Uma abordagem para Desenvolvimento de Sistemas Multiagentes Utilizando MAS-School, ANote e JADE

Enyo José Tavares Gonçalves
enyo@ifce.edu.br
Instituto Federal de Educação Ciência e Tecnologia do Ceará

Gilzamir F. Gomes
gilzamir@gmail.com
Universidade Estadual do Ceará

Mariela Inés Cortés
mariela@larces.uece.br
Universidade Estadual do Ceará

Robson Feitosa
robsonf@gmail.com
Universidade Estadual do Ceará

Gustavo Augusto L. de Campos
gustavo@larces.uece.br
Universidade Estadual do Ceará

Jerffeson T. de Souza
jeff@larces.uece.br
Universidade Estadual do Ceará

Resumo

Este artigo apresenta uma abordagem para o desenvolvimento de sistemas multiagente utilizando o método MAS-School, a linguagem de modelagem ANote e o framework JADE. O objetivo é fornecer um método simples para o desenvolvimento de sistemas multiagente complexos, combinando conceitos, métodos e ferramentas amplamente utilizados pela comunidade acadêmica. Para ilustrar os benefícios da abordagem, foi desenvolvido um sistema multiagente para Trading Agent Competition Supply Chain (TAC-SCM).

Palavras-chave: Sistemas Multi-agentes. Engenharia de Software Orientada a agentes. Frameworks. Linguagem de modelagem.

Abstract

This paper presents an approach to multi-agent systems development using MAS-school method, the modeling language Anote and JADE framework. The goal is to provide a simple method for the complex multi-agent systems development, combining concepts, methods and tools used by the academic community. To illustrate the approach benefits, a multi-agent system for Trading Agent Competition Supply Chain (TAC-SCM) was developed.

Keywords: Multi-agent Systems. Agent Oriented Software Engineering. Frameworks. Modeling language.

1 Introdução

No desenvolvimento de software orientado a agentes o conceito de agente é considerado como um elemento de primeira classe. Neste contexto, é de fundamental importância que as metodologias, linguagens de modelagem e linguagens de programação suportem direta ou indiretamente o conceito de agente.

Em cenários complexos como o do Trading Agent Competition (TAC) (Wellman et al., 2002), observa-se com sucesso a utilização de sistemas multiagentes, haja vista a separação das responsabilidades de cada componente de software para apoiar a resolução dos problemas intrínsecos a ambientes complexos. Portanto, este artigo propõe uma abordagem para o desenvolvimento de SMA que faz uso de técnicas e ferramentas utilizadas pela comunidade acadêmica. Para isso, é mostrado como combinar essas diferentes técnicas e ferramentas desde a fase de análise até a fase de codificação de um SMA. A abordagem utilizada adota o método MAS-School para guiar o desenvolvimento do software. Para a especificação dos artefatos produzidos nas etapas de análise, projeto e codificação foi utilizada a linguagem de modelagem ANote. Para auxiliar a fase de codificação, foi utilizado o framework JADE. (Bellifemine et al., 2007).

Outra motivação deste trabalho é compartilhar a experiência de desenvolvimento de um SMA com futuros desenvolvedores de SMA. Além disso, o mapeamento agente-objeto é importante dado que a maioria das linguagens de programação e frameworks utilizam para o desenvolvimento de SMA e é orientada a objetos.

A escolha do método MAS-School leva em conta o fato de ser focado na inserção de aprendizagem em SMA. Isso é importante porque o desenvolvimento de SMA para ambientes difíceis, estocásticos e parcialmente observáveis (Russell e Norvig, 2002) geralmente requer que todos ou parte dos agentes do sistema sejam dotados de algoritmos ou técnicas de aprendizagem de máquina. Além disso, o método MAS-School caracteriza-se como um método iterativo e incremental, possibilitando que o sistema seja desenvolvido em diversas etapas, iniciando com uma versão simples, seguido pela adição incremental de novas funcionalidades ou melhorias.

A linguagem de modelagem ANote (Choren e Lucena, 2004) foi escolhida pela sua simplicidade, expressividade e pelo fato de suportar a decomposição de objetivos; além de possuir uma interface gráfica, que facilita sobremaneira a modelagem de seus diagramas.

Apesar de ser baseado no paradigma orientado a objetos, JADE provê uma forma simples de especificação de agentes, de ontologias e outros conceitos essenciais para o desenvolvimento de SMA.

O restante deste artigo é estruturado como segue: na Seção 2 são apresentados alguns trabalhos relacionados; na Seção 3 é detalhado o mapeamento dos artefatos produzidos com ANote para objetos JADE; na Seção 4 é ilustrado um estudo de caso de um sistema multiagente para TAC-SCM. (Sadeh et al., 2003). Finalmente, na Seção 5 são apresentadas as principais conclusões sobre o trabalho desenvolvido.

2 Trabalhos Relacionados

Moraitis et al. (2002) apresentam uma tentativa de uso da metodologia Gaia a fim de conceber um SMA implementado com o framework JADE. Contudo, o conjunto de modelos provido pela metodologia Gaia não contempla objetivos explicitamente, além de seus modelos serem menos intuitivos que os modelos presentes na linguagem ANote.

A decomposição de objetivos possibilita dividir um problema complexo em subproblemas mais simples e, dessa forma, facilita a divisão de responsabilidades entre os diversos agentes do sistema. (Jennings e Wooldridge, 2000) mostram que abordagem de desenvolvimento de SMA para problemas complexos é adequada para a decomposição do problema principal em subproblemas mais fáceis de serem resolvidos. Assim, ANote possibilita visualizar essa decomposição de forma mais natural, por meio da especificação de um objetivo sistêmico e da decomposição desse objetivo em subobjetivos mais simples, formando uma hierarquia de objetivos em formato de árvore em que cada nó folha da árvore representa um objetivo funcional, ou seja, um objetivo específico que pode ser alcançado por um dos agentes do sistema.

Em Spanoudakis e Moraitis (2007), descreve-se a metodologia ASEME, desenvolvida para apoiar todas as fases do desenvolvimento de um SMA, baseada em várias outras metodologias como TROPOS (Castro, Kolp e Mylopoulos, 2002), MaSE (DeLoach, 1999) e AUML (Associates et al., 2000). ASEME é apresentada como uma combinação do que tem de melhor em várias metodologias para SMA; contudo, a ASEME não fornece um método para a inserção de aprendizado de máquina na fase de projeto do sistema.

Sardinha (Sardinha, 2005) propõe um método, o MAS-School, para a inserção de aprendizagem de máquina em SMA, contudo, utiliza o framework ASYNC e não o framework JADE para a fase de implementação de sistemas multiagente. O framework JADE, além de ser bem conhecido na comunidade acadêmica, provê funcionalidades que ASYNC não contempla diretamente, como ontologias.

O MAS-School utiliza um procedimento de avaliação das técnicas de aprendizagem de máquina. Esta avaliação consiste no desenvolvimento incremental do sistema. Primeiro, é desenvolvido um SMA somente com agentes reativos simples. O objetivo dessa primeira versão é simplesmente "testar a comunicação entre os agentes e a interação com o mundo externo". (Sardinha, 2005). Após a primeira versão, há um processo de inserção incremental de aprendizagem de máquina nos agentes do sistema. Cada nova versão do sistema pode ser resultado da inserção de um algoritmo ou alguma técnica de aprendizagem de máquina. Após inserir uma nova funcionalidade ou uma nova técnica de aprendizagem de máquina, o sistema é testado. Se for detectado que a técnica não melhorou o desempenho global do sistema, é necessária uma revisão da codificação. Após a revisão, o sistema é testado novamente. O processo continua até que se tenha obtido um resultado melhor ou até que todas as alterações necessárias para um bom desempenho do sistema tenham sido

inseridas.

A linguagem de modelagem ANote (Choren e Lucena, 2004) é adequada para o desenvolvimento de sistemas distribuídos baseados na aquisição de requisições orientadas a objetivos. A linguagem ANote provê um meta-modelo conceitual que pode ser representado como um grafo onde cada nó captura um conceito, como objetivo, ação, agente, ou cenário, e onde as arestas representam ligações semânticas entre tais abstrações. Para cada conceito, a linguagem ANote provê uma visão que captura aspectos específicos do sistema.

3 Desenvolvimento de Sistemas Multiagentes com MAS-School, ANote e JADE

O método MAS-School, a linguagem ANote e o framework JADE proveem as ferramentas necessárias para apoiar o desenvolvimento de um SMA. Essas ferramentas são utilizadas em várias etapas do processo de desenvolvimento. O método MAS-School, por exemplo, é utilizado durante todo o ciclo de desenvolvimento. Já a linguagem ANote é utilizada na fase de análise de requisitos e no projeto de um SMA. O framework JADE, na fase de codificação.

Apesar de haver uma associação conceitual do método MAS-School com a linguagem de modelagem ANote (ambos suportam diretamente os conceitos de agente e objetivo), o mesmo não acontece com o framework JADE. Este framework provê um arcabouço para o desenvolvimento de sistemas multiagente de acordo com o padrão da Foundation for Intelligent Physical Agents (FIPA).

O framework JADE possui classes e serviços que facilitam a fase de codificação de um SMA. Dentre as funcionalidades disponíveis, algumas são listadas a seguir: publicação de serviços por meio de páginas-amarelas, serviço de localização por meio de páginas-brancas, suporte a ontologias e protocolos de comunicação compatíveis com o padrão FIPA. No entanto, JADE é um framework orientado a objetos, escrito em Java. Dessa forma, o conceito de agente deve ser mapeado para objetos providos por JADE. Sardinha (Sardinha, 2005) também utiliza um framework orientado a objetos e mostra como fazer o mapeamento de agentes para objetos na fase de codificação.

Seguindo as etapas do método MAS-School, apresentadas na Figura 1, após a criação do modelo de um SMA em ANote, inicia-se a etapa de codificação. Como MAS-School é um método iterativo, a codificação pode ocorrer várias vezes durante o desenvolvimento do sistema. No entanto, após a primeira versão, as próximas codificações não afetarão sobremaneira o modelo criado. As mudanças serão referentes somente à inserção de novas estratégias e algoritmos de aprendizagem de máquina no código existente. Para essas mudanças, eventualmente deverão ocorrer alterações nos diagramas de planejamento e cenário, pois estes proveem descrições de mais baixo nível dos comportamentos dos agentes no sistema.

Figura 1: Etapas do método MAS-School.

3.1 O mapeamento de ANote para JADE

O primeiro passo do mapeamento é identificar quais elementos do meta-modelo conceitual de ANote devem ser mapeados para elementos do framework JADE. O meta-modelo conceitual da linguagem ANote é apresentado na Figura 2.

Figura 2: O meta-modelo conceitual da linguagem ANote. Fonte: (Choren e Lucena, 2004) - adaptado.

Uma versão parcial do diagrama de classes do framework JADE é apresentada na Figura 3. O conceito de agente é representado por meio do diagrama de agentes, como mostrado na Erro: Origem da referência não encontrada. Assim, agentes ANote devem ser mapeados para classes que herdam diretamente ou indiretamente da classe jade.core.Agent. Por exemplo, a Erro: Origem da referência não encontrada mostra a codificação do agente ANote Agent 1 como uma classe Agent1 que herda da classe jade.core.Agent.

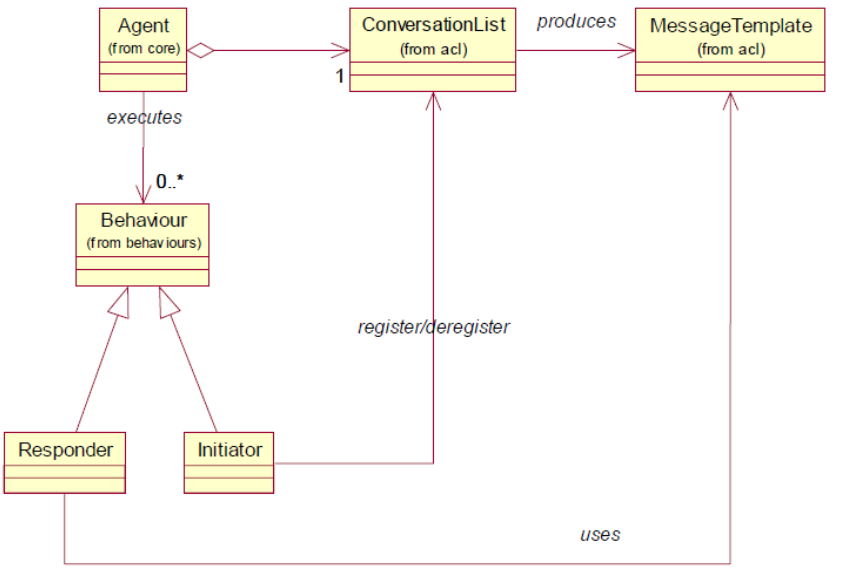

Figura 3: diagrama UML parcial do framework JADE.

A classe Agent representa uma classe base comum para a definição de agentes por parte do usuário. Portanto, do ponto de vista do programador, um agente JADE é simplesmente uma instância de uma classe Java que herda da classe

Agent. Isto implica a herança de características para suportar interações básicas com a plataforma de agentes (registro, configuração e gerenciamento remoto) e um conjunto básico de métodos que devem ser chamados para codificar comportamentos personalizados do agente, como métodos para enviar ou receber mensagens, utilizar protocolos de comunicação padrões, registro com vários domínios, entre outros.


```
package anote2jade;

import jade.core.Agent;

public class Agent1 extends Agent {
    @Override
    protected void setup() {
        super.setup();
    }
}
```

Figura 4: Representação gráfica em Anote e implementação de agentes em JADE.

O ciclo de vida do agente é definido de acordo com o padrão FIPA (Bellifemine, Caire e Greenwood, 2007). O primeiro passo é a execução do construtor do agente, seguido da atribuição de um identificador para o agente a ser inserido no sistema. O método setup (veja Figura 5) é executado a partir do momento em que o agente inicia suas atividades. Para atribuir um comportamento específico ao agente, o método setup deve ser sobrescrito.

```
public class Agent2 extends Agent {
    @Override
    protected void setup() {
        super.setup();
        addBehaviour(new ExemploComportamento());
    }
}
```

Figura 5: Atribuição de comportamento a um agente JADE.

A linguagem ANote fornece uma notação para descrição do curso de ações de um agente através de cenários os quais capturam o comportamento de um agente em um contexto específico. Um cenário é uma descrição denotando partes similares de possíveis comportamentos de agentes, comportamentos estes limitados a um contexto, ou seja, estados intencionais onde ações e interações ocorrem entre dois ou mais agentes.

Comportamentos de um agente JADE são codificados como uma subclasse da classe Behaviour. Pode-se, portanto, utilizar estas subclasses para a implementação dos cenários presentes no modelo em ANote, associando cada cenário no modelo a uma subclasse de Behaviour.

Uma classe de comportamento define basicamente dois métodos: action e done (Figura 6). A classe deve manter o estado da execução de modo que o método done retorne um valor lógico falso (equivalente ao literal false em Java) enquanto for necessário executar o método action; caso contrário, deve retornar um valor lógico verdadeiro (equivalente ao literal true em Java). A Figura 5 mostra como atribuir comportamento a um agente JADE, enquanto a Figura 6 mostra um exemplo de comportamento sobrescrevendo os métodos action e done da classe base jade.core.behaviours.Behaviour.

Deve-se observar que cada cenário é uma implementação de um objetivo funcional, derivado de objetivos mais genéricos. A decomposição começa com objetivos mais gerais e continua até que os objetivos funcionais sejam obtidos e associados a agentes capazes de alcançá-los. No entanto, mais de um cenário pode implementar um objetivo funcional. A abordagem recomendada é codificar uma classe de comportamento para cada cenário possível.

```java
class ExemploComportamento extends Behaviour {
    @Override
    public void action() {
        //Aqui deve ser implementado um comportamento para o agente
    }

    @Override
    public boolean done() {
        //Retorna verdadeiro caso seja necessário executar
        //o método action novamente.
        return false;
    }
}
```

Figura 6: Codificação de um comportamento em JADE.

Portanto, é necessário definir um modelo de cenário como uma subclasse de comportamento com os componentes de um cenário: agente líder, pré-condições, plano principal e planos variantes. Um exemplo de modelo de cenário é mostrado na Figura 7. Deve-se observar que os planos de ação (principal e variante) são representados por modelos (classes na linguagem Java) que realizam a interface ActionPlan. O mesmo ocorre com as pré-condições, cujos modelos realizam a interface Condition. Por motivos de falta de espaço, os códigos-fonte das interfaces ActionPlan e Condition não são mostrados.

Outro conceito do meta-modelo conceitual da ANote é o de recursos. Os recursos estão relacionados com a visão de ontologia da linguagem ANote. Esta visão identifica os componentes do sistema que não são agentes e, portanto, podem ser modelados como objetos.

De fato, uma ontologia define um vocabulário e um conjunto de relacionamentos entre os elementos do vocabulário. Sendo assim, o mapeamento desses componentes para modelos em uma linguagem de programação orientada a objetos é direto. Contudo, JADE tem uma forma própria para tratar ontologias.

```java
package anote2jade.core;
import anote2jade.*;
import anote2jade.agentes.cenarios.*;
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
public abstract class Scenario extends Behaviour {
    private static final long serialVersionUID = -8749959655916901761L;
    private Agent leadAgent; //o agente principal do cenário
    private ActionPlan mainActionPlan; //plano de ação principal
    private ActionPlan variantPlan; //plano de ação alternativo
    private Condition precondition = new TrueCondition(), variantPrecondition = new FalseCondition();
    private boolean done; //flag de controle, determina se o processamento deve continuar ou não
    public Scenario(Agent lead) { this.leadAgent = lead; }
    @Override
    public void action() {
        initializeConfiguration();
        if (precondition.evaluate()) {
            done = mainActionPlan.doAction();
        } else if (variantPrecondition.evaluate()) {
            done = variantPlan.doAction();
        }
    }

    @Override
    public boolean done() { return done; }
    public void setMainActionPlan(ActionPlan mainActionPlan) { this.mainActionPlan = mainActionPlan; }
    public void setVariantPlan(ActionPlan variantPlan) { this.variantPlan = variantPlan; }
    public ActionPlan getMainActionPlan() { return mainActionPlan; }
    public ActionPlan getVariantPlan() { return variantPlan; }
    public Agent getLeadAgent() { return leadAgent; }
    public void setVariantPrecondition(Condition variantPrecondition) { this.variantPrecondition = variantPrecondition; }
    public Condition getVariantPrecondition() { return variantPrecondition; }
    public void setPrecondition(Condition precondition) { this.precondition = precondition; }
    public Condition getPrecondition() { return precondition; }
    public abstract void initializeConfiguration();
}
```

Figura 7: Modelo de cenário.

Uma ontologia em JADE é uma instância da classe jade.content.onto.Ontology para a qual é definido o esquema dos tipos de predicados, ações de agente e os conceitos relevantes para o domínio de discurso a serem considerados. A criação de ontologias por meio do framework JADE é explicada em (Cabanillas, 2004).

As interações entre os agentes podem ser realizadas por meio de mensagens suportadas pelo framework JADE de acordo com o padrão FIPA. Para isso, utilizam-se linguagens de comunicação entre agentes. Uma linguagem de comunicação entre agentes é composta por: ontologia, linguagem interna e linguagem externa. O conhecimento que se deseja compartilhar é codificado por meio de uma linguagem interna. Já a linguagem externa encapsula o ato locucionário da fala, contendo informações necessárias para que a mensagem seja transportada do emissor para o receptor. As ontologias especificam vocabulários cujos termos são precisamente definidos em relação a um domínio especifico.

JADE suporta o padrão FIPA ACL (FIPA, 2002) para a linguagem externa, além de fornecer um esquema para a especificação de ontologias. O framework não especifica qual linguagem interna utilizar, ficando a critério dos projetistas e desenvolvedores. As mensagens são representadas por instâncias da classe jade.lang.acl.ACLMessage. Um objeto da classe jade.lang.acl.ACLMessage inclui as informações necessárias para o envio e entrega da mensagem. Exemplos dessas informações são: destinatário da mensagem, origem da mensagem (agente responsável por enviar a mensagem), ontologia (vocabulário utilizado na mensagem) e linguagem de conteúdo.

A linguagem de conteúdo é, geralmente, uma linguagem declarativa (Bellifemine, Caire e Greenwood, 2007) que faz uso do vocabulário expresso na ontologia de forma a possibilitar comunicação entre agentes. Observe que ontologias, linguagem de comunicação entre agentes e protocolos de comunicação estão interrelacionados.

Nessa seção foi apresentado o relacionamento entre conceitos da linguagem ANote e objetos do framework JADE. Na próxima seção, é mostrado como esse mapeamento é aplicado no desenvolvimento de um SMA para TAC-SCM.

4 Estudo de Caso

Nesta seção é apresentado um exemplo para demonstrar o mapeamento de conceitos referentes à ontologia e interações do ANote para objetos JADE, no desenvolvimento de um sistema multiagente para o torneio TAC-SCM. Devido à complexidade do sistema como um todo, foi decidido mostrar apenas a parte do sistema referente aos objetivos do agente AgentePropostaVenda. Todos os diagramas foram gerados por meio do ambiente albatroz. (Carvalho, 2005).

O primeiro passo, seguindo o método MAS-School, foi a obtenção dos requisitos sistêmicos e transformação desses requisitos em objetivos. A Figura 8 e a Figura 9 mostram os objetivos obtidos e os agentes designados para o sistema, respectivamente. Em seguida, foi definida a medida de desempenho sistêmico (lucro da fábrica) e a medida de desempenho para cada agente do sistema. No caso, para o agente AgentePropostaVenda foram atribuídos os objetivos Selecionar Pedidos de Orçamento Enviados por Clientes e Realizar Proposta de Venda para cada Cliente Selecionado.

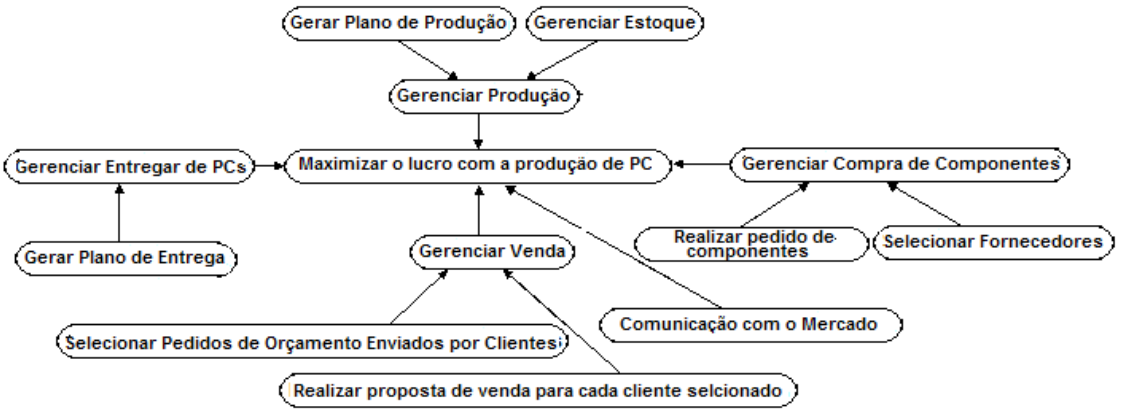

Figura 8: Objetivos do Sistema para TAC-SCM.

Observe novamente a Figura 8, o objetivo sistêmico foi decomposto em subobjetivos, formando uma árvore de objetivos. Na raiz da árvore, está o objetivo geral, que é decomposto em objetivos mais específicos. Até que objetivos específicos o suficiente para serem alcançados por determinados tipos de agentes sejam obtidos. Estes são os objetivos funcionais, localizados nos nós folhas da árvore de objetivos. O uso dos diagramas de objetivos do ANote facilitou a decomposição do sistema em agentes responsáveis por objetivos específicos. Além disso, o propósito de comunicação do diagrama e o relacionamento semântico com outros diagramas do ANote (como o diagrama de agentes) facilitou a identificação dos relacionamentos entre os agentes do sistema.

Na Figura 9 são mostrados os possíveis agentes para o sistema. As linhas conectando agentes mostram as possíveis interações entre os agentes do sistema. Observe que o agente de vendas (AgentePropostaVenda) se comunica com o agente que gerencia o estoque (AgenteGerirEstoque) e com o sensor de mercado (SensorDeMercado). Para cada agente do modelo, pode ser atribuído um ou mais objetivos funcionais. A definição correspondente de agentes em JADE é adequada para os propósitos do ANote. Para cada objetivo funcional atribuído a um agente, poderia ser especificado um comportamento associado ao agente que iria tentar alcançar o objetivo proposto. Os agentes em JADE podem receber vários comportamentos, facilitando dessa forma a associação de comportamentos aos objetivos especificados.

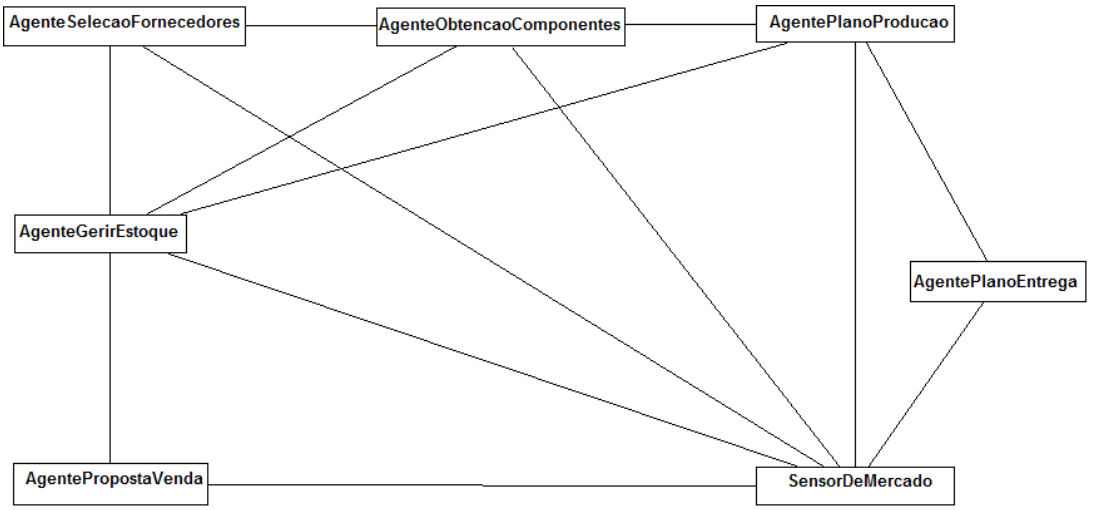

Figura 9: Agentes designados para o sistema multiagente.

A medida de desempenho para este agente é o valor utilidade do conjunto de ofertas vencidas pelo agente. A Figura 10 mostra a codificação do agente AgentePropostaVenda. Observe que além do cenário, são especificadas uma ontologia (VendaOntology) e uma linguagem de comunicação para o agente (FIPA_SL0). O mapeamento de ontologias em ANote para ontologias em JADE também é direta, pois a descrição de ontologias em ANote é realizada por meio de diagramas de classe e classes são suportadas diretamente pela linguagem Java.

```
package anote2jade.agentes;

import anote2jade.agentes.cenarios.OrcamentosDisponiveis;

public class AgentePropostaVenda extends Agent {
    /**
     *
     */
    public static final String NAME = "AgentePropostaVenda";
    private static final long serialVersionUID = -7896936490639906726L;
    public static final String REALIZAR_PROPOSTA_SERVICE = "RealizarPropostaVenda";
    public static final String SERVICE_TYPE = "AgentePropostaVendaService";

    @Override
    protected void setup() {
        super.setup();
        RealizarOfertadeComputadores scenario = new RealizarOfertadeComputadores(this);
        scenario.setMainActionPlan( scenario.new RealizarOfertaPlan(this));
        scenario.setVariantPlan(scenario.new RealizarOfertaVariantPlan());
        scenario.setPrecondition(new OrcamentosDisponiveis(this));

        addBehaviour(scenario);

        getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
        getContentManager().registerOntology(VendaOntology.getInstance(), VendaOntology.NAME);

        IAgentUtils.registerAgentInYellowPage(this, REALIZAR_PROPOSTA_SERVICE,
                SERVICE_TYPE, VendaOntology.NAME, FIPANames.ContentLanguage.FIPA_SL0);
    }
}
```

Figura 10: Implementação em JADE do agente AgentePropostaVenda.

O cenário em que o agente atua como líder é mostrado na Erro: Origem da referência não encontrada. Observe que para esse cenário, o agente AgentePropostaVenda precisa interagir com dois outros agentes: AgenteGerirEstoque e SensorDeMercado. A definição de comportamento em JADE possibilita a implementação de cenários como comportamentos que contém sequências principais de ações e sequências alternativas representando, respectivamente, o plano principal e os planos alternativos de um cenário.

Realizar oferta de computadores
LEAD AGENT
AgentePropostaVenda
PRECONDITION
disponibilidade de pedidos para orçamento
MAIN ACTION PLAN
ENQUANTO (mensagem de finalização não for recebida)
1.1. Obter últimos pedidos para orçamento enviados
1.3. Selecionar subconjunto de pedidos para orçamento com maior utilidade
1.4. Definir preço de oferta para cada pedido
1.5. Definir prazo de entrega para cada produto
1.6. Enviar ofertas para clientes
INTERACTIONS
AgenteGerirEstoque
SensorDeMercado
VARIANT PLAN
PRECONDITON (Pedidos para orçamento não disponíveis)
1. Aguardar mensagem do sensor de mercado
SE (mensagem de finalização de negociações)
2.1.1. Sair
ENTAO (mensagem com pedidos para orçamento)
2.2.1. Atualizar informações sobre pedidos para orçamento
2.2.2. Selecionar subconjunto de pedidos para orçamento com maior utilidade
ENTAO (Pedidos para orçamento disponíveis)

Figura 11: Cenário em que o agente AgentePropostaVenda atua como líder.

A Figura 12 mostra a configuração de uma mensagem a ser enviada pelo agente SensorDeMercado ao agente AgentePropostaVenda. A interação entre esses agentes se dá por meio da troca de mensagens contendo uma linguagem que faz uso de um vocabulário e regras pré-definidas. O vocabulário é expresso na ontologia de vendas, mostrada na Figura 13.

```
public boolean doAction() {
    try {
        Predicate pred = (Predicate) this.lead.getContentManager()
                .extractContent(latestMsg);
        if (pred instanceof Orcamentos) {
            Orcamentos orcamentos = (Orcamentos) pred;
            lead.getContentManager().lookupOntology(VendaOntology.NAME);
            ACLMessage msg = new ACLMessage(ACLMessage.CFP);
            AID aid[] = IAgentUtils.searchInYellowPage(lead,
                    AgentePropostaVenda.REALIZAR_PROPOSTA_SERVICE,
                    AgentePropostaVenda.SERVICE_TYPE, 1);
            if (aid != null && aid.length > 0) {msg.addReceiver(aid[0]);}
            msg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
            msg.setOntology(VendaOntology.NAME);
            lead.getContentManager().fillContent(msg, orcamentos);
            lead.send(msg);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return false;
}
}
```

Figura 12: Codificação do plano principal do cenário de EnviarPedidosOrcamento, liderado pelo agente SensorDeMercado.

As classes Proposta e todas as suas subclasses realizam a interface jade.content.Concept, o que é mostrado no diagrama da Figura 13. Já a classe SelecionaOrcamentos realiza a interface jade.content.Predicate.

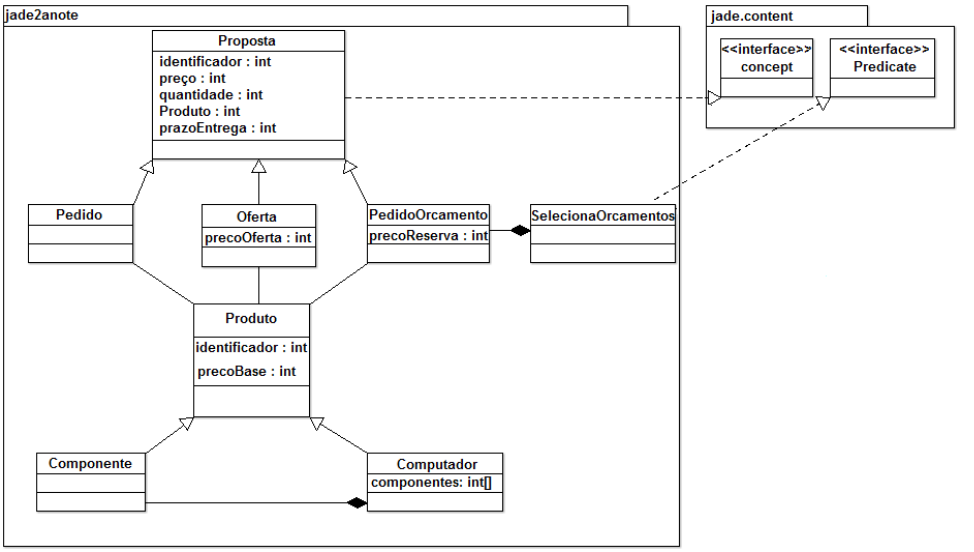

Figura 13: Modelo de uma ontologia de vendas criado por meio do editor albatroz.

Neste caso mostramos apenas a ontologia de vendas. O agente sensor de mercado recebe todos os pedidos de orçamento para um determinado dia e envia para o agente de venda uma solicitação para seleção de orçamentos que serão ofertados. A solicitação é representada pelo predicado SelecionaOrcamentos, como mostrado na ontologia de vendas. Portanto, a comunicação entre agentes é realizada sobre um vocabulário comum contendo conceitos e predicados. Os conceitos e predicados são expressos em uma linguagem declarativa, ou linguagem semântica (SL – Semantic Language) (Cabanillas, 2004), cujos codificadores e decodificadores estão disponíveis no framework JADE. Os conceitos expressos na linguagem semântica são decodificados em objetos de classes que representam conceitos e predicados. E estes objetos são codificados em expressões de uma linguagem declarativa.

A ontologia de venda em si é representada pela classe VendaOntology, mostrada na Figura 14. Os conceitos, predicados e ações presentes na ontologia são representados por classes contendo um conjunto de atributos acessados de acordo com a seguinte convenção: o valor de cada atributo é modificado por meio de um método set<nomeDoAtributo> e acessado por meio do método get<nomeDoAtributo>, onde <nomeDoAtributo> é o nome de um atributo que descreve uma característica de um conceito ou um parâmetro de um predicado.

```
package anote2jade.ontology;
import jade.content.onto.*; import jade.content.schema.*;
public class VendaOntology extends Ontology {
    private static final long serialVersionUID = 1L;
    public static final String NAME = "ontologia de vendas", PRODUTO = "PRODUTO", PRODUTO_IDENTIFICADOR = "identificador",
        PRODUTO_PRECO_BASE = "precoBase", PROPOSTA = "PROPOSTA", PROPOSTA_ID = "ID", PROPOSTA_PRECO = "preco",
        PROPOSTA_PRODUTO = "produto", PEDIDO_ORCAMENTO = "PEDIDO_ORCAMENTO", PEDIDO_ORCAMENTO_PRECO_RESERVA = "precoReserva",
        PEDIDO_ORCAMENTO_DATA_ENTREGA = "dataEntrega", PEDIDO_ORCAMENTO_QUANTIDADE = "quantidade", OFERTA = "OFERTA",
        OFERTA_PRECO_OFERTA = "precoOferta", ORCAMENTOS = "ORCAMENTOS", ORCAMENTOS_LISTA_ORCAMENTOS = "orcamentos";
    private static Ontology instance = new VendaOntology();
    public static Ontology getInstance() { return instance; }
    private VendaOntology() {
        super(NAME, BasicOntology.getInstance());
        try {
            add(new ConceptSchema(PRODUTO), Produto.class);
            ConceptSchema productSchema = (ConceptSchema) getSchema(PRODUTO);
            productSchema.add(PRODUTO_IDENTIFICADOR, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            productSchema.add(PRODUTO_PRECO_BASE, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            add(new ConceptSchema(PROPOSTA), Proposta.class);
            ConceptSchema propostaSchema = (ConceptSchema) getSchema(PROPOSTA);
            propostaSchema.add(PROPOSTA_ID, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            propostaSchema.add(PROPOSTA_PRECO, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            propostaSchema.add(PROPOSTA_PRODUTO, productSchema);
            add(new ConceptSchema(PEDIDO_ORCAMENTO), PedidoOrcamento.class);
            ConceptSchema pedidoOrcSchema = (ConceptSchema) getSchema(PEDIDO_ORCAMENTO);
            pedidoOrcSchema.add(PEDIDO_ORCAMENTO_PRECO_RESERVA, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            pedidoOrcSchema.add(PEDIDO_ORCAMENTO_DATA_ENTREGA, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            pedidoOrcSchema.add(PEDIDO_ORCAMENTO_QUANTIDADE, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
            add(new PredicateSchema(ORCAMENTOS) , Orcamentos.class);
            PredicateSchema ps = (PredicateSchema) getSchema(ORCAMENTOS);
            ps.add(ORCAMENTOS_LISTA_ORCAMENTOS, new AggregateSchema(BasicOntology.SET));
        } catch(OntologyException e) {
            e.printStackTrace();
        }
    }
}
```

Figura 14: Codificação da ontologia de venda por meio do Framework JADE.

A Figura 15 mostra um exemplo de conceito codificado de acordo com o que especifica o framework JADE. Observe que há um método get e um método set para cada atributo da classe Proposta.

```
package anote2jade.ontology;

import jade.content.Concept;

public class Proposta implements Concept {
    private static final long serialVersionUID = 1L;
    private int preco, identificador;
    private Produto produto;

    public void setPreco(int preco) { this.preco = preco; }
    public int getPreco() { return preco; }

    public void setIdentificador(int identificador) { this.identificador = identificador; }
    public int getIdentificador() { return identificador; }

    public void setProduto(Produto produto) { this.produto = produto; }
    public Produto getProduto() { return produto; }
}
```

Figura 15: Proposta faz parte do vocabulário da ontologia de vendas.

O diagrama de interação de ANote permite especificar, para um dado cenário, as interações dos agentes que participam desse cenário. O diagrama de interação da Figura 16 ilustra as interações previstas para o cenário RealizarOfertasDeComputadores. O uso de JADE facilita a codificação da troca de mensagens entre agentes provendo uma linguagem de comunicação de agentes que suporta ontologias e uma linguagem interna de descrição.

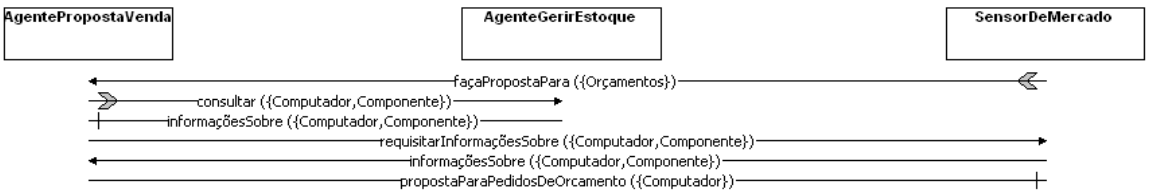

Figura 16: Diagrama de interação para o agente AgentePropostaVenda no cenário RealizarOfertaDeComputadores.

Esta seção mostrou apenas uma parte do sistema desenvolvido. O desenvolvimento dos outros cenários e agentes do sistema é realizado de forma semelhante ao desenvolvimento do cenário de realizar oferta de computadores e do agente AgentePropostaVenda. A primeira versão do sistema desenvolvido (utilizando agentes reativos) é uma versão simples do sistema para testar os aspectos de comunicação entre agentes, como recomendado pelo método MAS-School. O método MAS-School sugere que nas próximas etapas haja uma inserção gradual de métodos de aprendizagem de máquina nos agentes do sistema.

5 Conclusões e Trabalhos Futuros

A abordagem para o desenvolvimento de sistemas multiagente apresentada facilitou a identificação dos requisitos funcionais e a decomposição do problema em subproblemas mais fáceis de serem gerenciados. A combinação do método MAS-School, da linguagem de modelagem ANote e do framework JADE produz um processo de desenvolvimento iterativo, eficiente e flexível. As vantagens obtidas com a modelagem proposta são:

- ANote tem suporte aos principais conceitos necessários para a especificação de sistemas multiagente: ontologias, agentes, interações entre agentes por meio de troca de mensagens, descrição de cenários e planos de ação, entre outros. Isso facilitou a identificação dos elementos necessários para a composição do sistema, como a identificação dos agentes que compõem o sistema por meio da identificação dos objetivos funcionais.
- A codificação em JADE de um sistema multiagente modelado em ANote possibilitou um mapeamento rápido dos artefatos especificados por meio de ANote em códigos escritos na linguagem Java.
- A utilização de uma linguagem de modelagem orientada à agente facilitou a comunicação entre os membros da equipe de desenvolvimento, dado que o domínio estudado é claramente distribuído, no caso, o gerenciamento

de cadeias de suprimento. Assim, o uso de uma abstração de agentes e de comunicação entre agentes facilitou a identificação do problema principal no domínio estudado: a coordenação das decisões dos diferentes agentes que compõem o sistema. A linguagem ANote suporta a especificação de troca de mensagens entre agentes e em quais cenários estas trocas de mensagens poderiam ser realizadas. Além disso, JADE suporta linguagens que seguem o padrão FIPA, com a especificação de uma linguagem interna e com o suporte a ontologias.

- A abordagem apresentada facilitou a identificação dos requisitos funcionais e a decomposição do problema em subproblemas mais fáceis de serem gerenciados.
- A combinação de MAS-School, ANote e JADE mostrou-se um processo de desenvolvimento iterativo do ponto de vista da inserção de aprendizagem de máquina nos agentes do sistema.
- O uso de MAS-School propiciou uma experiência inicial necessária para se evitar erros em etapas posteriores no desenvolvimento.

É necessário ainda desenvolver ferramentas capazes de automatizar o mapeamento de conceitos providos por ANote para objetos do framework JADE. Portanto, como trabalho futuro é notado a necessidade de desenvolvimento de um framework que facilite a automatização desse mapeamento. Com isso, espera-se diminuir o tempo de desenvolvimento, pois parte do código será gerado de forma automática. Além disso, foi sentido falta da especificação de papéis de agentes. Assim, seria interessante uma alteração no meta-modelo conceitual do ANote para a adição deste conceito.

A primeira versão desenvolvida utiliza apenas agentes reativos, portanto a continuação deste trabalho é a inserção de aprendizagem de máquina e outras técnicas de Inteligência Artificial nos agentes do sistema.

6 Agradecimentos

Os autores agradecem a FUNCAP e CAPES pelo suporte parcial na realização deste trabalho.

Referências

BELLIFEMINE, F. *Jade programmer's guide*. [S.l.], 2007. Disponível em: <http://jade.tilab.com>. Acessado em: 10 fev. 2009.

BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. *Developing multi-agent systems with JADE*. [S.l.]: Wiley, 2007. (Wiley Series in Agent Technology).

CABANILLAS, G. C. D. JADE Tutorial - Application-Defined Content Languages and Ontologies. [S.l.], 2004, Disponível em: <http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf>. Acesso em: 10 fev. 2009.

CARVALHO, R. W. de. *Um ambiente de suporte para uma linguagem de modelagem de sistemas multi-agentes*. 2005. 155 f. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, 2005.

CASTRO, J.; KOLP, M.; MYLOPOULOS, J. Towards requirements-driven information systems engineering: the tropos project. *Information systems*, Elmsford, v. 27, n. 6, p. 365-389, sep. 2002.

CHOREN, R.; LUCENA, C. Modeling multi-agent systems with ANOTE. *Journal on Software and Systems Modeling (SoSyM)*, v. 4, n. 2, p. 199-208, springer. 2004.

DELOACH, S. Multiagent systems engineering: a methodology and language for designing agent systems. AGENT ORIENTED INFORMATION SYSTEMS (AOIS'99). *Proceedings*... [S.l.: s.n.], 1999. p. 45–57.

FIPA. *FIPA ACL Message Structure Specification*. Disponível em: <http://www.fipa.org>. Acesso em: 10 fev. 2009.

FIPA. *FIPA Communicative Act Library Specification*. Disponível em <http://www.fipa.org/specs/fipa00037>. Acesso em: 10 fev. 2009.

JENNINGS, N. R.; WOOLDRIDGE, M. On agent-based software engineering. Artificial Intelligence, v. 117, p. 277–296, 2000. Disponível em: <http://www.informatik.uni-trier.de/~ley/db/journals/ai/ai117.html>. Acesso em: 10 fev. 2009.

MORAITIS, P.; PETRAKI, E.; SPANOUDAKIS, N. Engineering JADE Agents with the GAIA methodology. In: KOWALCZYK, R. et al. (Ed.). Agent technologies, infrastructures, tools, and applications for e-services. Berlin: Springer-Verlag, 2002. p. 77–91. (Lecture Notes in Aritificial Intelligence, v. 2592).

ODELL, J. et al. Extending UML for Agents: Oriented information systems workshop. In: NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, 17., 2000, Austin. *Proceedings*... Austin: Association for the Advancement of Artificial Intelligence, 2000. p. 3–17.

RUSSELL, S.; NORVIG, P. *Artificial intelligence*: a modern approach. 2. ed. New Jersey: Pearson US Imports & PHIPEs, 2002.

SADEH, N. A supply-chain trading competition. *AI Magazine*, Menlo Park, v. 24, n. 1, p. 92–94, 2003.

SARDINHA, J. A. R. P. *MAS-School e ASYNC*: um método e um framework para construção de agentes inteligentes. 20005. 152 f. Tese (Doutorado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, 2005.

SPANOUDAKIS, N.; MORAITIS, P. The agent systems methodology (aseme): a preliminary report. In: EUROPEAN WORKSHOP ON MULTI-AGENT SYSTEMS, 5., (EUMAS'07), Hammamet, Tunisia, December 13 - 14, 2007. *Proceedings*... Disponível em: <http://www.math-info.univ-paris5.fr/~moraitis/webpapers/Moraitis-EUMAS07.pdf>. Acessado em: 10 fev. 2009.

WELLMAN, M. P. et al. The 2001 trading agent competition. *IEEE Internet Computing*, v. 13, p. 935–941, 2002. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.8865>. Acesso em: 10 fev. 2009.

Sobre os autores

Enyo José Tavares Gonçalves

Docente do Instituto Federal de Educação Ciência e Tecnologia do Ceará (IFCE) e Mestre em Ciência da Computação pelo Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará – UECE.

Gilzamir F. Gomes

Docente da Universidade Estadual Vale do Acaraú (UEVA) e Mestre em Ciência da Computação pelo Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará – UECE, tendo como linha de pesquisa Inteligência Artificial.

Mariela Inés Cortés

Doutora em Ciência da Computação pela Pontifícia Universidade Católica do Rio de Janeiro (2003) e Docente do Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará, tendo como linha de pesquisa Engenharia de Software.

Robson Feitosa

Mestre em Ciência da Computação pelo Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará.

Gustavo Augusto L. de Campos

Engenharia Elétrica pela Universidade Estadual de Campinas (2003) e Docente do Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará, tendo como linha de pesquisa inteligência Artificial.

Jerffeson T. de Souza

Ph.D. em Ciência da Computação pela School of Information Technology and Engineering (SITE) da University of Ottawa, Canadá (2004) e Docente do Mestrado Acadêmico em Ciência da Computação da Universidade Estadual do Ceará.