UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CURSO DE CIÊNCIA DA COMPUTAÇÃO

MÁRCIO JULIÃO ARAÚJO DA SILVA

ENGENHARIA DE SOFTWARE ORIENTADA A AGENTE

São Luís

2012

MÁRCIO JULIÃO ARAÚJO DA SILVA

ENGENHARIA DE SOFTWARE ORIENTADA A AGENTE

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Luciano Reis Coutinho

São Luís
2012

Silva, Márcio Julião Araújo da

Engenharia de Software Orientada a agentes / Márcio Julião Araújo da Silva. – 2012

65f.

Impresso por computador (Fotocópia).

Orientador: Luciano Reis Coutinho.

Monografia (Graduação) – Universidade Federal do Maranhão,

Curso de Ciência da Computação, 2012

1.Software – Engenharia 2. Agentes – Sistemas baseados 3. MESSAGE I Título

CDU 004.41

MÁRCIO JULIÃO ARAÚJO DA SILVA

ENGENHARIA DE SOFTWARE ORIENTADA A AGENTES

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

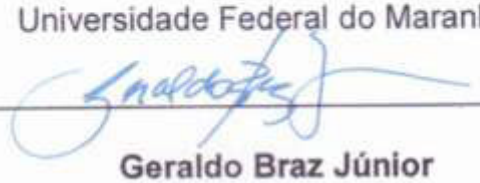
Aprovado: 18/06/2012

Banca Examinadora

Luciano Reis Coutinho (Orientador)

Doutor em Engenharia de Eletricidade

Universidade Federal do Maranhão



Geraldo Braz Júnior

Mestre em Engenharia de Eletricidade

Universidade Federal do Maranhão

Carlos Eduardo Portela Serra de Castro

Mestre em Informática

Pontifícia Universidade Católica

À minha família

Aos meus amigos.

RESUMO

Nos últimos anos, o desenvolvimento de sistemas baseados em agentes tem crescido muito, mostrando potencial na área da indústria de software. Diversas metodologias foram propostas para auxiliar nas etapas de produção do software, dentre elas a metodologia MESSAGE, a qual foi utilizada neste trabalho para modelar um sistema que registra horário de professores em um laboratório de forma autônoma. Após a modelagem, o sistema foi implementado utilizando a plataforma Jade para mostrar, de modo prático, a interação entre agentes.

Palavras - chave: Sistemas baseados em agentes. Metodologias. MESSAGE.

ABSTRACT

In recent years, the development agent-based systems has increased significantly, indicating the potential area of the software industry. Several methodologies have been proposed to assist in the production stages of the software, among then the MESSAGE methodology, which was used in this work to model system that records time teacher in a lab unattended. After modeling, the system was implemented using the Jade platform to display, in a practice way, the interaction between agents.

Keywords: Agent-based systems. Methodologies. MESSAGE.

AGRADECIMENTOS

A Deus, aos meus pais Raimundo Julião e Alzenir Silva, aos meus irmãos Cosmo, Jorge, Juliana, à minha cunhada Fábia, aos meus sobrinhos Kayllane, Kawanne e Lucas, e todos que me deram apoio a terminar esta etapa da minha vida.

Aos professores do Departamento de informática e de outros departamentos que me fizeram ver os assuntos da área de computação e me identificar com a matéria.

Ao meu orientador Luciano que me auxiliou nesta etapa final do meu curso.

Aos meus amigos e colegas, em especial Robinson Castro e Regina que sempre serão parte da minha família.

A Hamisson pelo apoio moral e pela força em acreditar em mim.

E a todas as pessoas que direta ou indiretamente me ajudaram a chegar aqui.

LISTA DE FIGURAS

LISTA DE SIGLAS

AMS – Agent Management System (Sistema de Gerenciamento de Agente)

AOM – Agent Oriented Methodology (Metodologia Orientada a Agente)

AOSE – Agent Oriented Software Engineering (Engenharia de Software Orientada a Agente)

APIs – Application Programming Interfaces (Interfaces de Programação para Aplicação)

AUML – Agent-based Unified Modeling Language (Linguagem de Modelagem Unificada baseada em Agente)

BDI – Beliefs, Desires and Intentions (Crenças, Desejos e Intenções)

CASE – Computer Aided Software Engineering (Engenharia de Software Auxiliada por Computador)
DF – Directory Facilitator (Facilitador de Diretório)

FIPA – Foundation for Intelligent Physycal Agents (Fundação para Material de Agents Inteligentes)

JADE – Java Agente Development framework (Plataforma de Desenvolvimento de Agente em Java)

KQML – Knowledge and Query Manipulation Language (Linguagem de Manipulação de Consulta e Conhecimento)

MAS – Multi-Agent System (Sistema Multiagente)

MASIF – Mobile Agent System Interoperability Facility (Facilidade de Interoperabilidade em Sistemas com Agentes Móveis)

MESSAGE – Methodology for Engineering Systems of Software Agents (Metodologia para Engenharia de Sistemas de Software baseados em Agentes)

OMG – Object Management Group (Grupo de Gerenciamento de Objeto)

RUP – Rational Unified Process (Processo Racional Unificado)

SLA – Specification Language Agent (Linguagem Específica de Agente)

UML – Unified Modeling Language (Linguagem de modelagem unificada)

Sumário

1 INTRODUÇÃO

Com o surgimento de sistemas mais complexos, mais autônomos, tem sido proposto um novo paradigma de desenvolvimento orientado a agente, que pode tomar decisões, realizar negociações na ausência de usuários humanos e até mesmo de outros sistemas, e que tenta suprir esta demanda comercial.

Ao mesclar técnicas tradicionais de orientação a objeto e inteligência artificial foi possível a criação e desenvolvimento de uma nova área da computação conhecida como Engenharia de Software orientada a Agentes. Essa área tem como objetivo apresentar os processos de construção do software voltados para integração dos conceitos de agentes como papéis, objetivos, serviços, sociedade e interação.

Engenharia de software baseado a agentes constitui uma área relativamente nova da computação que tem ganhado impulso nos últimos anos, devido ao rápido desenvolvimento das tecnologias que necessitam de novas formas de resolução de tarefas. Devido à natureza dinâmica, esse domínio impõe uma série de desafios que são enfrentados por diversos grupos ao redor do mundo. Esta nova concepção apresenta diversas ramificações quanto à funcionalidade e aplicação, como serviços de informação, comércio eletrônico, serviços de apoio e emergência.

O objetivo deste trabalho é apresentar o conceito e uma aplicação da engenharia de software baseada em agente, tendo como foco principal a interação entre agentes com negociação para se alcançar um objetivo.

Como exemplo de implementação, apresenta-se um sistema baseado em agente chamado “Registra Horário”. Esse sistema foi desenvolvido em Java sob a plataforma NetBeans, utilizando as APIs (Application Programming Interface) do JADE.

O restante do trabalho está organizado nos seguintes capítulos:

O capítulo 2 apresenta o conceito e os processos de engenharia de software.

O capítulo 3 explica a tecnologia de agente, assim como a abordagem AUML, FIPA e algumas das principais metodologias como Gaia, Tropos e Prometheus.

O capítulo 4 mostra a metodologia MESSAGE escolhida para modelar o sistema apresentado neste trabalho.

O capítulo 5 apresenta a modelagem do sistema “Registra Horário” através da metodologia MESSAGE.

O capítulo 6 mostra o projeto desenvolvido.

O capítulo 7 apresenta a conclusão do trabalho.

1.1 Trabalhos Relacionados

Mesmo sendo uma área relativamente nova, a Engenharia de Software orientada a Agente tem sido objeto para vários pesquisadores ao redor do mundo, sendo propostas metodologias, frameworks, plataformas e sistemas.

Em (ZAMBONELLI, et al, 2003), é apresentado a metodologia Gaia com seus conceitos básicos, metaforizando um organização, mostrando as fases do desenvolvimento e seus respectivos modelos.

2 ENGENHARIA DE SOFTWARE

Segundo Pressman, (2005), a Engenharia de Software é uma disciplina que orienta os desenvolvedores de software a modelar os requisitos que compõe as necessidades dos clientes, tornando possível um mapeamento para construir o sistema. A Engenharia de Software está preocupada com a praticidade do desenvolvimento e entrega de um software útil. Já o autor Sommerville, (2007), diz que a Engenharia de Software é um ramo da engenharia cujo foco é o desenvolvimento dentro de custos adequados de sistemas de software de alta qualidade.

A Engenharia de Software é uma área da computação que trata da produção do software do início até a manutenção, fornecendo meios para desenvolver o software de forma organizada e garantindo qualidade e confiabilidade de maneira econômica.

O conjunto de atividades que resultarão no software é chamado de Processo de Software. Dependendo do sistema, há um processo específico para ele. Entretanto, há quatro atividades fundamentais que todo processo deve ter, que são: Especificação, Desenvolvimento, Validação e a Evolução do Software. A Especificação ou Engenharia de Requisitos permite identificar quais serviços o sistema necessita para executar suas funcionalidades e suas restrições de operação e de desenvolvimento. O Desenvolvimento tem a fase de Projeto que envolve vários modelos do sistema em diferentes níveis de abstração, e a fase de Implementação, em que é feita a conversão da especificação do sistema em sistema executável, ou seja, a codificação das funcionalidades. A Validação verifica, através de inspeções e revisões, se o sistema está conforme com a sua especificação, ou seja, com o que foi acertado antes, e com as expectativas do cliente. A Evolução do Software que consiste em mudar o software de acordo com novas exigências ou manutenção.

Para Sommerville, (2007), o Processo de software tem sua representação abstrata através de um modelo de Processo de software. Para cada modelo de software tem-se um processo com uma determinada perspectiva e com informações parciais sobre o processo.

O Modelo de Software também de chamado de Ciclo de Vida do Software envolve as seguintes fases: Planejamento, Análise e especificação dos requisitos, Projeto, implementação, Teste, Entrega e Implementação, Operação e Manutenção.

O Planejamento fornece uma estrutura que permita fazer estimativas razoáveis de recursos, custos e prazos, sendo atualizado assim que o projeto progride.

A Análise e Especificação de Requisitos realiza o levantamento de requisitos do domínio do problema, para identificar a funcionalidade e o comportamento esperado do sistema, para serem modelados, avaliados e documentados.

O Projeto aplica os requisitos tecnológicos aos requisitos essenciais ao sistema, necessitando que a plataforma de implementação seja conhecida, envolvendo duas etapas que são o Projeto da Arquitetura do sistema que define a arquitetura geral do software, baseando-se no modelo construído na fase de análise de requisitos, e o Projeto Detalhado que detalha o projeto do software para cada componente identificado na etapa anterior, refinando esses componentes em níveis de maior detalhamento até que possam ser codificados e testados.

A Implementação onde ocorre a tradução do projeto para uma forma passível de execução pela máquina, cada unidade de software do projeto detalhado é implementada.

Os Testes incluem diversos níveis de testes como teste de unidade, teste de integração e teste de sistema, cada unidade de software implementada dever ser testada e os resultados documentados. Os componentes devem ser integrados sucessivamente até se obter o sistema, finalmente o sistema é testado como todo. Para Somerville, (2007), o teste possui duas fases: demonstrar para o desenvolvedor e para o cliente que o software atende aos requisitos, e descobrir falhas ou defeitos no software que apresenta comportamento incorreto, não desejável ou em não conformidade com sua especificação.

Entrega e Implementação que estabelece que o software satisfaça os requisitos do usuário, instalando o software e conduzindo testes de aceitação (validação).

Operação onde o software é utilizado pelos usuários no ambiente de produção.

Manutenção que fará mudanças no software por erros encontrados, ou adaptação para futuras mudanças, podendo levar a um novo processo, onde cada fase precedente é reaplicada no contexto de um software existente ao invés de um novo.

Alguns dos modelos são: Modelo de Cascata, o Modelo de Desenvolvimento Evolucionário e a Engenharia de Software baseada em Componentes. O Modelo de Cascata determina que cada fase do processo é separada, passando para a próxima fase somente depois que terminar a fase atual por completo, deixando explícito que no início, os requisitos foram entendidos de forma clara pelos desenvolvedores. O Modelo de Desenvolvimento Evolucionário permite a produção rápida de um sistema inicial, baseada em especificações abstratas, para que depois seja refinado mediante a aprovação do cliente, fazendo com que ocorra a intercalação das atividades, até que o sistema desejado seja obtido. E finalmente, a Engenharia de Software baseada em Componentes dispõe de componentes reusáveis, privando o sistema da construção do zero ao integrá-los através de um framework.

A Engenharia de Software lida com sistemas complexos e conta com o auxílio de ferramentas que apoiam as atividades do processo. As chamadas CASE (Computer-Aided Software Engineering) fornecem a automação das atividades através de ferramentas como editores de textos, sistemas de depuração interativos, linguagem de alto nível, geradores de interface com o usuário, etc.

Outra parte essencial da Engenharia de software é o Gerenciamento, que segundo Sommerville, (2007), um bom Gerenciamento não garante um bom software, mas um mau gerenciamento geralmente resulta na falha de projeto. Realizando as atividades de gerenciamento como elaboração da proposta, custo do projeto, seleção e avaliação do pessoal, planejamento de projeto, cronograma do projeto e o gerenciamento de riscos, garante-se que o software não sofra atraso, que não custe mais do que foi estimado e não falhe ao atender os requisitos.

3 TECNOLOGIA DE AGENTES

3.1 Agente

Segundo Wooldridge (2002), infelizmente não há uma definição universal aceitável do termo agente, e existem muitos debates e controvérsias acerca deste assunto. Para os autores Wooldridge e Jennings (1995), agente é um sistema de computador que está situado em algum ambiente, e que é capaz de ações autônomas neste ambiente a fim de realizar seus objetivos de projeto.

Agente é uma entidade computacional que está situada em um ambiente, só ou com outros agentes, onde o mesmo pode perceber o ambiente, notando mudanças, percebendo a existência de outros agentes, além de poder interagir com o ambiente e com os outros agentes, para executar suas tarefas de projetos de forma independente, e também cooperando junto com os outros agentes, para juntos atingirem um objetivo global para o sistema o qual foram desenvolvidos.

Segundo (BERGENTI et al., 2004), um agente não tem controle total sobre o seu ambiente e sim um melhor controle parcial no que cabe a ele influenciá-lo, o que para o agente, significa que uma mesma ação executada duas vezes em circunstâncias aparentemente idênticas, podem ter efeitos completamente diferentes ou até mesmo falhar no efeito desejado. Por essa razão, o ambiente se torna não determinístico.

Um agente possui um repertório de ações disponíveis para ele. Este conjunto de possíveis ações representa a capacidade eficácia do agente que é a habilidade para modificar seu ambiente. Mesmo possuindo um conjunto de ações, nem todas podem ser executadas, pois existem casos onde determinada ação não poderá ser realizada. Por isso, ações têm pré-condições associadas, as quais definem os momentos possíveis nas quais elas podem ser aplicadas. Cabe ao agente decidir qual ação ele executará a fim de satisfazer melhor seus objetivos. Na Figura 1 temos um agente que tem o objetivo de manter a temperatura da sala agradável. A sala é o seu ambiente, e como se encontra muito quente, a ação que o agente realizará será ligar o ar condicionado, ou seja, para executar a ação de ligar

o ar condicionado, a pré-condição do ambiente está muito quente tem que ter sido estabelecida antes.

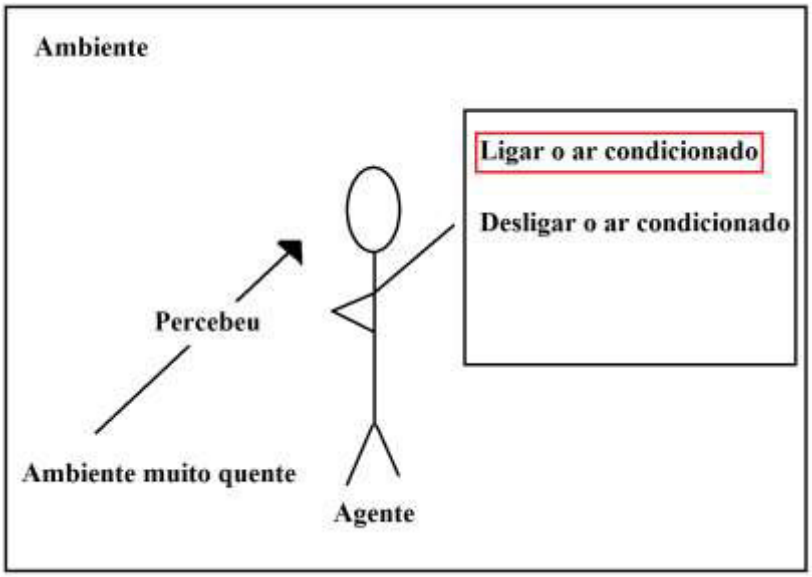


Figura 1 Um agente em seu ambiente.

Segundo (BERGENTI et al., 2004), um agente possui estado mental que é representado em termos do que ele sabe (ex.: suas crenças) e o que ele está atualmente perseguindo (suas intenções). Algumas diferenças entre esses modelos para representar a execução do estado são: os agentes não podem manipular o estado de outro agente diretamente, mas podem afetá-lo somente através da comunicação; possuem uma representação explícita de seus objetivos; têm um conhecimento explícito do seu ambiente, incluindo outros agentes no ambiente; Exceto por um atributo único, os agentes não têm atributos públicos. Um agente pode pedir a outro para executar um objetivo (passagem de mensagem declarativa -> até podem explicitamente dizer como).

Além de serem apropriados a aplicações incomuns, fazendo uso de suas características avançadas, como aprendizagem e autonomia, os agentes também representam uma alternativa válida para outras tecnologias sólidas, por produzirem um alto nível de abstração, possuindo vantagens de reuso.

Na fase de análise da engenharia de software, existem vários tipos de requisitos, como requisitos do usuário, requisitos do sistema, requisitos funcionais e não funcionais. Os agentes se concentram mais em requisitos funcionais (o que o sistema tem que fazer, ou descrições de como algumas computações devem ser

realizadas) e requisitos não funcionais (requisições que restringem o sistema, requisições externas).

Os agentes possuem propriedades que lhes permitem executar suas atividades, perceberem o ambiente ao seu redor, agirem segundo ações sobre eles, e estas propriedades são: Proatividade permitindo a perseguição de novos objetivos; Sociabilidade habilitando a interação com outros agentes enviando e recebendo mensagens; Reatividade que permite responder em tempo as mudanças no ambiente; e Autonomia que permite um agente operar sem supervisão.

3.2 Sistema Multiagente

Um sistema com vários agentes é denominado de sistema multiagente. Com isso tem-se um sistema com entidades agindo em prol de um objetivo global ou agindo concorrentemente para atingir seus interesses próprios. Em sistemas multiagentes os agentes formam uma sociedade, com suas regras estruturadas para garantir uma existência pacífica, harmoniosa e coordenada, como exemplificado na Figura abaixo.

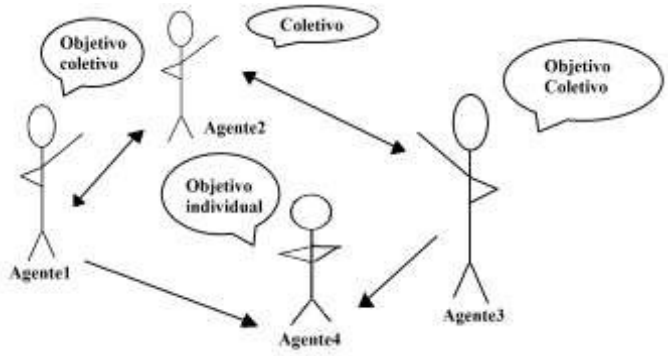


Figura 2 Sistema multiagente

Segundo Shehory e Sturm, (2001), as características de um sistema multiagente são: Autonomia, Complexidade, Adaptabilidade, Concorrência, Distribuição, Comunicação. Para Wooldridge, (2002), as relações entre os agentes são:

Independência – onde não há dependência entre agentes.

Unilateral – um agente depende de outro, mas são vice-versa.

Mútua – os agentes envolvidos dependem uns dos outros para o mesmo objetivo.

Dependência Recíproca – um agente depende de outro para um objetivo e este outro agente depende do primeiro para outro objetivo.

3.3 Abordagem AUML

AUML (*Agent-based Unified Modeling Language*) é uma linguagem com o propósito de oferecer aos desenvolvedores uma notação que seja usada para análise, projeto e implementação de sistemas multiagente conforme (BERGENTI et al., 2004). Muitas metodologias acabam caracterizadas com extensões de sistemas orientados a objeto, por isso os diagramas utilizados pela AUML foram reusados da UML (Unified Modeling Language), adequando-se perfeitamente para o desenvolvimento desses sistemas. Quando esses diagramas não podem ser simplesmente reusados, ocorre a extensão dos diagramas UML, como o diagrama de classe de agente e o diagrama de sequência.

O Diagrama de classe de agente representa os agentes e sua arquitetura, englobando seu conhecimento, planos e protocolos usados. O Diagrama de Sequência mostra as interações entre os agentes e as mensagens trocadas por eles em tempo de execução, possuindo duas dimensões, a vertical que ordena as mensagens de cima para baixo de acordo com o eixo do tempo, e a dimensão horizontal que representa as instâncias ou papéis. Este Diagrama foi muito utilizado devido ele ter sido adotado pela FIPA (*Foundation for Intelligent Physycal Agents*) para expressar os protocolos de interação.

3.4 FIPA

FIPA é uma associação internacional sem lucro de companhias e organizações compartilhando o esforço para produzir especificações para tecnologias genéricas de agentes (BERGENTI et al., 2004). O Padrão FIPA define o modelo de referência de uma plataforma de agente e um conjunto de serviços que deveriam ser produzidos. A coleção desses serviços, e suas interfaces padronizadas, representam as regras que permitem uma sociedade de agente existir, operar e gerenciar. O padrão identifica os papéis de alguns agentes necessários para gerenciar a plataforma, e descrever a linguagem e a ontologia do

gerenciamento. Algumas linguagens de desenvolvimento de agentes que estão relacionadas com a FIPA são: *Agent Development Kit, Lightweight Extensible Agent Plataform*, ZEUS e o JADE.

3.5 Ferramentas e Plataformas

Segundo (BERGENTI et al., 2004), uma plataforma de agente pode ser entendida como um conjunto de serviços que permite um gerenciamento e comunicação do agente. Uma plataforma de agente pode vir com componentes de agente onde o desenvolvedor pode reusar para seu sistema.

Existem plataformas e ferramentas que ajudam na criação dos agentes, com uma estrutura que facilita sua implementação. Algumas ferramentas encontradas são:

a) Jack que é uma ferramenta comercial que utiliza os conceitos BDI, desenvolvida pela Agent Oriented Software Pty. Ltd *(*BUZETTA, 1999*)*.
b) Zeus é uma ferramenta para construção de agentes (Nwana, 1999). Desenvolvida pela British Telcom, é um framework baseado em Java e usa a linguagem KQML para o desenvolvimento de agentes colaborativos ou cooperativos (BIGUS, 2001).
c) FIPA–OS é uma plataforma baseada em Java com implementação open source (BIGUS, 2001).
d) ASL é uma plataforma de agente que ajuda o desenvolvimento em C/C++, Java, JESS, CLIPS e Prolog (KERR et al., 1998).
e) Grasshopper é uma plataforma de agente móvel baseada em Java puro, ajustando-se com padrões existentes, como definidos por OMG-MASIF (Mobile Agent System Interoperability Facility) (MILOJICIT et al., 1998) e especificações FIPA. Por isso Grasshopper é um plataforma aberta, permitindo interoperabilidade máxima com outros agentes móveis e sistemas de agentes inteligentes, e fácil integração com CORBA e serviços Java e APIs.

3.6 A plataforma JADE

JADE (*Java Agent Development framework*) é um framework para ajudar o desenvolvimento de aplicações de agentes na submissão com a especificação para sistemas multiagente inteligentes e operáveis (BELLIFEMINE et al., 2001). Foi desenvolvido pela Universidade de Parma na Itália, consistindo de um conjunto de classes que implementa um sistema de gerenciamento do agente, um facilitador de diretório e um canal de comunicação de agente (BELLIFEMINE et al., 2003).

Os agentes localizam-se nos contêineres do JADE, onde fica o ambiente de execução. O conjunto de contêineres ativos é chamado de Plataforma. Quando iniciada uma execução, o contêiner principal é ativado, assim que novos contêineres forem necessários, terão que se registrar no contêiner principal. Havendo outro contêiner principal em algum lugar na rede, ele constituirá em uma diferente plataforma para o qual os outros contêineres normais terão que se registrar. A Figura 3 mostra os agentes, os contêineres e as plataformas em JADE, podendo haver comunicação em um mesmo contêiner, os agentes A2 e A3, em diferentes contêineres da mesma plataforma, A1 e A2, ou em plataformas diferentes, A4 e A5.

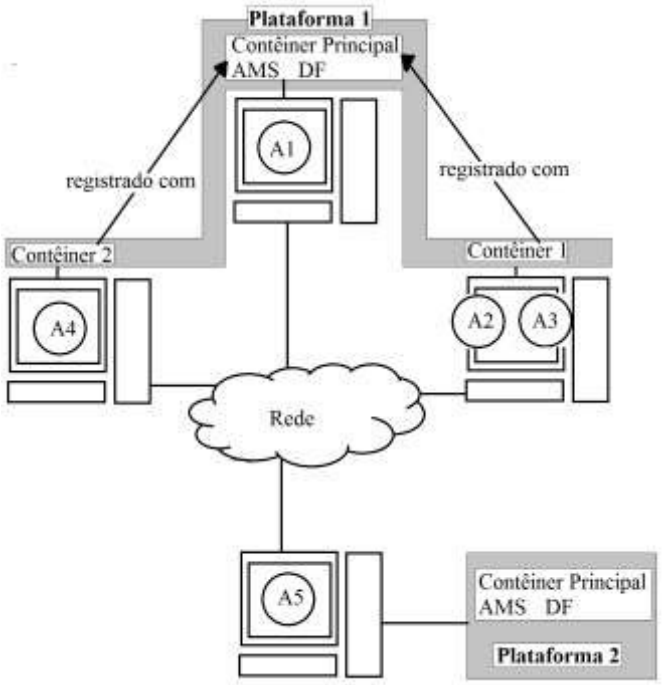


Figura 3 Contêineres e Plataformas do JADE.

No contêiner principal, há dois agentes especiais: MAS (*Agent Management System*) e DF (*Directory Facilitator*).O MAS fornece o serviço de nomeação, permitindo por exemplo, que cada agente na plataforma, tenha um único nome. O DF produz o serviço de páginas amarelas por meio do qual um agente pode encontrar serviços fornecidos por outros agentes. A Figura 04 mostra a tela de interface do JADE.

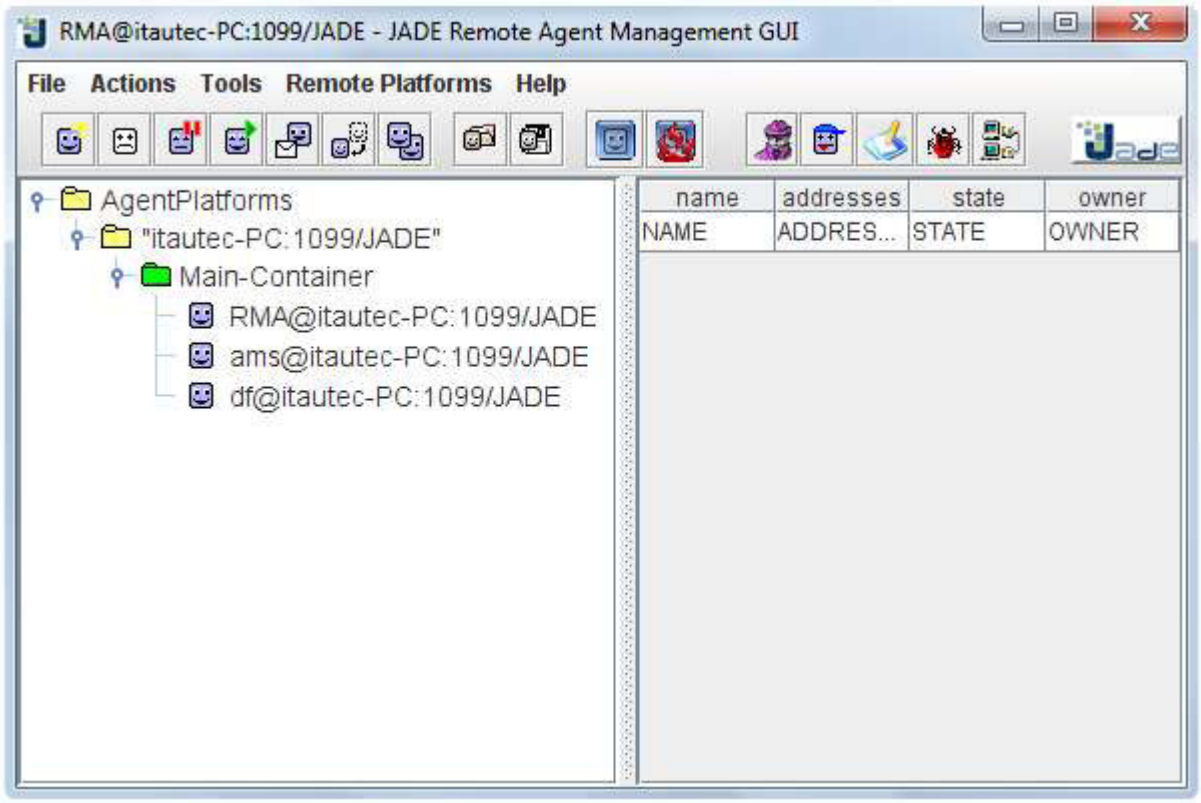


Figura 4 Tela da Interface do JADE.

3.7 Metodologias

Uma metodologia geralmente fornece ferramentas, que são associadas com a notação, a fim de ajudar na parte gráfica e para executar uma validação de consistência. Como essas metodologias orientadas a agente e multiagente são um novo campo de pesquisa e esses paradigmas estão associados a novos conceitos (agente, ambiente, tarefa, serviço, organização, interação, etc.), surgem novas ferramentas, novos modelos para auxiliar o desenvolvedor a tratá-los. As principais metodologias se concentram nas três primeiras fases do processo: requisitos, análise e projeto.

3.7.1 A metodologia Gaia

Segundo Shehory e sturm, (2001), Gaia estende AOM (*Agent Oriented Methodology*), com o foco em aspectos de modelagem em sistemas baseados em agente. Gaia foi a primeira metodologia completa proposta para guiar o processo de desenvolvimento da análise ao projeto de sistemas multiagentes (BERGENTI et al., 2004), mas não se preocupa com a implementação, deixando para o projetista esta parte.

Gaia possui duas versões onde a primeira se limitava a sistemas fechados, além de adotar técnicas não padronizadas, possuindo cinco modelos: O Modelo de Papel, o Modelo de Interação, o Modelo de Agente, o Modelo de Serviço e o Modelo de Conhecimento. Os dois primeiros são produzidos na fase de análise e apresentam o sistema como um conjunto de papéis abstratos interagindo. Cada papel possui os atributos Permissão, Responsabilidade, Atividade e Protocolo que são definidos a fim de formar seu esquema. O atributo Permissão especifica quais recursos podem ser usados para executar o papel e quais recursos restringem o executor do papel, segundo Shehory e Sturm, (2001). A responsabilidade determina a funcionalidade do papel, expressa em termos de propriedades de segurança, que fixam condições verdadeiras para todo o sistema, e propriedades vitais, que segundo (BERGENTI et al., 2004), descrevem o ciclo de vida ou comportamento geral do papel. As atividades representam as tarefas ou ações que um papel pode

executar. E os Protocolos são tarefas ou ações que um papel leva envolvendo interações com outros papéis.

Os modelos de agente, de serviço e de conhecimento são produzidos na fase de projeto, formando uma completa especificação do projeto do sistema multiagente para ser usado pela fase de implementação. O modelo de agente especifica os tipos de agentes que formarão o sistema atual. O modelo de serviço especifica os serviços que são implementados por esses tipos de agentes. E o modelo de conhecimento descreve a comunicação entre os tipos de agentes, conforme a Figura 5.

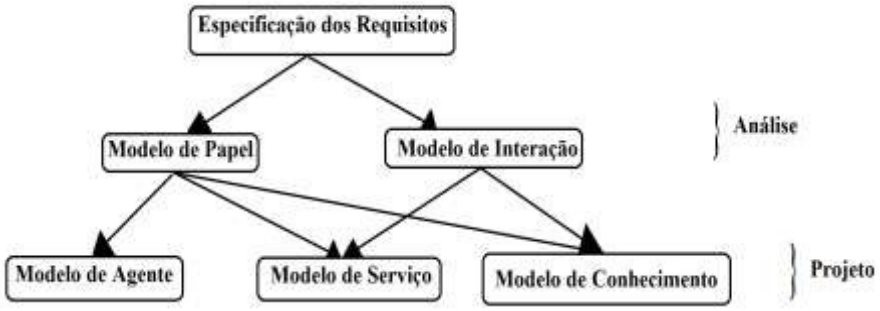


Figura 5 Modelos da metodologia Gaia

A segunda versão é uma extensão da primeira, que visou superar suas limitações, tornando-a apropriada para sistemas complexos e abertos. Considera uma organização como sendo mais que uma simples coleção de papéis, adicionando abstrações organizacionais, conforme a Figura 6.

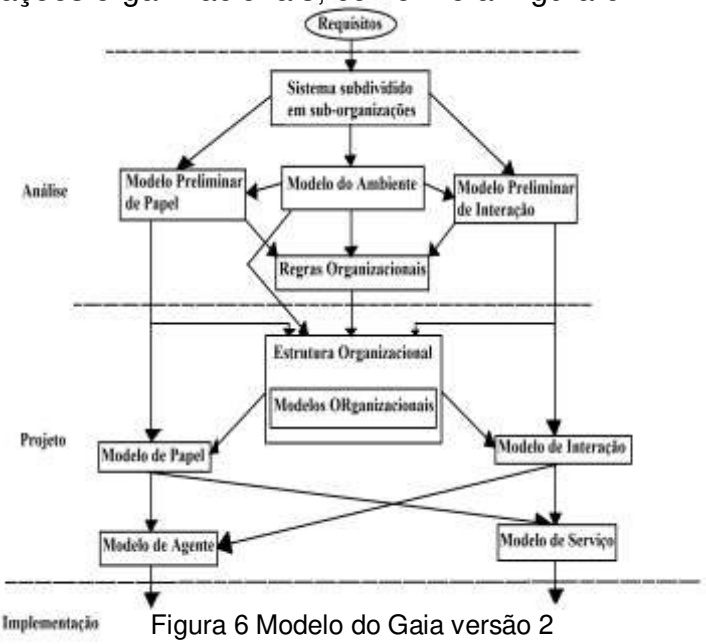


Figura 6 Modelo do Gaia versão 2

Segundo Bayer e Svantesson, (2001), as regras organizacionais devem vir depois de ter as habilidades básicas da análise, requeridas pela organização, assim como as interações básicas que são requeridas para a exploração dessas habilidades. A implementação e/ou uma organização computacional terão de respeitar um número de restrições. Isso acontece entre papéis e protocolos. Uma restrição global não pode facilmente ser expressa em termos de papéis individuais ou protocolos de interação individual. Para capturar este tipo de restrição usa-se o conceito de regras organizacionais.

Estrutura organizacional – trata-se de uma escolha que não deveria ser antecipada durante a fase de análise. O modelo de papel define toda a estrutura organizacional, incluindo a topologia do modelo de interação e o regime de controle das atividades da organização. Em uma organização, há sempre relacionamentos inter-agente: controle, benevolência, pares, dependências, posse, etc. Além do modelo de ambiente, duas abstrações a mais vêm nesta segunda versão, que são as regras organizacionais e estruturas organizacionais.

Na fase de análise da segunda versão do Gaia os objetivos da organização constituem todo o sistema e seu comportamento global. O modelo ambiental representa o ambiente em termos de variáveis e recursos computacionais. O modelo de papel preliminar é incompleto, com algumas interações inter-papel não identificadas. O modelo de interação preliminar abstrai da estrutura organizacional. E um conjunto de regras organizacionais que governam a organização em seu comportamento global, restringindo a execução de atividades de papéis e protocolos.

A fase de projeto inclui as seguintes subfases: Definição de toda arquitetura do sistema, da estrutura organizacional; um grande número de estruturas organizacionais diferentes para melhorar a eficiência e requisições funcionais; Revisão e conclusão dos modelos preliminares de papel e de interação, baseado na estrutura organizacional adotada; Definição do modelo de agente especificando os tipos de agente e instâncias; e Definição de modelos de serviços para especificar os principais serviços (Blocos de atividades com suas pré-condições e pós-condições) que os tipos de agentes têm que produzir.

3.7.2 A metodologia Tropos

Tropos é uma metodologia de desenvolvimento de software orientada a agente com o foco na análise de requisitos inicial, onde o domínio dos stakeholders e suas intenções são identificados e analisados. Este processo de análise permite a racionalidade para desenvolver o software. O processo de desenvolvimento do Tropos consiste de cinco fases: Requisitos iniciais, Requisitos finais, projeto arquitetural, projeto detalhado e implementação, Dam e Winikoff, (2003).

A fase de requisitos iniciais é influenciado pelo framework i* de Eric Yu, que segundo (BERGENTI et al., 2004), inclui o modelo de dependência estratégica para descrever a rede de relações entre atores, e modelo de razão estratégica para descrever e ajudar a racionalidade de cada ator através de suas relações com outros atores. Tropos usa o conceito de ator para modelar os stakeholders e objetivos para modelar suas intenções, e estes objetivos são divididos em Hard-objetivos que eventualmente levam aos requisitos funcionais e Soft-objetivos relativos aos requisitos não funcionais. Há dois modelos que representam objetivos e atores sobre este ponto na metodologia. Primeiro, o diagrama de ator descreve os stakeholders e suas relações em seu domínio. O segundo, o diagrama de objetivos mostra a análise de objetivos e planos com relação a um ator específico que tem a responsabilidade de alcançá-los. Objetivos e planos são analisados baseados sob muitas técnicas de racionalidade. A fase dos requisitos finais envolve estender os modelos os quais foram criados na fase anterior. A importância desta fase é a modelagem do alvo do sistema dentro do seu ambiente. O sistema é modelado com um ou mais atores. Suas dependências com outros atores no modelo contribuem para a conclusão do objetivo do stakeholder. Por isso, essas dependências definem os requisitos funcionais e não funcionais do sistema.

O projeto arquitetural é definido por três passos. No primeiro passo, novos atores são incluídos e descritos por um diagrama de ator. O segundo passo identifica as capacidades dos agentes. O terceiro passo agrupa os agentes para formar tipos de agente, onde cada tipo de agente é formado por unir alguns números de capacidades.

O projeto detalhado envolve definir a especificação de agentes no nível micro. Há três diagramas que o projetista precisa produzir para descrever as capacidades, os planos dos agentes e a interação entre eles. O Tropos usa o

diagrama de atividade da UML para representar capacidades e planos no nível detalhado. Diagramas de Planos são representações de cada agente no plano no diagrama de capacidade. A interação entre agentes no sistema é representada pelo diagrama de interação de agentes.

Na fase de Implementação é escolhida uma plataforma BDI (*Beliefs, Desires and Intentions*), especificamente Jack agentes inteligentes, para implementação de agentes. Jack produz cinco principais construções de linguagens: Agentes, capacidades, banco de dados relacional, eventos e planos. Neste estágio, desenvolvedores precisam mapear cada conceito na fase de projeto para construir esses cinco em Jack. O Tropos produz muitas normas e heurística para mapear conceitos Tropos para conceitos BDI e conceitos BDI para construções Jack.

3.7.3 A Metodologia Prometheus

Segundo Dam e Winikoff, (2003), o Prometheus é uma metodologia de Engenharia de Software Orientada a Agente (AOSE) detalhada mais apropriada a desenvolvedores que não dominam esta área. Ele consiste de três fases:

A fase de especificação – Aqui, como primeiro passo, deve-se determinar o ambiente do sistema em termos de percepções que são informações vindas do ambiente, e em termos de ações que são os meios que o agente afeta o seu ambiente. Como segundo passo, deve-se determinar as funcionalidades do sistema que é a identificação dos objetivos, identificando funcionalidades que alcançam esses objetivos e definindo cenários de caso de uso que mostram o sistema em operação.

A fase de projeto arquitetural – Como primeiro passo desta fase, temos a definição de tipos de agente que são agrupados de acordo com as funcionalidades requeridas para cumprir seus trabalhos e todos os dados identificados com a ajuda do diagrama de união de dados, e o diagrama de conhecimento do agente que representa cada tipo de agente no sistema. Como segundo passo, temos o projeto da estrutura global do sistema que é capturado pelo diagrama de visão geral do sistema, produzindo uma imagem geral do sistema de como ele funcionará, mostrando os tipos de agente, a comunicação entre os agentes e os dados. E como último passo desta fase, temos a definição da interação entre agentes que, segundo protocolos de interação, capturam o comportamento dinâmico do sistema ao definir

as sequências de mensagens válidas entre agentes, segundo (BERGENTI et al., 2004), usando diagrama de interação desenvolvidos através dos cenários.

Segundo (BERGENTI et al., 2004), a terceira fase é de projeto detalhado que se preocupa com a natureza de cada agente e como eles alcançarão suas tarefas dentro do sistema. Concentra-se em definir capacidades, eventos internos, planos e estrutura de dados detalhado para cada tipo de agente. A capacidade do agente é descrita via um descritor de capacidade que contém informação de quais eventos são gerados e quais são recebidos.

4 A METODOLOGIA MESSAGE

Dentre as metodologias estudadas, o MESSAGE (*Methodology for Engineering Systems of Software Agents*) se destacou por ser uma metodologia mais completa em relação às outras, utilizando a UML como ponto de partida, adicionando elementos necessários à modelagem orientada a agente. Esta modelagem busca descrever a forma como um sistema multiagente funciona para a realização de um objetivo coletivo, similarmente às organizações humanas e sociedades, além do comportamento cognitivo dos agentes.

Segundo (BERGENTI et al., 2004), o MESSAGE é uma metodologia que produz uma linguagem de modelagem gráfica, um processo de desenvolvimento e normas de como aplicar a metodologia à qual cobre as fases de análise e projeto, além de explorar as relações para implementação, teste e distribuição.

A Linguagem de Modelagem do MESSAGE – Estende os conceitos básicos de UML de Classe e associação com o nível de conhecimento de conceitos centrais do agente.

Agente – Um agente é uma entidade atômica autônoma que é capaz de realizar algumas (potenciais) funções úteis (BERGENTI et al., 2004). Esta capacidade funcional é capturada como um serviço do agente. Um serviço é como o nível do conhecimento análogo de operações do objeto. A qualidade de autonomia significa que uma ação de um agente não é diretamente dedicada somente a eventos externos ou interações, mas também por suas próprias motivações. Esta motivação é capturada como um atributo nomeado propósito. O propósito influenciará se um agente concorda com uma requisição para realizar um serviço e também a maneira que ele produz o serviço.

Organização – Uma organização é um grupo de agentes trabalhando juntos para um propósito comum (BERGENTI et al., 2004). Seus serviços são produzidos coletivamente por seus agentes constituintes. Possui estrutura expressada através de relações de poder entre os constituintes, e mecanismos de comportamento/coordenação expressados através de interações entre constituintes.

Papel – Descreve as características externas de um agente em um contexto particular. Um agente pode ser capaz de executar muitos papéis, e muitos

agentes podem ser capazes de executar o mesmo papel. Papéis podem ser usados como referências indiretas a agentes. Isto é útil na definição de modelos reusáveis.

Objetivo – Está associado com estado do agente. Se um objetivo instanciado é apresentado na memória do trabalho do agente, depois o agente pretende induzir sobre o estado referenciado pelo objetivo.

Tarefa – É um tipo de atividade do MESSAGE. Tem um conjunto de pares de situações descrevendo pré-condições e pós-condições. Se a tarefa é realizada quando uma pré-condição é válida, então ela pode supor a pós-condição associada para garantir que a tarefa seja completada. Tarefas compostas podem ser expressas em termos de sub-tarefas ligadas casualmente (que podem ter diferentes executores da tarefa pai). Tarefas são máquinas de estado, assim, diagramas de atividade da UML podem ser usados para mostrar as dependências temporais de sub-tarefas.

Interação e Protocolo de Interação – O conceito do MESSAGE de interação é outro tipo de atividade. Uma interação por definição tem mais de um participante, e um propósito que os participantes coletivamente devem mirar para alcançar. Um protocolo de interação define um modelo de troca de mensagem associada com uma interação.

4.1 Visões e Modelos de Análise do MESSAGE

O modelo de análise do MESSAGE é uma cadeia complexa de classes inter-relacionadas e derivadas de conceitos definidos no meta-modelo MESSAGE. Há definido cinco visões que concentram o envolvimento de subconjuntos de conceitos de entidades e relações: Organização, Objetivo/Tarefa, Agente/Papel, Interação e Domínio.

Visão de Organização – Esta visão mostra entidades concretas como agentes, organizações, papéis, recursos (como banco de dados e aplicações de serviços), no sistema e seu ambiente e relações de granularidade entre elas (agregação, poder e relação de conhecimento). Como exemplo, imagine uma universidade. A universidade é uma organização, seus funcionários são seus agentes, as funções desempenhadas pelos funcionários são seus papéis. Os meios utilizados para seus serviços são os recursos.

Visão de Objetivo/Tarefa – Esta visão mostra em detalhe os objetivos que o agente/papel possui e as tarefas que eles realizam para alcançá-los. No exemplo

dado da universidade, cada funcionário da administração terá um objetivo na organização como chefe de departamento, professor, secretário, diretor.

Visão de Agente/Papel – Esta visão se concentra nos agentes individuais e papéis, descrevendo suas características, como quais objetivos eles são responsáveis, quais eventos eles precisam perceber, quais recursos eles controlam, as regras de comportamento necessárias, etc. Como exemplo, o professor precisa ministrar aulas em determinada carga horária, adotando determinados livros, escolhendo uma maneira de avaliar seus alunos.

Visão de Interação – Esta visão mostra, para cada interação entre agente/papel, o iniciador, o colaborador, o motivador (geralmente um objetivo pelo qual o iniciador é responsável), a informação relevante fornecida/alcançada por cada participante, os eventos que provocam a interação, e outros efeitos relevantes da interação como um agente se tornando responsável por um novo objetivo.

Visão de Domínio – Esta visão mostra o domínio de conceitos e relações específicas que são relevantes para o sistema em desenvolvimento.

Essas cinco visões podem ser graficamente visualizadas nos seguintes diagramas que serão mostrados no próximo capítulo, o diagrama de Organização, Objetivo/Tarefa, Delegação e Interação, os quais estendem o diagrama de classe UML, e o fluxo de trabalho que estende o diagrama de atividade UML.

4.2 O Modelo de Projeto do MESSAGE

O modelo de projeto do MESSAGE refina o modelo de análise. Ele fornece a interação detalhada do agente descrevendo a comunicação inter-agente e informação trocada entre agentes e com seu ambiente. O modelo de projeto também fornece meios através do qual o agente pode identificar outros agentes com os quais se comunicar. Modela a organização do agente. Isto cobre a capacidade dos agentes para cooperar com outros agentes para resolver problemas. A Visão de Agente do Modelo de Projeto descreve o comportamento e a estrutura interna do agente. O projeto interno dos agentes individuais é descrito em termos de componentes reusáveis. Construções para descrever conceitos básicos são definidas: observação, ação, comunicação, objetivos, planos, etc. Esses conceitos de projeto são relatados para corresponder conceitos no modelo de análise, mas abordam questões sobre como esses conceitos seriam implementados. O Projeto de

MAS também leva em conta escolha da plataforma e aderência a um padrão como FIPA. Embora em princípio o projeto de alto nível seria independente de uma plataforma específica, abstrações restringem o projeto a uma família de plataformas específicas. O processo de projeto do MESSAGE é baseado na seleção para cada agente de uma arquitetura de agente, o refinamento do modelo de análise dentro do modelo de projeto, e a alocação de elementos do modelo para a arquitetura do agente.

4.3 O Processo de Projeto e Análise

O MESSAGE adota o RUP (*Rational Unified Process*), que é dirigido ao desenvolvimento de sistemas orientados a objetos, como seu framework de ciclo de vida genérico de engenharia de software, concentra-se nas atividades de análise e projeto, que são atividades de modelagem. A principal produção de cada atividade é um modelo do sistema em um nível apropriado de abstração.

Da análise, teremos uma especificação do sistema, que é o modelo de análise, para interpretar o problema a ser resolvido e seus requisitos, representado como um modelo abstrato que vise:

a) Entender melhor o problema

b) Confirmar que é o problema certo para resolver (validação)

c) Facilitar o projeto da solução

Por isso deve está relacionado ambos a declaração de requisitos e para o modelo de projeto (uma descrição abstrata da solução). Os objetivos de alto nível são decompostos e satisfeitos em termos de serviços fornecidos pelos papéis. As interações entre papéis que são necessários para satisfazer os objetivos são também descritas. Os modelos de análise são produzidos por refinamento de passos.

Na fase de projeto, temos uma divisão em projeto de alto nível e projeto de baixo nível. O de alto nível refina o modelo de análise, determinando papéis a agentes e descreve como os serviços são fornecidos em termos de tarefa que podem ser decompostas em ações diretas, ações comunicativas para envio e recebimento de mensagens, usando protocolos de interação. Já o de baixo nível considera que existe uma plataforma específica, na qual o desenvolvedor pensa em

implementações onde os conceitos do projeto de alto nível serão mapeados em diferentes elementos computacionais.

Em cada estágio, o desenvolvedor precisa realizar um refinamento dos passos do modelo. O MESSAGE tem definido para análise algumas estratégias de refinamentos. Para estruturar o refinamento, o MESSAGE propõe níveis de refinamento. Diferentes níveis são numerados começando do nível 0. Cada nível inicia com um conjunto de elementos que são modificados usando diferentes estratégias de refinamento. O nível seguinte contém informação sobre o sistema com um grau de abstração inversamente proporcional ao número de nível.

O nível 0 está preocupado com a definição do sistema para ser desenvolvido com respeito a seus stakeholders e o ambiente. O sistema aparece como um conjunto de organizações que interagem com recursos, atores ou outras organizações. Atores podem ser usuários humanos ou outros agentes existentes. Os estágios subsequentes do refinamento resultam na criação de modelos no nível 1, nível2 e assim por diante.

No nível 0, o processo de modelagem inicia construindo a organização e as visões de Objetivo/Tarefa. Essas visões depois agem como entrada para criar as Visões de Agente/Papel e Domínio. O modelo neste nível dá uma visão global do sistema, seu ambiente e suas funcionalidades. A granularidade do nível 0 foca na identificação de entidades e suas relações de acordo com o meta-modelo. Os seguintes níveis determinam a estrutura e o comportamento da entidade como organização, agentes, tarefas, objetivos e entidades de domínio. Níveis adicionais poderiam ser definidos para analisar aspectos específicos do sistema, lidando com requisitos funcionais e requisitos não funcionais como perfomance, distribuição, tolerância a falhas e segurança. Desde que cada nível expanda conceitos aparecidos nos níveis anteriores.

5 SISTEMA DE REGISTRO DE HORÁRIO

Como vimos no capítulo anterior, a metodologia MESSAGE fornece uma boa estrutura para modelar um sistema e será utilizada para modelar o sistema Registra Horário. O objetivo da modelagem é organizar os agentes dentro do sistema, determinar quais agentes irão interagir entre si e com quais outras entidades, suas tarefas, objetivos e recursos que serão utilizados.

Descrição do estudo de caso: Registra Horário

Contexto. O Controle acadêmico quer melhorar seus serviços ajudando seus clientes (professores e alunos) a obter uma interação direta com os horários, registrando os horários que os professores desejam, informando aos alunos a nova grade de horários.

Requisitos. Dado o fato que muitos professores e alunos têm a disponibilidade de terminais wireless, a eficiência do processo de reservar um horário e consultá-lo pode ser melhorada por desenvolver um sistema (utilizando os terminais wireless e network terrestre) que:

a) Marcar um horário para o professor
b) Informar ao aluno a grade de horário

Objetivo. O objetivo do sistema Registra Horário para o nosso estudo é apresentar como os agentes se relacionam entre si para a obtenção de um objetivo, global ou individual, fornecendo aos professores uma maneira automática de reservar suas aulas através dos recursos disponíveis pelos agentes.

A modelagem apresentada aqui seguirá os passos do capítulo anterior, apresentando os diagramas do nível 0 e nível 1. Começaremos no nível 0: Análise. Aqui dois diagramas aparecem, o diagrama de organização (relação estrutural) e o diagrama de organização (relação de conhecimento).

No diagrama de Organização (relação estrutural) temos apenas que o sistema Registra Horário é considerado parte da organização do Controle Acadêmico, que contém outras estruturas existentes, e será analisado no nível 1. Neste nível nenhum agente é modelado. Observe a Figura 7.

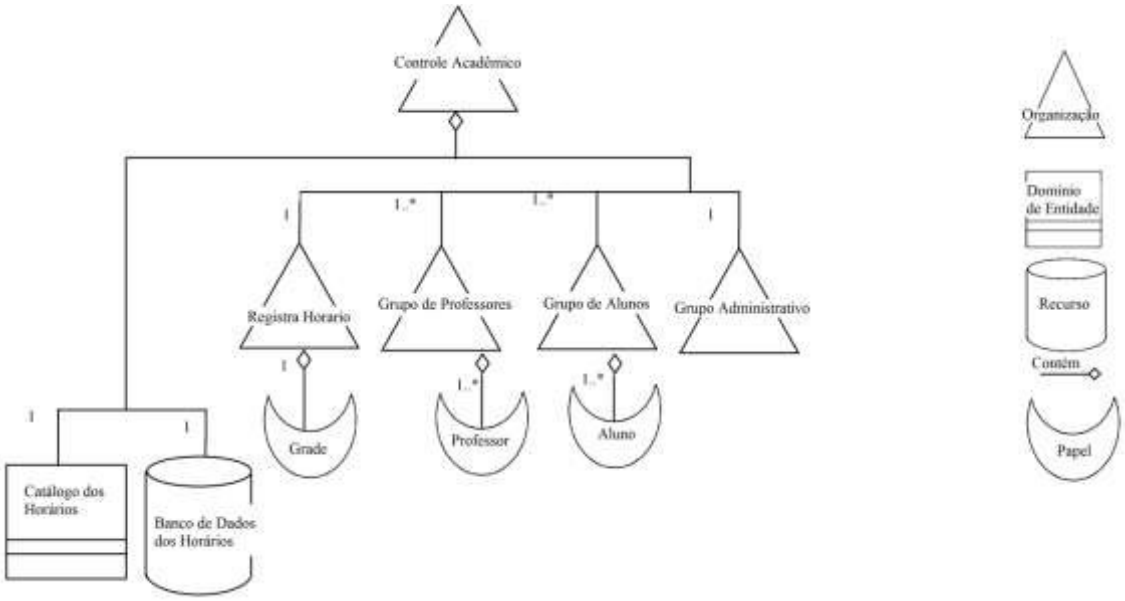


Figura 7 Diagrama de Organização nível 0 (relação estrutural).

A Figura 8 mostra o diagrama de organização nível 0 (relação de conhecimento). O sistema Registra Horário interage com o grupo Administrativo, com o papel Grade, com o recurso Banco de Dados dos Horários para armazenar e recuperar os horários registrados. O papel Grade interage com o Professor para poder reservar o horário, e com o papel Aluno para fornecer os horários.

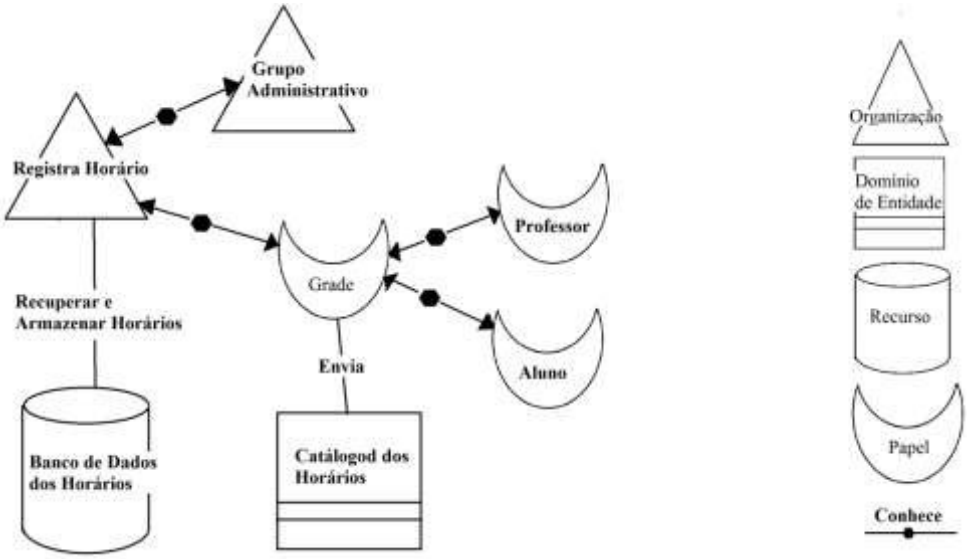


Figura 8 Diagrama de Organização nível 0 (relação de conhecimento).

O diagrama de objetivo do nível 0 mostra os objetivos comuns que são desejados. Na Figura 9, temos que o principal objetivo do sistema que é Registrar Horário, é satisfeito quando o objetivo Horário Registrado foi alcançado. O objetivo Horário Registrado é satisfeito quando ocorre um objetivo Registrar seu Horário e um objetivo Reservar Horário. Os objetivos Fornecer Horário e Consultar Horário são satisfeitos quando o objetivo Horário Registrado é satisfeito.

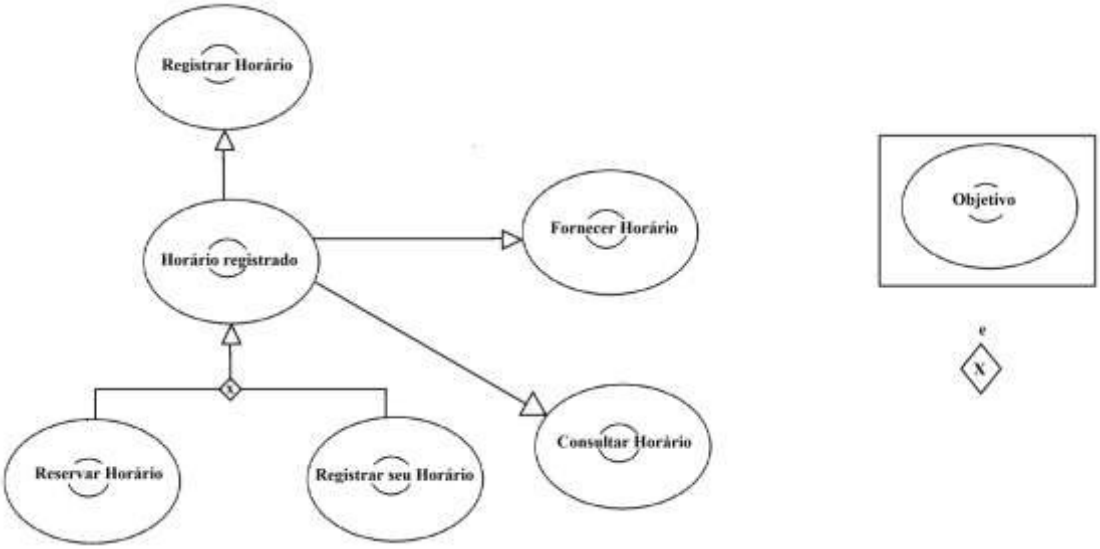


Figura 9 Diagrama de Objetivo/ Tarefa

Vamos agora para o nível 1: Análise. O sistema Registra Horário tem dois tipos de usuário, um professor e um aluno, ambos interagem com o papel Grade. Depois de atender a solicitação do professor, o papel da Grade interage com o Banco de Dados, modelado como recurso, para reservar o horário. E para atender a solicitação do papel do aluno, o papel da Grade recupera do Banco o horário. Tem-se também a interação do papel do professor com outro professor, conforme a Figura 10.

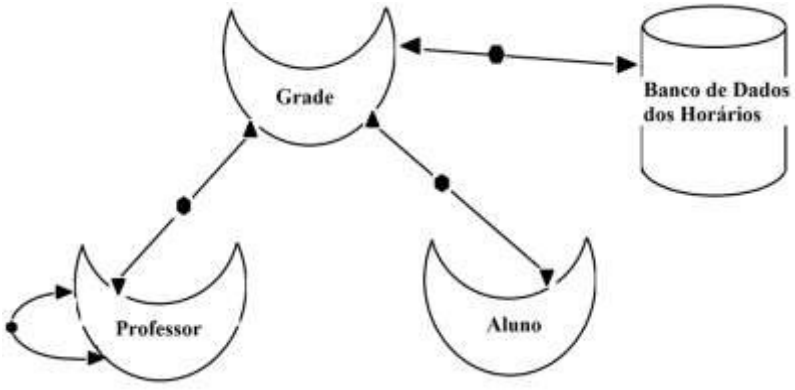


Figura 10 Diagrama de Organização, Papéis e suas interações.

No diagrama de delegação temos os objetivos da raiz e das folhas mostrados no diagrama de objetivo, juntos com seus respectivos papéis. O objetivo Registrar Horário é satisfeito pelo o objetivo Registrar seu Horário do papel Professor e pelo objetivo Reservar Horário do papel Grade. Os objetivos Fornecer Horário e Consultar Horário são satisfeitos pelo objetivo Reservar Horário. Veja a Figura 11.

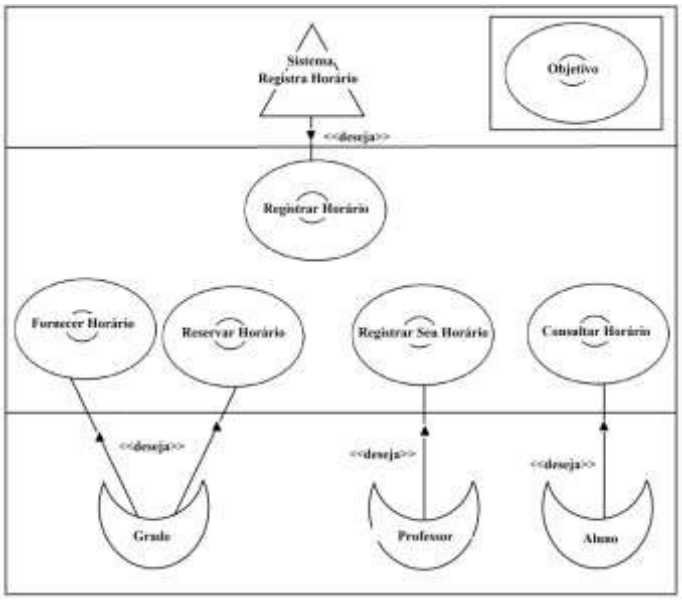


Figura 11 Diagrama de Delegação

No diagrama de interação representamos a interação de agentes, um iniciador e um colaborador, ambos com funções que se complementam para a obtenção de um objetivo. O iniciador mostrado na Figura 12 é o Professor que faz um pedido do horário à Grade. O colaborador representado pela Grade reserva o horário e envia a confirmação deste ao Professor.

Figura 12 Diagrama de Interação entre o Professor e a Grade.

Outra interação é vista na Figura 13, o iniciador é o Professor que quer fazer uma negociação de um horário já ocupado pelo colaborador, outro Professor. O colaborador interage com a aceitação da negociação.

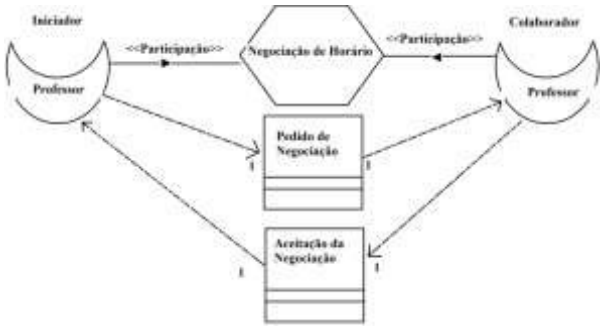


Figura 13 Diagrama de Interação entre dois Professores.

Por fim, na Figura 14 temos a interação na qual o iniciador é o Aluno que solicita os horários à Grade. A Grade, como colaborador, envia o horário organizado ao Aluno.

Figura 14 Diagrama de Interação entre Aluno e Grade.

Na Figura 15 temos o diagrama de domínio, que é descrito com diagramas de classes, suas associações e seus atributos. O PedidoHorario possui os atributos horário e o dia, possuindo uma relação de agregação com PedidoNegociacao que possui o atributo prioridade. PedidoNegociacao tem uma relação de composição com AceitacaoNegociacao, cujo atributo é o valor da prioridade. A OrganizaçãoHorario fornece o atributo grade ao PedidoHorario. Por fim, temos a Consulta Horário.

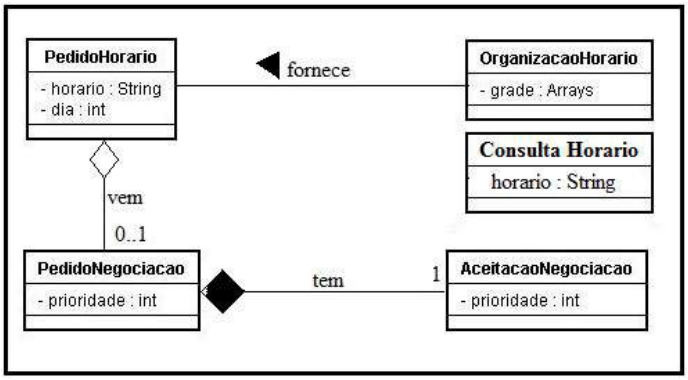


Figura 15 Diagrama do domínio

Nesta parte da modelagem ocorrerá a transição para o projeto de alto nível, no qual o modelo de análise será refinado. A determinação dos agentes se deu por número de papéis no sistema. Observando a Figura 16, vemos que para executar o papel da Grade o AgenteGrad terá que fornecer os serviços Reservar Horário, Fornecer Horário e acessar o Banco de Dados dos Horários. Já na Figura 17, o AgenteProfessor terá que fornecer o serviço de Registrar seu Horário para executar o papel do Professor. E o AgentAluno terá que fornecer o serviço Consultar Horário para executar o papel do Aluno.

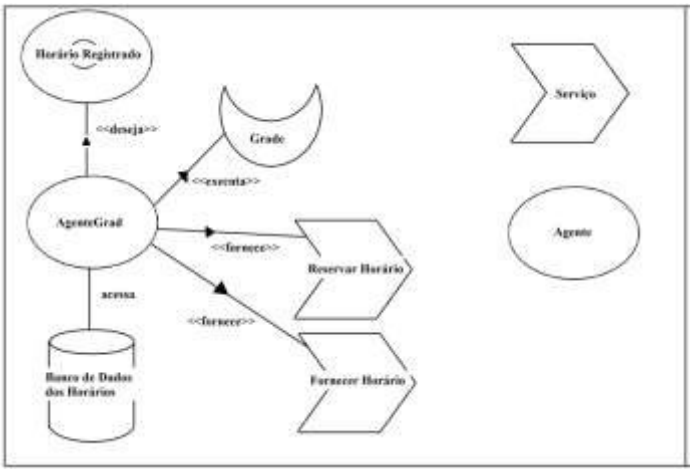


Figura 16 Diagrama de agentes (a).

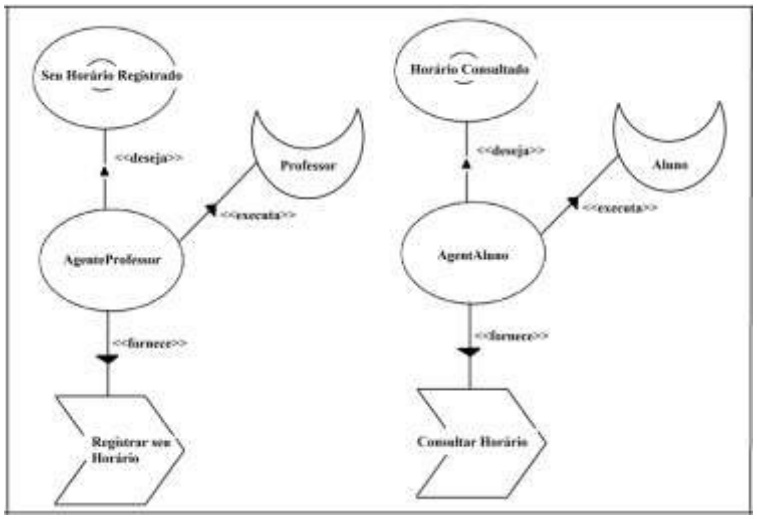


Figura 17 Diagrama de agentes (b).

O diagrama de sequência ilustrado na Figura 18 dá a ordem das tarefas executadas no sistema para que o AgenteGrad forneça o serviço Reservar Horário. A entrada para a tarefa Pede Horário do papel Professor é um PedidoHorário. A saída é o horário que será marcado pelo papel Grade, que depois passa para a tarefa Informa Horário.

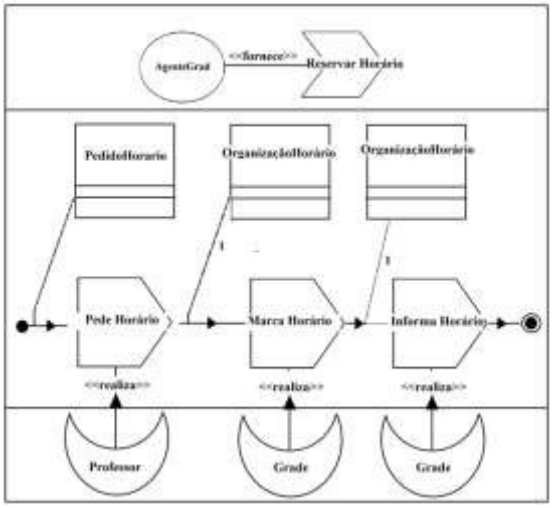


Figura 18 Digrama de sequência (a)

O diagrama de sequência ilustrado na Figura 19 dá a ordem das tarefas executadas no sistema para que o serviço Forneça Horário aconteça. A entrada para a tarefa Pede Horário do papel Aluno é um Consulta Horário. A saída é o Recupera Horário que será passado para Informa Horário.

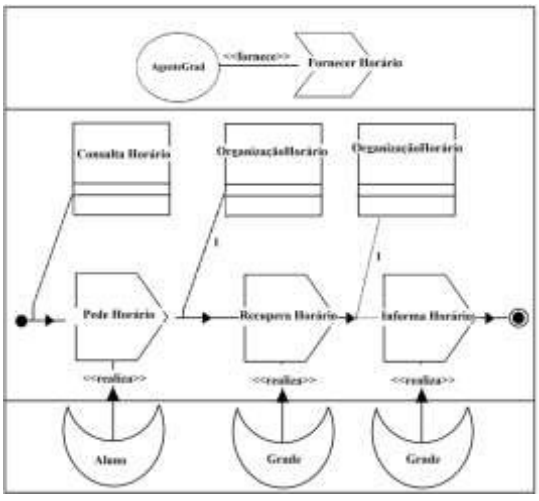


Figura 19 Diagrama de sequência (b)

O protocolo de interação usado na interação entre o papel Professor e o papel Grade é mostrado na Figura 20, onde temos que o Professor envia uma mensagem do tipo Request ao Grade que responde com uma mensagem do tipo Confirm, quando o pedido foi realizado, Not-understood quando a mensagem não foi entendida, ou seja, quando o conteúdo recebido é algo diferente do esperado, e Refuse quando o pedido não pôde ser atendido.

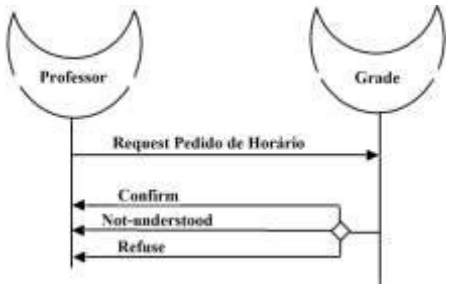


Figura 20 Protocolo de interação (a).

Quando a interação é com dois papéis do Professor, o Professor que deseja reservar um horário que já se encontra ocupado, envia uma mensagem do tipo Propose para o outro Professor. Caso o resultado seja que o ocupante ganhe a negociação, este enviará uma mensagem do tipo Refuse, caso contrário enviará uma mensagem Agree, e se a mensagem não for entendida, uma mensagem Not-understood será enviada. Confira a Figura 21.

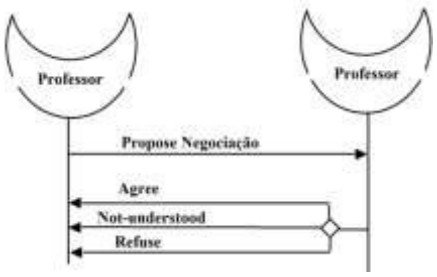


Figura 21 Protocolo de interação (b).

Na Figura 22 temos a interação do papel do Aluno que envia uma mensagem do tipo Request para o papel da Grade o qual responde com uma mensagem do tipo Inform ou Not-understood caso a mensagem de solicitação não seja entendida.

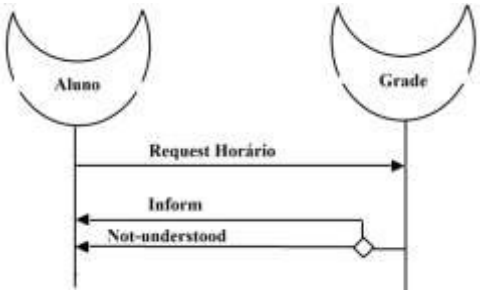


Figura 22 Protocolo de interação (c).

6 IMPLEMENTAÇÃO EM JADE

A implementação foi feita em Java no NetBeans versão 6.9.1, com bibliotecas do Jade na versão 3.5. Foram construídas três classes que correspondem aos agentes AgenteGrad, AgenteProfessor e o AgentAluno, cujos códigos podem ser vistos nos anexos A, B e C respectivamente. O AgenteGrad é o agente responsável por registrar os horários dos professores, representados por um agenteProfessor. O AgentAluno que representa um aluno, consulta o agenteGrad a fim de saber quais os horários do laboratório e também para saber os horários vagos para que ele possa usar para fins de estudo.

Assim que o AgenteGrad é executado, sua tela exibe a grade de horários vazia como é mostrado na Figura 23. Este agente fica esperando por requisições vindas de outros agentes. É seu objetivo principal reservar a um AgenteProfessor o horário desejado. Um outro objetivo é fornecer a grade atual a um AgenteAluno.

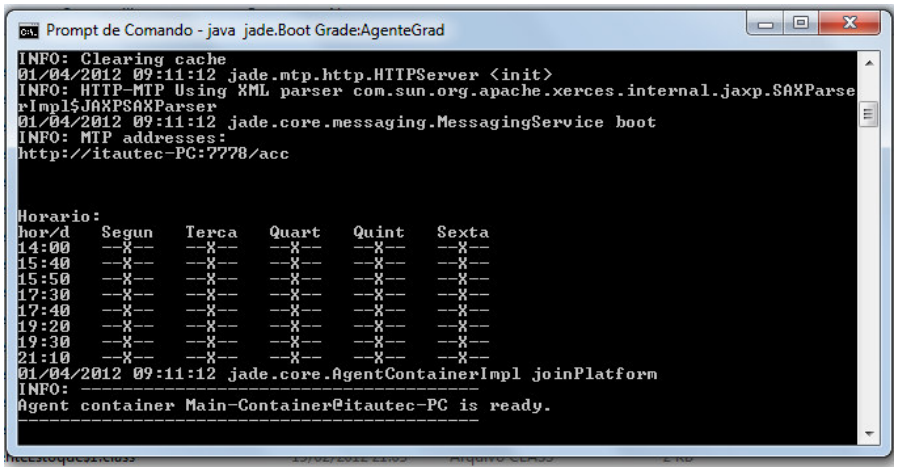


Figura 23 Tela inicial do agenteGrade

O AgenteProfessor é executado colocando como parâmetros o horário e o dia da semana, que irá enviar uma mensagem requisitando ao AgenteGrad o horário. Seu objetivo é ter seu horário reservado. Sua tela é mostrado na Figura 24.

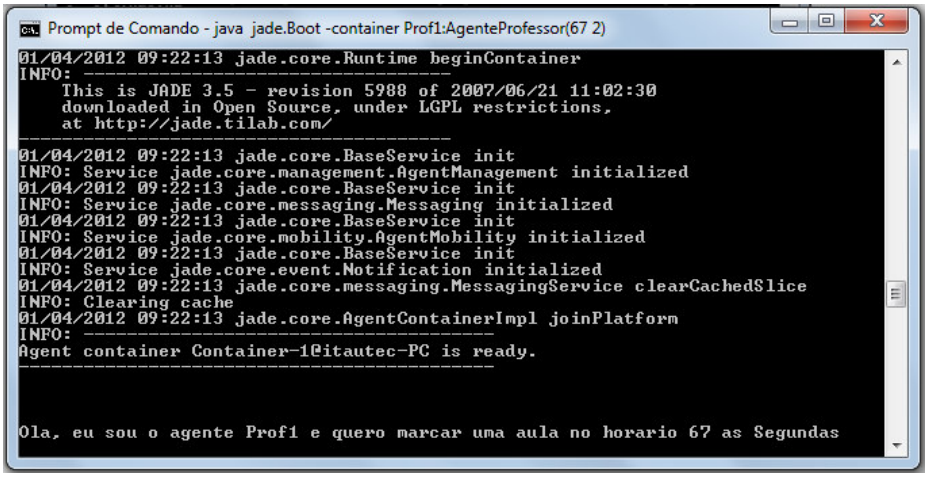


Figura 24 Tela do AgenteProfessor

Na Figura 25, observamos a interação entre o AgenteProfessor e o AgenteGrad através de mensagens que o agente Sniffer da plataforma Jade fornece. De cima para baixo temos que a primeira mensagem é a requisição do AgenteProfessor sobre o horário 6-7 às segundas, a outra mensagem enviada ao AgenteProfessor é de que sua requisição foi aceita.

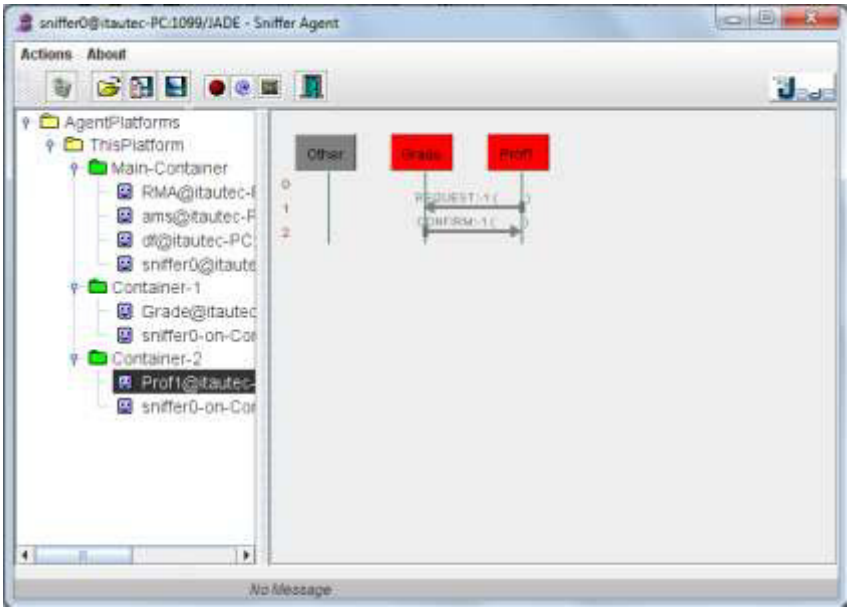


Figura 25 Interação entre agente Grade e agente Prof1

Como resultado, a Figura 26 mostra o AgenteGrad com o horário das 14:00 às 15:40, que corresponde ao horário 6-7 das segundas-feiras, reservado ao AgenteProfessor solicitante, cujo nome local é Prof1.

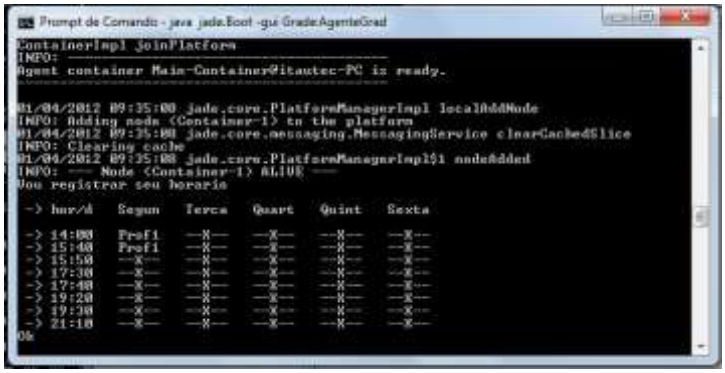


Figura 26: Horário reservado do Prof1.

A Figura 27 mostra a interface com a interação entre o agenteGrad e o AgentAluno.

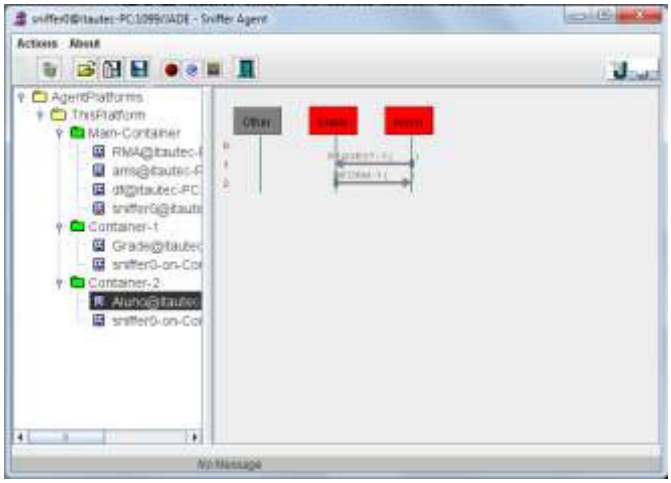


Figura 27: Interação entre AgenteGrad e AgentAluno.

A Figura 28 mostra a tela do agentAluno depois que sua solicitação foi atendida pelo AgenteGrad.

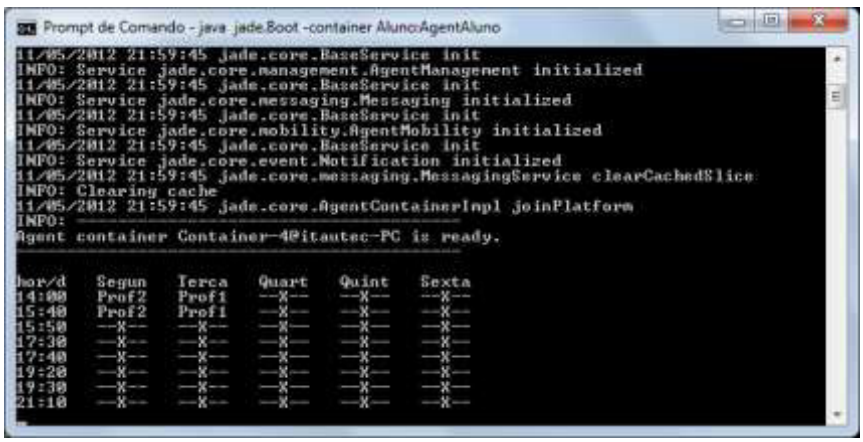


Figura 28: Tela do AgentAluno com a grade dos horários.

Quando todos os horários forem reservados, o AgenteGrad apresentará a tela mostrada na Figura 29 e cuja interação com os agentes é mostrado na figura 30.

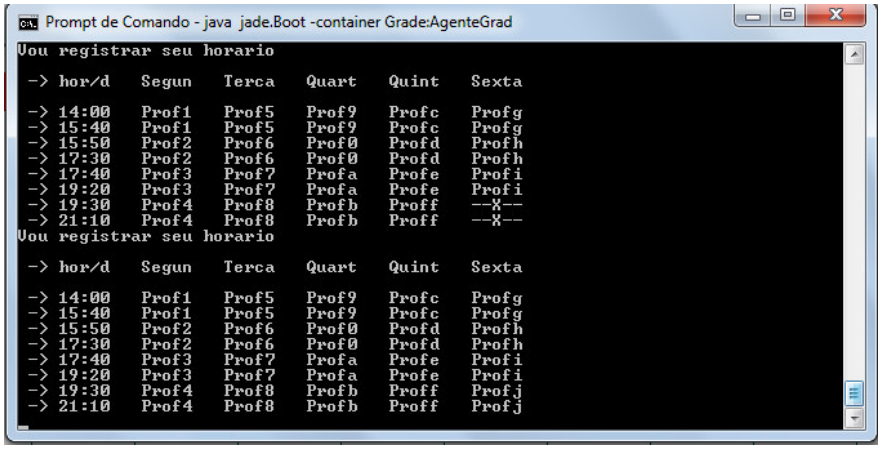


Figura 29 AgenteGrade com todos os horários preenchidos

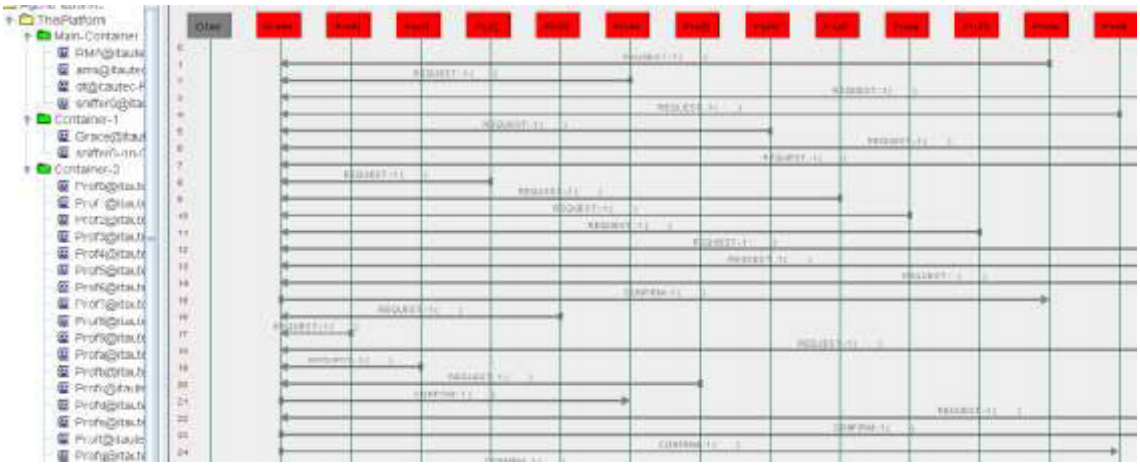


Figura 30 Interação dos agentes Professor com o agenteGrade

A Figura 31 mostra um caso típico de tentativa de uma ocupação de um horário que já se encontra ocupado por outro AgenteProfessor. Como resultado, o AgenteGrad envia uma mensagem recusando o pedido, e informa em seguida, qual AgenteProfessor reservou aquele horário, para que eles possam negociar, retirando inclusive o horário do Prof1, ou seja, o horário 6-7 da segunda feira ficou sem ocupante temporariamente. A negociação funciona da seguinte maneira: ambos os agentes (Prof1 e Prof2) possuem valores que indicam a sua prioridade para dar aula no laboratório, então o Prof2 envia uma proposta contendo sua prioridade ao Prof1, no caso da figura 29, fica evidente que a prioridade do Prof2 é inferior a do Prof1, sendo recusada, por isso teve outro horário marcado. O Prof1 então envia ao AgenteGrad a informação que ganhou a negociação, tendo uma confirmação do AgenteGrad. O Prof2 tenta outro horário, requisitando ao AgenteGrad, tendo a conformação do mesmo.

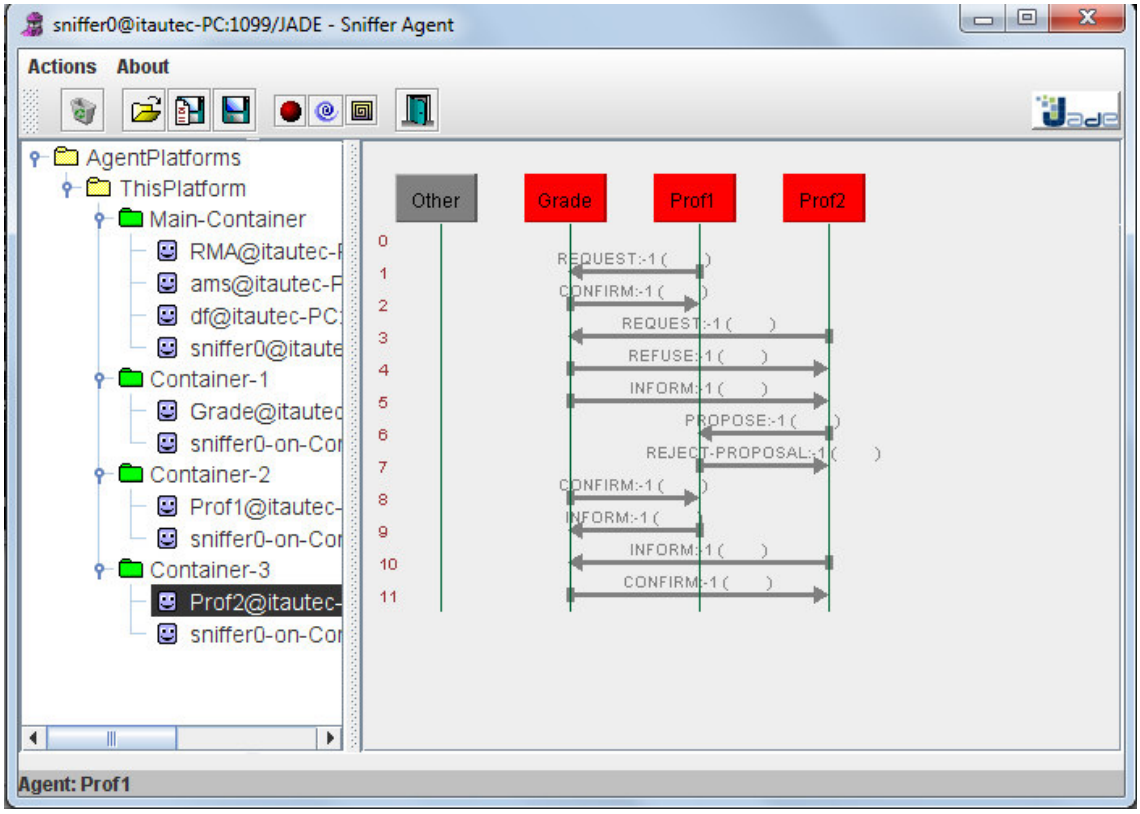


Figura 31 Tela da interação entre agentes por um mesmo horário

A Figura 32 mostra o resultado da negociação em que o agente Prof1 permaneceu com o seu horário 6-7 das segundas, em contra partida, o agente Prof2 alterou seu horário para 6-7 das terças.

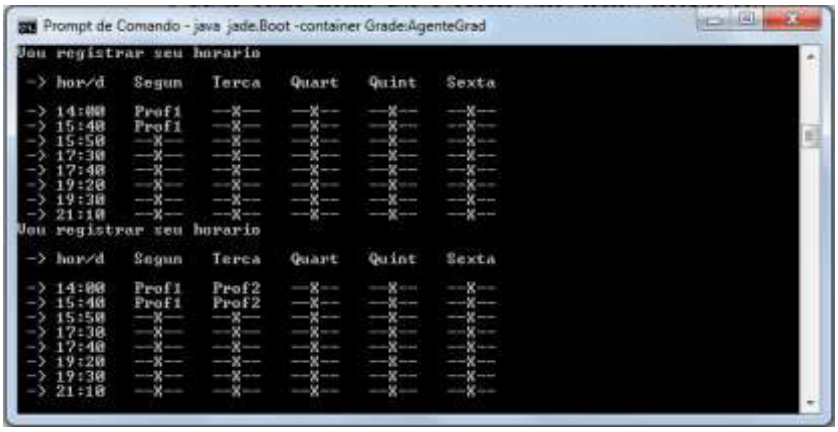


Figura 32 Resultado da interação entre os agentes Professor sobre um mesmo horário.

7 CONCLUSÃO

A engenharia de software baseada em agentes ainda não atinge um público expressivo no mercado. No entanto, essa engenharia já possui potencial de crescimento devido ao rápido avanço tecnológico e da complexidade dos sistemas.

A plataforma Jade é sem dúvida importante para o desenvolvimento de sistemas multiagente, pois permite desenvolver de forma simples aplicações que dependem de agentes, implementando a criação, comunicação e interação entre agentes.

Durante o levantamento bibliográfico foram identificadas diversas formas de se modelar um sistema multiagente, mostrando a dificuldade de se escolher a metodologia adequada a situação problema em questão.

Dessa forma, o sistema “Registra Horário” foi desenvolvido para mostrar a praticidade de um sistema multiagente, interagindo na plataforma Jade, uma das plataformas mais comuns, modelado no MESSAGE uma metodologia completa. O sistema desenvolvido apresentou um comportamento regular em relação às funções fornecidas pelo jade.

O sistema foi desenvolvido em Java sobre a plataforma NetBeans, e utilizou as APIs do Jade. O sistema é suportado apenas por sistemas operacionais Windows. O sistema Registra Horário foi satisfatório para estudar as relações de interação apresentadas, além de mostrar o potencial desenvolvido nessa área da evolução tecnológica.

Como trabalho futuro, poderia ser implementado um “SISTEMA DE PEDIDO DE TAXI”, onde o cliente poderia entrar com a sua localização e o local de destino para a central de taxi, esta passaria para o taxi livre mais próximo do solicitante, analisando se seria mais vantajoso ao taxista levá-lo ou não, pois poderia ter outro cliente, mais longe que o primeiro e que pagasse mais por um ponto de destino mais perto.

As dificuldades encontradas foram construir os diagramas como o diagrama de sequência, encontrar uma plataforma que facilitasse a criação dos agentes.

Uma grande facilidade foi entender a plataforma JADE, pois permitiu a criação dos agentes de forma fácil, permitindo a interação entra os mesmos. Considero boa a avaliação do sistema desenvolvido, pois conseguiu demonstrar a execução dos papéis e a interação dos mesmos para seus objetivos.

REFERÊNCIAS

BAYER, P.; SVANTESSON, M., (2001). **Comparison of Agent oriented methodologies analysis and design, MAS-CommonKADS versus Gaia**. In Fredriksson, M., editor, Bleckinge institute of technology student workshop on agent programming (BITSWAP 2001), Ronneby, Sweden.

BELLIFEMINE, F.; PGGI, A.; RIMASSA, G. (2001). **Developing Multi-agent with a FIPA-Compilant Agent Framework.** *Software Pratice and Experience,* 31, páginas 103-128.

BELLIFEMINE, F.; CAIRE, G.; POGGI, A.; RIMASSA, G. **JADE**: a White Paper. Telecom Italia EXP Magazine, Vol. 3, n. 3, 2003.

BERGENTI, F., GLEIZES, M., ZAMBONELLI, F, 2004. **METHODOLOGIES AND SOFTWARE ENGINEERING FOR AGENT SYSTEMS** The Agent-Oriented Software Engineering HandBook.

BIGUS, J. P.; BIGUS, J. **Constructing Intelligent Agents Using Java**. Wiley Computer Publishing, (2001).

BUZETTA, P.; RONNQUIST, R.; HODGSON, A.; LUCAS A. **Jack Intelligent Agents-Components for Intelligent Agents in Java**. AOS Technical Report TR9901, 1999.

DAM, K. H.; M., WINIKOFF, M. **Comparing AgentOriented Methodologies**. Fifth International Biconference Workshop on Agent-Oriented Information Systems (AOIS-2003). Malbourne, Australia, 2003.

KERR, D., O' SULLIVAN, D., EVANS, R., RICHARDSON, R.; SOMER, F., (1998). **Experiences using Intellignet Agent Technologies as Unifying Approach to Network and Service Management.** *Proceedings of IS & N'98,* Antwerp, Belgium.

MILOJICIT, D., BREUGST, M., BUSSE, I., CAMPBELL, J., COVACI, S., FRIEDMAN, B., KOSAKA, K., LANGE, D., ONO, K., OSHIMA, m., THAM, C., VIRDHAGRISWARAN, S.; WHITE J. (1998). **MASIF – The OMG Mobile Agent System Interoperability Facility.** Rothermel K., and Hohl, F. (Eds.) *Proceedings of ths 2nd International Workshop Mobiles Agents,* páginas 50-67, Springer-Verlag.

NWANA, H. S.; NDUMU, D. T.; LEE. L. C.; COLLIS, J. C.:**ZEUS: A toolkits for Building Distribuited Multi-Agent**. In Proceedings of the Third International Conference on Autonomous Agents, ACM Press, 1999.

SHEHORY, O.; STURM, A. **Evaluation of Modeling Techniques for Agent-Based Systems**. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceeding of the Fifth Internationl Conference on Autonomous Agents, páginas 624-631. ACM Press, Maio 2001.*

PRESSMAN, R. **Software Engineering:** a Practitioner's Approach, 6ªedição, Mc Graw Hill, 2005.

SOMMERVILLE, **ENGENHARIA DE SOFTWARE,** 2007

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**, 2002.

WOOLDRIDGE, M. J.; JENNING, N. R. (1995a). **Intelligent Agents: Theory and Practice.** *Knowledge Engineering Review*, 10(2), paginas 115-152.

WOOLDRIDGE, M. J.; JENNING, N. R. (1995b). **Agent Theories, Architectures, and Languages: A Survey.** Wooldridge, M. J.; Jennings, N. R. (Eds) *Intelligent Agents*, páginas 1-39, Springer-Verlag.

ZAMBONELLI, F.; JENNING, N. R.; WOOLDRIDGE, M. **Development Multiagent Systems:** The Gaia Methodology. ACM Transactions on Software Engineering and Methodology, Vol. 12, Nº. 3, Julho, Páginas 317- 370, 2003.

Anexos

Anexo A

Código da classe AgenteGrad

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.lang.acl.ACLMessage;
import java.util.StringTokenizer;
import java.util.GregorianCalendar;
import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.Date;
public class AgenteGrad extends Agent{
   String horario = " ";
   public void setup()
   {
      System.out.println("\n\n");
      final String grade[][] = new String [9][6];
      grade[0][0] = "hor/d";grade[0][1] = "Segun";grade[0][2] = "Terca";grade[0][3] =
"Quart";grade[0][4] = "Quint";grade[0][5] = "Sexta";
      grade[1][0] = "14:00";grade[1][1] = "--X--";grade[1][2] = "--X--";grade[1][3] = "--X--
";grade[1][4] = "--X--";grade[1][5] = "--X--";
      grade[2][0] = "15:40";grade[2][1] = "--X--";grade[2][2] = "--X--";grade[2][3] = "--X--
";grade[2][4] = "--X--";grade[2][5] = "--X--";
      grade[3][0] = "15:50";grade[3][1] = "--X--";grade[3][2] = "--X--";grade[3][3] = "--X--
";grade[3][4] = "--X--";grade[3][5] = "--X--";
      grade[4][0] = "17:30";grade[4][1] = "--X--";grade[4][2] = "--X--";grade[4][3] = "--X--
";grade[4][4] = "--X--";grade[4][5] = "--X--";
      grade[5][0] = "17:40";grade[5][1] = "--X--";grade[5][2] = "--X--";grade[5][3] = "--X--
";grade[5][4] = "--X--";grade[5][5] = "--X--";
      grade[6][0] = "19:20";grade[6][1] = "--X--";grade[6][2] = "--X--";grade[6][3] = "--X--
";grade[6][4] = "--X--";grade[6][5] = "--X--";
      grade[7][0] = "19:30";grade[7][1] = "--X--";grade[7][2] = "--X--";grade[7][3] = "--X--
";grade[7][4] = "--X--";grade[7][5] = "--X--";
      grade[8][0] = "21:10";grade[8][1] = "--X--";grade[8][2] = "--X--";grade[8][3] = "--X--
";grade[8][4] = "--X--";grade[8][5] = "--X--";

      System.out.println("Horario: "
                        + "\n" + grade[0][0]+"   "+grade[0][1]+"   "+grade[0][2]+"
"+grade[0][3]+"   "+grade[0][4]+"   "+grade[0][5]
                        + "\n" + grade[1][0]+"   "+grade[1][1]+"   "+grade[1][2]+"
"+grade[1][3]+"   "+grade[1][4]+"   "+grade[1][5]
                        + "\n" + grade[2][0]+"   "+grade[2][1]+"   "+grade[2][2]+"
"+grade[2][3]+"   "+grade[2][4]+"   "+grade[2][5]
                        + "\n" + grade[3][0]+"   "+grade[3][1]+"   "+grade[3][2]+"
"+grade[3][3]+"   "+grade[3][4]+"   "+grade[3][5]
```

```
                    + "\n" + grade[4][0]+"   "+grade[4][1]+"   "+grade[4][2]+"
"+grade[4][3]+"   "+grade[4][4]+"   "+grade[4][5]
                    + "\n" + grade[5][0]+"   "+grade[5][1]+"   "+grade[5][2]+"
"+grade[5][3]+"   "+grade[5][4]+"   "+grade[5][5]
                    + "\n" + grade[6][0]+"   "+grade[6][1]+"   "+grade[6][2]+"
"+grade[6][3]+"   "+grade[6][4]+"   "+grade[6][5]
                    + "\n" + grade[7][0]+"   "+grade[7][1]+"   "+grade[7][2]+"
"+grade[7][3]+"   "+grade[7][4]+"   "+grade[7][5]
                    + "\n" + grade[8][0]+"   "+grade[8][1]+"   "+grade[8][2]+"
"+grade[8][3]+"   "+grade[8][4]+"   "+grade[8][5]);
      horario = "Horario: "
                    + "\n" + grade[0][0]+"   "+grade[0][1]+"   "+grade[0][2]+"
"+grade[0][3]+"   "+grade[0][4]+"   "+grade[0][5]
                    + "\n" + grade[1][0]+"   "+grade[1][1]+"   "+grade[1][2]+"
"+grade[1][3]+"   "+grade[1][4]+"   "+grade[1][5]
                    + "\n" + grade[2][0]+"   "+grade[2][1]+"   "+grade[2][2]+"
"+grade[2][3]+"   "+grade[2][4]+"   "+grade[2][5]
                    + "\n" + grade[3][0]+"   "+grade[3][1]+"   "+grade[3][2]+"
"+grade[3][3]+"   "+grade[3][4]+"   "+grade[3][5]
                    + "\n" + grade[4][0]+"   "+grade[4][1]+"   "+grade[4][2]+"
"+grade[4][3]+"   "+grade[4][4]+"   "+grade[4][5]
                    + "\n" + grade[5][0]+"   "+grade[5][1]+"   "+grade[5][2]+"
"+grade[5][3]+"   "+grade[5][4]+"   "+grade[5][5]
                    + "\n" + grade[6][0]+"   "+grade[6][1]+"   "+grade[6][2]+"
"+grade[6][3]+"   "+grade[6][4]+"   "+grade[6][5]
                    + "\n" + grade[7][0]+"   "+grade[7][1]+"   "+grade[7][2]+"
"+grade[7][3]+"   "+grade[7][4]+"   "+grade[7][5]
                    + "\n" + grade[8][0]+"   "+grade[8][1]+"   "+grade[8][2]+"
"+grade[8][3]+"   "+grade[8][4]+"   "+grade[8][5];

      addBehaviour(new CyclicBehaviour(this)
      {
         public void action()
         {
            ACLMessage msg = myAgent.receive();
            if(msg != null)
            {
                                    if(msg.getContent().equalsIgnoreCase("Horario"))
                  {
                     ACLMessage msg5 = new ACLMessage(ACLMessage.INFORM);
                     msg5.addReceiver(new AID("Aluno", AID.ISLOCALNAME));
                     //msg5.addReceiver(new AID(msg5.getSender().));
                     msg5.setContent(horario);
                     myAgent.send(msg5);
                     System.out.println("Vou enviar o horario!");
                  }else
                  {
                     String nome = msg.getSender().getLocalName();
                     StringTokenizer st = new StringTokenizer(msg.getContent());
                     String aux = st.nextToken();//horário por ex.: 67
```

```
                    String aux1 = st.nextToken();//Dia da Semanal
                    String aux2 =aux;
                    String aux3 = aux1;
                    boolean reservado = false;
                    int j = 1;
                    int num = Integer.parseInt(aux2);
                    int num2 = Integer.parseInt(aux3);
                    switch(num2)
                    {
                       case 2:
                          j = 1;
                          break;
                       case 3:
                          j = 2;
                          break;
                       case 4:
                          j = 3;
                          break;
                       case 5:
                          j = 4;
                          break;
                       case 6:
                          j = 5;
                          break;
                    }
                    switch(num)
                    {
                       case 67:
                          if(grade[1][j] == "--X--")
                          {
                             grade[1][j] = nome;
                             grade[2][j] = nome;
                          }else
                          {
                             String nome1 = grade[1][j];
                             ACLMessage msg1 = new
ACLMessage(ACLMessage.REFUSE);
                             msg1.addReceiver(new
AID(msg.getSender().getLocalName(),AID.ISLOCALNAME));
                             msg1.setLanguage("Portugues");
                             msg1.setOntology("Educacao");
                             msg1.setContent("O agent " + nome1 + " ocupa este
horario " + num + " " + num2);
                             grade[1][j] = "--X--";
                             grade[2][j] = "--X--";
                             myAgent.send(msg1);
                             reservado = true;

                          }
                          break;
```

```
                        case 89:
                            if(grade[3][j] == "--X--")
                            {
                                grade[3][j] = nome;
                                grade[4][j] = nome;
                            }else
                            {
                                String nome1 = grade[3][j];
                                ACLMessage msg1 = new
ACLMessage(ACLMessage.REFUSE);
                                msg1.addReceiver(new
AID(msg.getSender().getLocalName(),AID.ISLOCALNAME));
                                msg1.setLanguage("Portugues");
                                msg1.setOntology("Educacao");
                                msg1.setContent("O agent " + nome1 + " ocupa este
horario " + num + " " + num2);
                                grade[3][j] = "--X--";
                                grade[4][j] = "--X--";
                                myAgent.send(msg1);
                                reservado = true;
                            }
                            break;
                        case 1011:
                            if(grade[5][j] == "--X--")
                            {
                                grade[5][j] = nome;
                                grade[6][j] = nome;
                            }else
                            {
                                String nome1 = grade[5][j];
                                ACLMessage msg1 = new
ACLMessage(ACLMessage.REFUSE);
                                msg1.addReceiver(new
AID(msg.getSender().getLocalName(),AID.ISLOCALNAME));
                                msg1.setLanguage("Portugues");
                                msg1.setOntology("Educacao");
                                msg1.setContent("O agent " + nome1 + " ocupa este
horario " + num + " " + num2);
                                grade[5][j] = "--X--";
                                grade[6][j] = "--X--";
                                myAgent.send(msg1);
                                reservado = true;
                            }
                            break;
                        case 1213:
                            if(grade[7][j] == "--X--")
                            {
                                grade[7][j] = nome;
                                grade[8][j] = nome;
                            }else
```

```
                        {
                            String nome1 = grade[7][j];
                            ACLMessage msg1 = new
ACLMessage(ACLMessage.REFUSE);
                            msg1.addReceiver(new
AID(msg.getSender().getLocalName(),AID.ISLOCALNAME));
                            msg1.setLanguage("Portugues");
                            msg1.setOntology("Educacao");
                            msg1.setContent("O agent " + nome1 + " ocupa este
horario " + num + " " + num2);
                            grade[7][j] = "--X--";
                            grade[8][j] = "--X--";
                            myAgent.send(msg1);
                            reservado = true;
                        }
                        break;
                    }
                    if(!reservado)
                    {
                        System.out.println("Vou registrar seu horario");
                        System.out.println();
                        for(int i = 0;i<9;i++){
                        System.out.println(" -> " + grade[i][0] + "   " + grade[i][1] + "   " +
grade[i][2] + "   " + grade[i][3] + "   " + grade[i][4] + "   " + grade[i][5]);
                        if(i == 0)
                            System.out.println();
                        }
                    ACLMessage msg2 = new ACLMessage(ACLMessage.CONFIRM);
                    msg2.addReceiver(new AID(nome, AID.ISLOCALNAME));
                    msg2.setLanguage("Portugues");
                    msg2.setOntology("Educacao");
                    msg2.setContent("Horario registrado"
                        + "\n" + grade[0][0]+"   "+grade[0][1]+"   "+grade[0][2]+"
"+grade[0][3]+"   "+grade[0][4]+"   "+grade[0][5]
                        + "\n" + grade[1][0]+"   "+grade[1][1]+"   "+grade[1][2]+"
"+grade[1][3]+"   "+grade[1][4]+"   "+grade[1][5]
                        + "\n" + grade[2][0]+"   "+grade[2][1]+"   "+grade[2][2]+"
"+grade[2][3]+"   "+grade[2][4]+"   "+grade[2][5]
                        + "\n" + grade[3][0]+"   "+grade[3][1]+"   "+grade[3][2]+"
"+grade[3][3]+"   "+grade[3][4]+"   "+grade[3][5]
                        + "\n" + grade[4][0]+"   "+grade[4][1]+"   "+grade[4][2]+"
"+grade[4][3]+"   "+grade[4][4]+"   "+grade[4][5]
                        + "\n" + grade[5][0]+"   "+grade[5][1]+"   "+grade[5][2]+"
"+grade[5][3]+"   "+grade[5][4]+"   "+grade[5][5]
                        + "\n" + grade[6][0]+"   "+grade[6][1]+"   "+grade[6][2]+"
"+grade[6][3]+"   "+grade[6][4]+"   "+grade[6][5]
                        + "\n" + grade[7][0]+"   "+grade[7][1]+"   "+grade[7][2]+"
"+grade[7][3]+"   "+grade[7][4]+"   "+grade[7][5]
                        + "\n" + grade[8][0]+"   "+grade[8][1]+"   "+grade[8][2]+"
"+grade[8][3]+"   "+grade[8][4]+"   "+grade[8][5]);
```

```
                    horario = "\n" + grade[0][0]+"   "+grade[0][1]+"   "+grade[0][2]+"
"+grade[0][3]+"   "+grade[0][4]+"   "+grade[0][5]
                        + "\n" + grade[1][0]+"   "+grade[1][1]+"   "+grade[1][2]+"
"+grade[1][3]+"   "+grade[1][4]+"   "+grade[1][5]
                        + "\n" + grade[2][0]+"   "+grade[2][1]+"   "+grade[2][2]+"
"+grade[2][3]+"   "+grade[2][4]+"   "+grade[2][5]
                        + "\n" + grade[3][0]+"   "+grade[3][1]+"   "+grade[3][2]+"
"+grade[3][3]+"   "+grade[3][4]+"   "+grade[3][5]
                        + "\n" + grade[4][0]+"   "+grade[4][1]+"   "+grade[4][2]+"
"+grade[4][3]+"   "+grade[4][4]+"   "+grade[4][5]
                        + "\n" + grade[5][0]+"   "+grade[5][1]+"   "+grade[5][2]+"
"+grade[5][3]+"   "+grade[5][4]+"   "+grade[5][5]
                        + "\n" + grade[6][0]+"   "+grade[6][1]+"   "+grade[6][2]+"
"+grade[6][3]+"   "+grade[6][4]+"   "+grade[6][5]
                        + "\n" + grade[7][0]+"   "+grade[7][1]+"   "+grade[7][2]+"
"+grade[7][3]+"   "+grade[7][4]+"   "+grade[7][5]
                        + "\n" + grade[8][0]+"   "+grade[8][1]+"   "+grade[8][2]+"
"+grade[8][3]+"   "+grade[8][4]+"   "+grade[8][5];
                       myAgent.send(msg2);
                       }else
                       {
                          System.out.println("Vou esperar pela negociacao!");
                          ACLMessage msg2 = new
ACLMessage(ACLMessage.INFORM);
                          msg2.addReceiver(new AID(nome, AID.ISLOCALNAME));
                          msg2.setLanguage("Portugues");
                          msg2.setOntology("Educacao");
                          msg2.setContent("Vou esperar pela negociacao!"
                          + "\n" + grade[0][0]+"   "+grade[0][1]+"   "+grade[0][2]+"
"+grade[0][3]+"   "+grade[0][4]+"   "+grade[0][5]
                          + "\n" + grade[1][0]+"   "+grade[1][1]+"   "+grade[1][2]+"
"+grade[1][3]+"   "+grade[1][4]+"   "+grade[1][5]
                          + "\n" + grade[2][0]+"   "+grade[2][1]+"   "+grade[2][2]+"
"+grade[2][3]+"   "+grade[2][4]+"   "+grade[2][5]
                          + "\n" + grade[3][0]+"   "+grade[3][1]+"   "+grade[3][2]+"
"+grade[3][3]+"   "+grade[3][4]+"   "+grade[3][5]
                          + "\n" + grade[4][0]+"   "+grade[4][1]+"   "+grade[4][2]+"
"+grade[4][3]+"   "+grade[4][4]+"   "+grade[4][5]
                          + "\n" + grade[5][0]+"   "+grade[5][1]+"   "+grade[5][2]+"
"+grade[5][3]+"   "+grade[5][4]+"   "+grade[5][5]
                          + "\n" + grade[6][0]+"   "+grade[6][1]+"   "+grade[6][2]+"
"+grade[6][3]+"   "+grade[6][4]+"   "+grade[6][5]
                          + "\n" + grade[7][0]+"   "+grade[7][1]+"   "+grade[7][2]+"
"+grade[7][3]+"   "+grade[7][4]+"   "+grade[7][5]                          + "\n" +
grade[8][0]+"   "+grade[8][1]+"   "+grade[8][2]+"   "+grade[8][3]+"   "+grade[8][4]+"
"+grade[8][5]);myAgent.send(msg2);}}}elseblock(); }});}}
```

Anexo B

Código da classe AgenteProfessor

```
import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.*;
import jade.domain.FIPAException;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import java.util.StringTokenizer;
public class AgenteProfessor extends Agent{
   public void setup()
   {
      System.out.println("\n\n");
      final Object[] args = getArguments();
      addBehaviour(new OneShotBehaviour(this)
      {
         public void action()
         {
            ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
            msg.addReceiver(new AID("Grade", AID.ISLOCALNAME));
            msg.setLanguage("Portugues");
            msg.setOntology("Educacao");
            if(args != null && args.length > 0)
            {
               String hora = (String)args[0];
               String dia = (String)args[1];
               int di = Integer.parseInt(dia);
               String d = " ";
               switch(di)
               {
                  case 2:
                     d = "Segundas";
                     break;
                  case 3:
                     d = "Tercas";
                     break;
                  case 4:
                     d = "Quartas";
                     break;
                  case 5:
                     d = "Quintas";
                     break;
                  case 6:
                     d = "Sextas";
                     break;
```

```
                }
                msg.setContent(hora + " " + dia);
                System.out.println("Ola, eu sou o agente " + getLocalName() + " e quero
marcar uma aula no horario " + hora + " as " + d + " ");
             }
             myAgent.send(msg);
          }
       });
       addBehaviour(new CyclicBehaviour(this)
       {
          public void action()
          {
            ACLMessage msg = myAgent.receive();

             if(msg!=null)
             {
                StringTokenizer st = new StringTokenizer(msg.getContent());
                String aux1 = st.nextToken();
                String aux2 = st.nextToken();
                if(aux1.equalsIgnoreCase("O") && aux2.equalsIgnoreCase("agent"))
                {
                   String aux4 = st.nextToken();
                   String aux5 = st.nextToken();
                   String aux6 = st.nextToken();
                   String aux7 = st.nextToken();
                   String aux0 = st.nextToken();
                   String aux8 = st.nextToken();
                   ACLMessage msg3 = new ACLMessage(ACLMessage.PROPOSE);
                   msg3.addReceiver(new AID(aux4, AID.ISLOCALNAME));
                   msg3.setLanguage("Portugues");
                   msg3.setOntology("Educacao");
                   double pri = (Math.random()*10);
                   int p = (int)pri;
                   msg3.setContent("Vamos negociar" + " " + p + " " + aux0 + " " +
aux8);//aqui
                   myAgent.send(msg3);
                }
                if(aux1.equalsIgnoreCase("Vamos") &&
aux2.equalsIgnoreCase("negociar"))
                {
                   String aux4 = st.nextToken();
                   String au = st.nextToken();
                   int au5 = Integer.parseInt(au);
                   String au2 = st.nextToken();
                   int p1 = Integer.parseInt(aux4);
                   double pri2 = (Math.random()*10);
                   int p2 = (int)pri2;
                   System.out.println("pri2 " + p2);
                   if(p1 >= p2)
                   {
```

```
                    ACLMessage msg5 = new ACLMessage(ACLMessage.AGREE);
                    msg5.addReceiver(new AID(msg.getSender().getLocalName(),
AID.ISLOCALNAME));
                    msg5.setLanguage("Portugues");
                    msg5.setOntology("Educacao");
                    msg5.setContent("Voce venceu" + " " + au + " " + au2);
                    int numero1 = Integer.parseInt(au2);
                    numero1 = numero1 + 1;
                    myAgent.send(msg5);
                    ACLMessage msg6 = new ACLMessage(ACLMessage.INFORM);
                    msg6.addReceiver(new AID("Grade", AID.ISLOCALNAME));
                    msg6.setLanguage("Portugues");
                    msg6.setOntology("Educacao");
                    if(numero1 == 7)
                    {
                       if(au5 == 1213)
                       {
                          System.out.println("Já era");
                          System.exit(1);
                       }
                       if(au5 == 67){
                          au5 = 89;
                       }else{
                       if(au5 == 89){
                          au5 = 1011;
                       }else{
                       if(au5 == 1011){
                          au5 = 1213;
                       }
                    }}}
                    msg6.setContent(au5 + " " + numero1);
                    myAgent.send(msg6);
                 }else
                 {
                    ACLMessage msg5 = new
ACLMessage(ACLMessage.REJECT_PROPOSAL);
                    msg5.addReceiver(new AID(msg.getSender().getLocalName(),
AID.ISLOCALNAME));
                    msg5.setLanguage("Portugues");
                    msg5.setOntology("Educacao");
                    msg5.setContent("Eu venci" + " " + au + " " + au2);
                    myAgent.send(msg5);
                    ACLMessage msg9 = new ACLMessage(ACLMessage.INFORM);
                    msg9.addReceiver(new AID("Grade", AID.ISLOCALNAME));
                    msg9.setLanguage("Portugues");
                    msg9.setOntology("Educacao");
                    msg9.setContent(au + " " + au2);
                    myAgent.send(msg9);
                 }
              }
```

```
                if(aux1.equalsIgnoreCase("Eu") && aux2.equalsIgnoreCase("venci"))
                {
                   String auxiliar = st.nextToken();
                   String auxiliar2 = st.nextToken();
                   int numero = Integer.parseInt(auxiliar2);
                   numero = numero + 1;
                   int numero2 = Integer.parseInt(auxiliar);
                   if(numero == 7)
                   {
                      if(numero2 == 1213){
                         System.out.println("ja era2");
                         System.exit(1);
                      }
                      if(numero2 == 67)
                         numero2 = 89;
                      else{
                      if(numero2 == 89)
                         numero2 = 1011;
                      else{
                      if(numero2 == 1011)
                         numero2 = 1213;
                   }}}
                   ACLMessage msg7 = new ACLMessage(ACLMessage.INFORM);
                   msg7.addReceiver(new AID("Grade", AID.ISLOCALNAME));
                   msg7.setLanguage("Portugues");
                   msg7.setOntology("Educacao");
                   msg7.setContent(numero2 + " " + numero);
                   myAgent.send(msg7);
                }
                if(aux1.equalsIgnoreCase("Voce") &&
aux2.equalsIgnoreCase("venceu"))
                {
                   String a1 = st.nextToken();
                   String a2 = st.nextToken();
                   ACLMessage msg7 = new ACLMessage(ACLMessage.INFORM);
                   msg7.addReceiver(new AID("Grade", AID.ISLOCALNAME));
                   msg7.setLanguage("Portugues");
                   msg7.setOntology("Educacao");
                   msg7.setContent(a1 + " " + a2);
                   myAgent.send(msg7);
                }
             }else block(); } }); } }
```

Anexo C

Código da classe AgentAluno

```
import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
import java.util.StringTokenizer;
public class AgentAluno extends Agent{
   int cont = 0;
   public void setup()
   {

      addBehaviour(new OneShotBehaviour(this)
      {
         public void action()
         {
            ACLMessage msg1 = new ACLMessage(ACLMessage.REQUEST);
            msg1.addReceiver(new AID("Grade", AID.ISLOCALNAME));
            msg1.setContent("Horario");
            myAgent.send(msg1);
         }
      });
      addBehaviour(new CyclicBehaviour(this){
            public void action()
   {
         ACLMessage msg = myAgent.receive();
         if(msg != null)
         {
            String conteudo = msg.getContent();
            System.out.println(conteudo);

         }else
            block();
   }
      });

   }
}
```