas detalhados têm por finalidade mostrar não só as crenças e as ações que os agentes podem realizar (já são representadas no diagrama de visão geral do sistema), mas também detalhar cada protocolo de comunicação que um agente possui com os demais.

A Figura 3 apresenta como o agente prefeito foi modelago utilizando PDT. O protocolo **negociacao_infra-estrutura** representado na Figura 2 é desmembrado em duas mensagens que são mostradas na Figura 3. Estas são: a mensagem **prefeito_solicita_infra-estrutura**, onde o agente prefeito a envia para o agente adm_agua-pura, solicitando a construção de uma nova infra-estrutura de rede de água e/ou esgoto; e a mensagem **agua-pura_solicita_infra-estrutura**, onde o agente adm_agua-pura envia a solicitação para o agente prefeito. Esta última mensagem dispara no agente prefeito a ativação do plano de ações **prefeito:instalar**. Este plano, por sua vez, dispara a realização

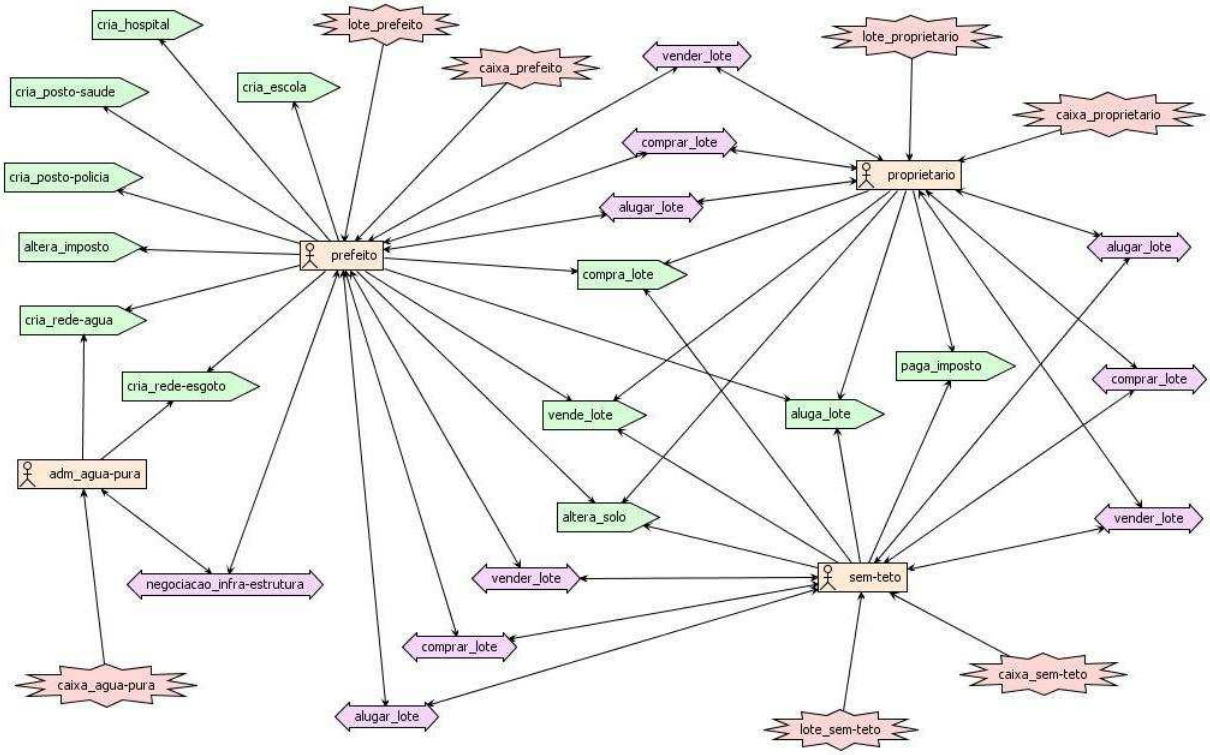

Figura 2. Diagrama da Visão Geral do Sistema.

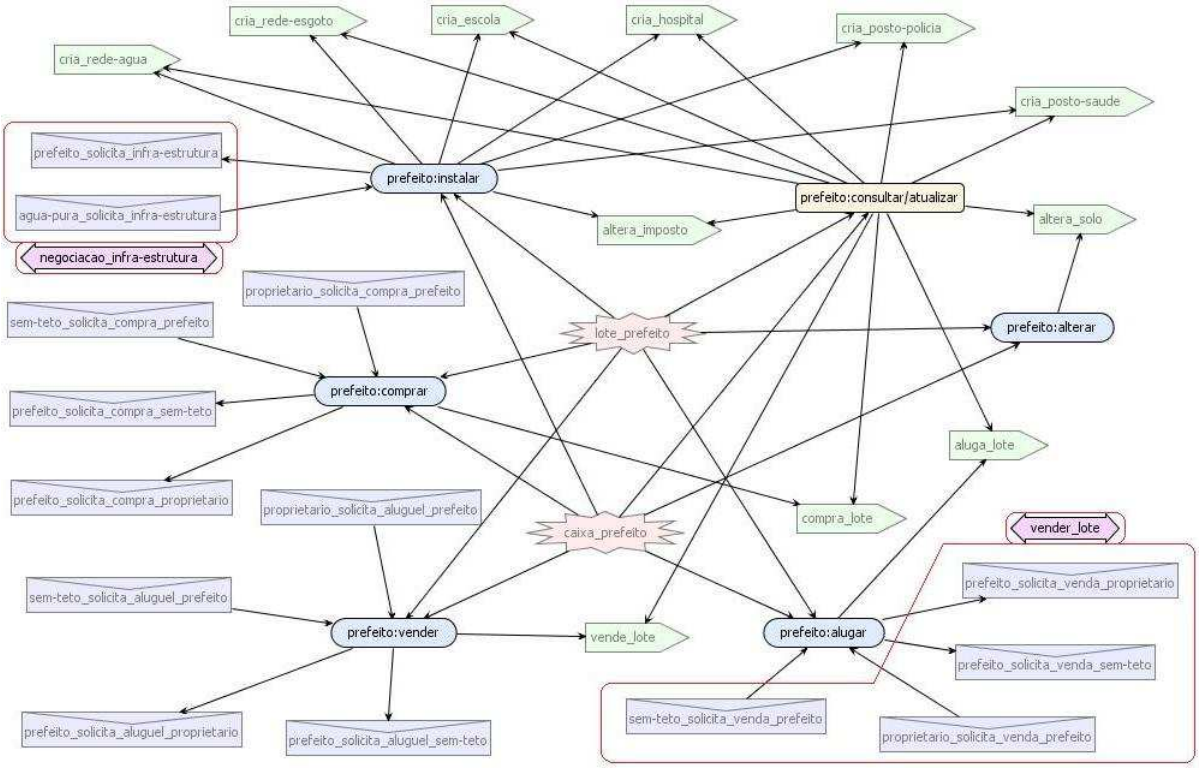

Figura 3. Diagrama da Visão Detalhada do Agente Prefeito.

das ações correspondentes ao que foi solicitado, mas antes destas ações serem realizadas, é feita uma consulta à crença caixa_prefeito, para saber se há recursos para a construção de uma nova infra-estrutura, através da capacidade **prefeito: consultar/atualizar** que o agente prefeito dispõe. Após esta consulta feita, e a realização das ações necessárias, esta mesma capacidade atualiza a crença caixa_prefeito com o novo valor calculado.

3.2. Códigos Gerados

De forma a realizar uma análise prática da modelagem realizada utilizando a metodologia Prometheus com um código gerado de forma manual para agentes BDI, utilizou-se a ferramenta Jason [Bordini et al. 2007], visto que o código dos agentes do JogoMan foram desenvolvidos nesta ferramenta. Para transformar os diagramas gerados com a ferramenta PDT, foi utilizado um plugin que dá suporte a geração de um pseudo-código que o Jason pode interpretar, denominado PD2JASON [PD2JASON 2008].

```
/* prefeito:comprar :
   Failure :
   Procedure :
   This plan don´t have one trigger.
   Verify if the message is actually of the "tell" kind.
*/
 : true
   <- ;
      +;
      +;
      .send(sem-teto, tell, );
      .

/* prefeito:vender :
   Failure :
   Procedure :
   This plan don´t have one trigger.
   Verify if the message is actually of the "tell" kind.
```

Figura 4. Parte do código do agente prefeito gerado a partir dos diagramas e do plugin PD2Jason.

A Figura 4 apresenta o pseudo-código gerado para o agente prefeito utilizando o plugin PD2JASON. Pode-se perceber que a utilização da metodologia auxilia na estruturação e organização do código, mas não na concepção em si do código. Pela parte do treco apresentado na figura, pode-se ver dois planos de ações, comprar e vender. Estes planos só possuem um cabeçalho, pois a criação do plano em si fica a critério do programador. O código também apresenta algumas mensagens, como ”Não há nenhum gatilho para este plano”e ”Verificar se a mensagem é realmente do tipo tell”, que podem auxiliar o programador a idenficar erros de modelagem e futura implementação de código.

```
//aceita proposta
+!propose(L,V,A,B,J):  value(V2)[source(self)] & .gte(V,V2)
      <- .send(B,achieve,accept_proposal(L,V,B,A,"os valores ofertados estao de acordo com os investimentos feitos
      negotiation(L,B).


//pede contra proposta
+!propose(L,V,A,B,J): count(C)[source(self)] & value(V2)[source(self)] & .gte(5,C) & value(V2)[source(self)] & .gte
      <- +count(C+1);
         -count(C);
         negotiation(L,B);
         .send(B,achieve,request(L,V,B,A,"possui infra-estrutura basica e tem valor superior")).

//rejeita proposta
+!propose(L,V,A,B,J): count(C)[source(self)] & .gte(C,5) &  value(V2)[source(self)] & .gte(V2,V)[source(self)]
      <- .send(B,achieve,reject_proposal(L,V,B,A,"os valores ofertados estao abaixo dos investimentos feitos neste

//rejeita proposta para qualquer caso que nao se encaixe
+!propose(L,V,A,B,J): true
      <- .send(B,achieve,reject_proposal(L,V,B,A,"os valores ofertados estao abaixo dos investimentos feitos neste
```

Figura 5. Parte do código do agente prefeito gerado de forma manual.

A Figura 5 apresenta o código desenvolvido para o agente prefeito de forma não automática (manual), sem utilização de nenhuma metodologia. Este código é completo e funcional.

4. Conclusões

Com base nos estudos realizados, e na geração de código baseado na metodologia Prometheus utilizando a ferramenta gráfica que dá suporte a mesma, pode-se concluir que as ferramentas que dão suporte à modelagem conceitual de sistemas, incluindo sistemas que envolvem agentes de software, são muito importantes à nível de modelagem, para que se tenha uma visão geral e detalhada do sistema. Pode-se realizar uma analogia a utilização destas ferramentas com ferramentas CASE para UML (Linguagem de Modelagem Unificada), que dão esse suporte visual ao programador e aos usuários do sistema, facilitando o entendimento do sistema e a implementação do mesmo. As ferramentas CASE para UML também geram pseudo-código em algumas linguagens, como JAVA e C++.

Contudo, a geração de código através destas ferramentas ainda é muito prematura, pois estas geram um 'pseudo-código' que ajudam na organização e padronização do mesmo. Mas estas ferramentas ainda não geram códigos executáveis, ficando a cargo do programador esta tarefa.

Referências

Adamatti, D. F. (2007). *Inserção de Jogadores Virtuais em Jogos de Papéis para Uso em Sistemas de Apoio a Decisão em Grupo: Um Experimento no Domínio da Gestão de Recursos Naturais*. PhD thesis, Universidade de São Paulo - Doutorado em Engenharia Eletrica.

Amandi, A. A. (1997). *Programação de Agentes Orientada a Objetos*. Tese de doutorado, Universidade Federal do Rio Grande do Sul.

Bastos, R. M. (1998). *O Planejamento de Alocação de Recursos Baseado em Sistema Multi-Agentes*. Tese de doutorado, Universidade Federal do Rio Grande do Sul.

Bordini, R. H., Wooldridge, M., and Hubner, J. F. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley e Sons, 225p, London.

Bresciani, P., Perini, A., Giorgini, P., Giunghiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 08(01):203–236.

Dam, K. H. and Winikoff, M. (2003). Comparing agent-oriented methodologies. In *International Bi-Conference Workshop on Agent-Oriented Information Systems*, pages 78–93.

Drummond, A. and Benzatti, D. (2008). Metodologia para o desenvolvimento de sistemas baseados em agentes prometheus. Disponível em: http://www.ic.unicamp.br/~eliane/Cursos/MO409/Curso2004(com%20MC746)/Apresentacoes/g6_a2_agentes2.ppt, acesso em: setembro de 2008.

Giunghigli, F., Mylopoulos, J., and Perini, A. (2002). The tropos software development methodology: Processes, models and diagrams. In *International Workshop on Agent-Oriented Software Engineering*, pages 162–173.

Iglesias, C. A., Garijo, M., Gonzalez, J. C., and Velasco, J. R. (1998). Analysis and design of multiagent systems using mas-commonkads. In *Lecture Notes in Computer Science, v.1365*, pages 313–327.

JACK (2008). Jack intelligent agents. Disponível em: http://www.agent-software.com/products/jack/, acesso em: setembro de 2008.

JDE (2008). Jack development environment. Disponível em: http://www.agent-software.com.au/shared/products/whatsnew50.html, acesso em: setembro de 2008.

Juchem, M. and Bastos, R. M. (Relatório Técnico). Engenharia de sistemas multiagentes:uma investigação sobre o estado da arte. 2001 014, Pontíficia Universidade Católica do Rio Grande do Sul.

Kendall, E. A., Malkoun, M. T., and Jiang, C. (1996). A methodology for developing agent-based systems. In *Lecture Notes in Computer Science, v.1087*, pages 85–99.

Maria, B. A. D. (2005). Usando a abordagem MDA no desenvolvimento de sistemas multi-agentes. Dissertação de mestrado, Pontifícia Universidade Católica do Rio de Janeiro.

Moulin, B. and Chaib-Draa, B. (1996). *An Overview of Distributed Artificial Intelligence*. John Wiley and Sons, London.

Padgham, L. and Winikoff, M. (2002). Prometheus: A pragmatic methodology for engineering intelligent agents. In *Workshop on Agent-Oriented Methodologies*, pages 97–108.

Padgham, L. and Winikoff, M. (2004). *Developing Intelligent Agent Systems: A Pratical Guide*. RMIT University, 240p, Melbourne.

PD2JASON (2008). Plugin de geração de código. Disponível em: http://downloads.sourceforge.net/jason/Pdt2Jason-0.2b.zip?modtime=1207724903&big_mirror=0, acesso em: setembro de 2008.

PDT (2008). Prometheus design tool. Disponível em: http://www.cs.rmit.edu.au/agents/pdt/, acesso em: setembro de 2008.

Pressman, R. S. (2006). *Engenharia de Software*. McGraw-Hill, 6a. edition.

Rabelo, R. J. (2007). Projeto de sistemas multiagentes. Disponível em: http://www.das.ufsc.br/%7Erabelo/Ensino/DAS6607/Aula5.pdf, acesso em: setembro de 2008.

Shoham, Y. (1993). Agent-oriented programming. In *Elsevier Science Publishers, Artificial Intelligence, v.60*, pages 51–92.

Taveter, K. (1999). *Business Rules' Approach to the Modelling, Design, and Implementation of Agent-Oriented Information Systems*. VTT Information Technology, p.317-335, Finland - FIN.

Wooldrige, M. (1999). Intelligent agents. In Weiss, G., editor, *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, chapter 1, pages 27–78. The MIT Press.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *CM Transactions on Software Engineering and Methodology*, pages 317–370.