AGENTES DE SOFTWARE NO APOIO AO GERENCIAMENTO DE REPOSITÓRIOS DE COMPONENTES NÃO-CENTRALIZADOS

Vinícius de Souza Oliveira

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientadora: Kele Teixeira Belloze

Juiz de Fora, MG
Julho de 2010

AGENTES DE SOFTWARE NO APOIO AO GERENCIAMENTO DE REPOSITÓRIOS DE COMPONENTES NÃO-CENTRALIZADOS

Vinícius de Souza Oliveira

MONOGRAFIA SUBMETIDADA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada pela banca constituída pelos seguintes membros:

_____

Profa. Kele Teixeira Belloze – orientadora
M. Sc. em Sistemas e Computação, IME-RJ, 2007

_____

Prof. Ilaim Costa Junior
D. Sc. em Eng. de Produção, UFSC, 2002

_____

Prof. Jairo Francisco de Souza
M. Sc. em Eng. de Sist. e Comp., COPPE/UFRJ, 2007

Juiz de Fora, MG
Julho de 2010

Agradecimentos

À minha família, pelo apoio e incentivo em todos os momentos da minha vida.

Aos amigos sempre presentes e que me acompanharam nessa caminhada.

À professora Kele, pelos ensinamentos e conselhos.

À todos os demais professores, pelo conhecimento e experiência compartilhada.

Sumário

Lista de Figuras

Resumo

A utilização de repositórios de componentes viabiliza o reuso de *software*, gerando ganhos de produtividade e redução de custos. A aplicação de agentes desenvolvida neste trabalho tem como objetivo apoiar o gerenciamento de repositórios de componentes não-centralizados. O trabalho apresenta os conceitos relacionados à agentes, repositórios de componentes e também o estudo realizado sobre linguagens de modelagem de agentes e *frameworks* de desenvolvimento de sistemas multi-agentes. O sistema foi desenvolvido com o apoio da ferramenta JADE e da linguagem de modelagem ANote.

Palavras-Chave: Agentes de Software, Sistemas Multi-Agentes, Repositório de Componentes, Reuso.

1. Introdução

1.1 Motivação

Com a crescente importância atribuída ao reuso de software e informações, muito se tem discutido a respeito da organização e centralização desses valores. O repositório surge como uma solução para se resolver um paradoxo observado nas organizações: sabe-se que o compartilhamento de experiências e informações pode proporcionar grandes vantagens competitivas, entretanto muitas empresas ficam impossibilitadas de aproveitar tais benefícios porque não conseguem lidar com a variedade e a distribuição ineficiente das informações desejadas e, muitas vezes, sequer têm conhecimento das informações disponíveis (HENNINGER, 1997). A utilização de agentes de *software* neste contexto poderia contribuir com o apoio ao gerenciamento de repositórios.

1.2 Objetivos

O objetivo deste trabalho é mostrar a viabilidade do uso de agentes para auxiliar o gerenciamento de repositórios de componentes não-centralizados e assim estimular o reuso de software. Para isso foi desenvolvida uma aplicação prática, tendo como base os conceitos de agentes de software. A aplicação proposta é um sistema multi-agentes para apoiar o gerenciamento de repositórios de componentes não-centralizados. A comunicação entre os agentes permitirá a notificação de inserção de novos componentes em diferentes repositórios e também irá possibilitar que o usuário solicite os componentes que lhe foram notificados. De forma geral, o escopo da aplicação pode ser ampliado para apoiar o gerenciamento de qualquer artefato produzido durante um projeto de software.

1.3 Metodologia

Para o desenvolvimento do trabalho proposto, foram estudados os conceitos e definições de agentes, assim como os tipos de agentes, tipos de colaboração e comunicação. Também foi realizada a análise de linguagens de modelagem e frameworks para o desenvolvimento de sistemas multi-agentes, permitindo assim a comparação entre as ferramentas e a escolha da mais adequada ao desenvolvimento do sistema. A linguagem de modelagem escolhida foi o ANote, enquanto o JADE foi a ferramenta escolhida para o desenvolvimento do sistema.

Uma pesquisa sobre repositórios, e mais especificamente repositórios de componentes, foi realizada e foi levantada uma discussão sobre como os repositórios trabalhariam de forma não centralizada e desta maneira, entender como os agentes ajudariam nesta situação e quais seriam as vantagens da aplicação de agentes sobre este domínio e também a possibilidade de aplicação em outros domínios.

Foram levantados componentes *open source* para popular os repositórios que serviram para a realização de testes da aplicação. Foram realizadas a modelagem e a implementação dos agentes e posteriormente simulações da execução do sistema para que a comunicação entre os agentes fosse analisada.

1.4 Organização do Trabalho

A monografia está dividida em 6 capítulos, inclusive este primeiro que apresenta o tema proposto e a organização do trabalho. No capítulo 2 são apresentadas as definições de agentes e sistemas multi-agentes. O capítulo 3 apresenta a descrição das ferramentas pesquisadas para o desenvolvimento do sistema e um comparativo entre elas. O capítulo 4 aborda a definição de repositório de componentes, a aplicação de agentes sobre este domínio e a expansão para outros domínios. No capítulo 5 são apresentados os detalhes da modelagem e desenvolvimento do sistema multi-agentes. Por último, no capítulo 6 são feitas algumas considerações sobre o estudo e desenvolvimento realizados e são propostos trabalhos futuros.

2. Agentes de Software

2.1. Introdução

O modelo de desenvolvimento baseado em agentes tem se tornado cada vez mais importante na solução de sistemas complexos e distribuídos. A capacidade de autonomia, pró-atividade e reatividade possibilita que os agentes executem tarefas de forma inteligente e independente. Interações entre agentes que trabalham juntos para atingir um objetivo em comum vêm reforçando e amadurecendo o conceito de sistemas multi-agentes. Várias metodologias, arquiteturas e ferramentas estão sendo desenvolvidas para facilitar o desenvolvimento destes (AYRES, 2003; QUINTÃO, 2008; SOSA, 2007; PEROTTO, 2002; CAFARATE, 2008).

Em sistemas multi-agentes as aplicações são projetadas e desenvolvidas nos termos das entidades autônomas do software (agentes), que através de protocolos e linguagens de alto-nível interagem entre si para atingir seus objetivos. Agentes podem ser simples como sub-rotinas, mas geralmente são entidades maiores com algum controle de persistência (por exemplo, *threads* de controle distinto dentro de um único espaço de endereço, processos distintos em uma única máquina, ou processos separados em máquinas diferentes).

A expressividade é uma característica importante da linguagem usada por agentes, pois permite a troca de dados e informação lógica, comandos individuais e scripts. Com isso é possível a comunicação de informações entre os agentes, viabilizando a realização de tarefas complexas.

A programação orientada a agentes é um paradigma de desenvolvimento relativamente novo, que traz consigo conceitos de teorias de inteligência artificial distribuída. A principal diferença entre a programação orientada a agentes e a programação orientada a objetos é quanto à linguagem da interface. Em geral, na programação orientada a objetos, o significado da mensagem pode variar de um objeto para outro. Na tecnologia de programação baseada em agentes, os agentes usam uma linguagem comum com uma semântica independente do agente.

2.2. Definição

Segundo Jennings (2000), agente de software é um sistema encapsulado que está situado em um ambiente e que apresenta características como flexibilidade e autonomia para atingir os seus objetivos. Duas das principais diferenças entre os agentes e os modelos computacionais tradicionais são (JENNINGS, 2000):

- A forma de interação nos sistemas tradicionais é estaticamente estruturada, onde cada método ou função só possui o conhecimento interno do seu funcionamento, não sendo capaz de se adaptar às diferentes situações do sistema. No caso dos agentes, estas interações são em nível de conhecimento, ou seja, eles possuem o conhecimento e a inteligência de se adaptarem às diferentes situações ocorridas no ambiente em que o sistema encontra-se trabalhando. Isto ocorre, porque as interações são compreendidas em termos de quais objetivos precisam ser atingidos, em qual momento e por quem;
- O fato dos agentes serem flexíveis, solucionadores de problemas e lidarem com um ambiente em que só possuam um controle parcial, faz com que as suas interações precisem ser manipuladas de forma flexível também, uma vez que o ambiente de um sistema pode estar em constantes mudanças. O mesmo não ocorre nos sistemas tradicionais pelo fato destes lidarem com informações estáticas e não possuírem flexibilidade o suficiente para ser pró-ativo, por exemplo.

De acordo com Russel e Norvig (1995), um agente é uma entidade autônoma, composta de sensores e atuadores, capaz de agir de forma a alcançar resultados otimizados. O agente interage com o mundo e com outros agentes que o rodeiam através de sensores, agindo sobre o mesmo com a utilização dos executores ou efetores. Por exemplo, nos agentes humanos os olhos e ouvidos são sensores; as mãos e a boca são os executores ou efetores. A figura 1 ilustra as interações de um agente genérico com seu ambiente.

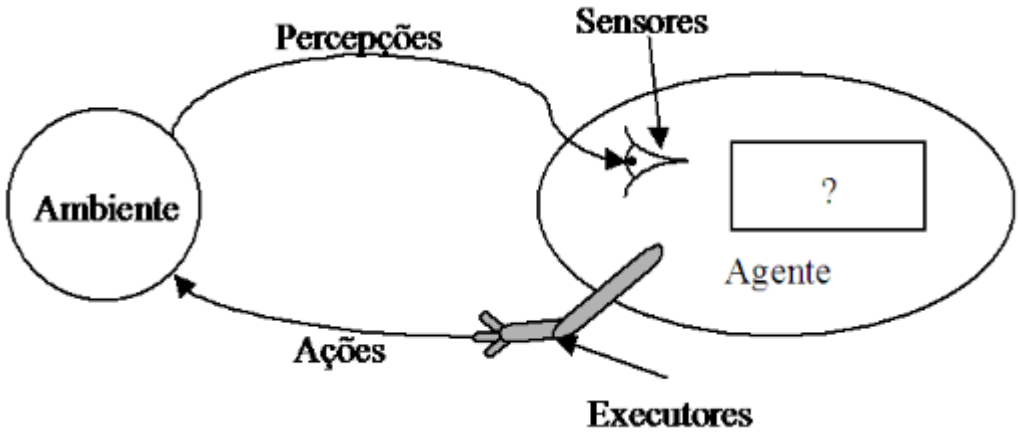

Figura 1: As interações de um agente genérico com seu ambiente (RUSSEL e NORVIG, 1995).

O agente possui uma memória interna que é atualizada com a chegada de novas percepções, sendo utilizada nos procedimentos de tomada de decisão, os quais geram ações para serem executadas. Para manter um histórico do comportamento do agente, a memória do agente também é atualizada a partir da ação selecionada (GIRARDI, 2004). A figura 2 apresenta a estrutura básica de um agente através de pseudo-código.

```
função Agente (percepção) retorna ação
  estática: memória {a memória que o agente possui do seu ambiente}
    memória ← Atualizar-Memória (memória, percepção)
    ação    ← Escolher-Melhor-Ação (memória)
    memória ← Atualizar-Memória (memória, ação)
retornar ação
```

Figura 2: A estrutura genérica de um agente (RUSSEL e NORVIG, 1995).

De modo geral, produzem ações e percepções sem requerer intervenção humana constante. Numa abordagem aplicada à Inteligência Artificial, por exemplo, um agente ideal deveria funcionar continuamente e adquirir experiências e conhecimentos sobre o ambiente que está interagindo, possibilitando a tomada de decisões a partir de diferentes situações.

2.3. Características

Para Wooldridge (1999), reatividade, pró-atividade e habilidade social são características suficientes para classificar um agente como inteligente. A seguir são descritas estas características:

- Reatividade: o agente deve perceber seu ambiente e responder às mudanças ocorridas. Em alguns casos o agente fica em estado de espera e só é ativado se algum evento específico ocorrer no ambiente;
- Pró-atividade: não agem simplesmente em resposta ao seu ambiente, eles são capazes de atingir suas metas tomando iniciativas para o cumprimento das mesmas;
- Habilidade social: interagem com outros agentes e em alguns casos com seres humanos por algum tipo de linguagem de comunicação.

Um agente pode ainda ter várias outras características, não precisando ter todas elas. Existem agentes que possuem muitas e outros que possuem todas. O consenso é que essas características tornam um agente diferente de simples programas e objetos. Algumas das características são descritas a seguir (WOOLDRIDGE, 1999):

- Adaptabilidade: agentes com capacidade de ajustar-se aos hábitos, métodos de trabalho e preferências de seus usuários;
- Autonomia: capacidade de agir sem intervenção externa direta, baseando suas ações não só no seu conhecimento embutido, mas também em seu conhecimento sobre o ambiente;
- Benevolência: suposição de que os agentes não têm objetivos contraditórios e que todo agente consequentemente sempre tentará fazer o que lhe é solicitado;
- Colaboração: um agente não deve aceitar (e executar) instruções sem considerações. Ele deve levar em conta que o usuário humano comete erros, omite informações importantes e/ou fornece informações ambíguas. Neste caso, um agente deve checar estas ocorrências fazendo perguntas ao usuário. Deve ser permitido a um agente recusar executar certas tarefas que possam sobrecarregar a rede ou causar danos a outros usuários;

- Degradação gradual: capacidade do agente de executar parte de uma tarefa quando existe incompatibilidade na comunicação ou domínio;
- Flexibilidade: agente capaz de fazer uma escolha dinâmica das ações e da sequência de execução das mesmas, em um determinado estado do ambiente;
- Inteligência: é o conjunto de recursos, atributos e características que habilitam o agente a decidir que ações executar, bem como a capacidade de tratar ambiguidades. O raciocínio desenvolve-se através de regras, conhecimento e evolução artificial;
- Mobilidade: capacidade de um agente se mover de um ambiente (máquina) para outro;
- Planejamento: é a habilidade de sintetizar e escolher entre diferentes opções de ações desejadas para atingir os objetivos;
- Representabilidade: um agente que representa o usuário em suas ações.

2.4. Classificação

A classificação de agentes nos permite visualizá-los de forma mais ampla. Existem diversas classificações de agentes, como por exemplo: quanto ao nível de inteligência, quanto à tarefa que executam, quanto à mobilidade, quanto à aquisição de inteligência e quanto à ênfase dada a alguns atributos primários. A seguir são apresentadas estas classificações e também uma classificação geral dos agentes.

2.4.1. Nível de Inteligência

Segundo Nwana (1996), os agentes apresentam diferentes níveis de inteligência, sendo classificados nos seguintes níveis:

- Baixo: agentes que desempenham tarefas rotineiras, disparadas por eventos externos. Estes agentes executam redes de regras complexas, mas não se adaptam a mudanças;
- Médio: utilizam uma base de conhecimento para desenvolver raciocínio em eventos monitorados. Podem se adaptar a mudanças de condições na base de conhecimento e manipular as novas condições;

- Alto: agentes que utilizam aprendizado e raciocínio na base de conhecimento. Aprendem com o comportamento do usuário e podem se adaptar a mudanças.

2.4.2. Tarefa que Executam

De acordo com Jennings (1995), os agentes podem ser classificados quanto à tarefa que executam, da seguinte forma:

- Gopher: agentes que executam tarefas simples, baseando-se em suposições e regras pré-especificadas. Por exemplo, agentes que avisam ao usuário quando o preço médio de uma determinada ação varia em 10% ou mais;
- Prestadores de serviço: executam tarefas de alto nível, quando requisitadas pelo usuário. Um agente que marca reuniões de viagem para o usuário pode ser considerado um exemplo de agente prestador de serviços;
- Pró-ativo: agentes que desempenham tarefas mais complexas. Podem pesquisar informações, filtrar dados e também executar tarefas para o usuário sem que o mesmo as requisite, sempre que isto for julgado apropriado. Por exemplo, um agente monitor de *newsgroup* é um agente pró-ativo.

2.4.3. Mobilidade

Agentes podem ser classificados de acordo com sua capacidade de se mover na rede, como descrito por Nwana (1996):

- Agentes estáticos: são agentes que não podem se mover pela rede, atuando apenas localmente;
- Agentes móveis: agentes que possuem a habilidade de se mover pela rede.

2.4.4. Aquisição de Inteligência

Quanto à aquisição de inteligência, podemos classificá-los da seguinte forma (NWANA, 1996):

- Agentes deliberativos: possuem um modelo de raciocínio e um modelo simbólico e interno, utilizado para tomar decisões e executar tarefas necessárias para alcançar seus objetivos. Também são chamados de simbólicos ou cognitivos;
- Agentes reativos: executam apenas quando estimulados, em resposta ao estado atual do ambiente no qual estão inseridos. Não possuem um modelo simbólico e interno dos mesmos. Também são denominados reflexivos.

2.4.5. Ênfase Dada a Alguns Atributos Primários

Também de acordo com Nwana (1996), alguns atributos primários e ideais (cooperação, aprendizado e autonomia) podem ser utilizados para classificar agentes em: agentes inteligentes, agentes de aprendizado colaborativo, agentes de interface e agentes colaborativos. A figura 3 apresenta esta classificação.

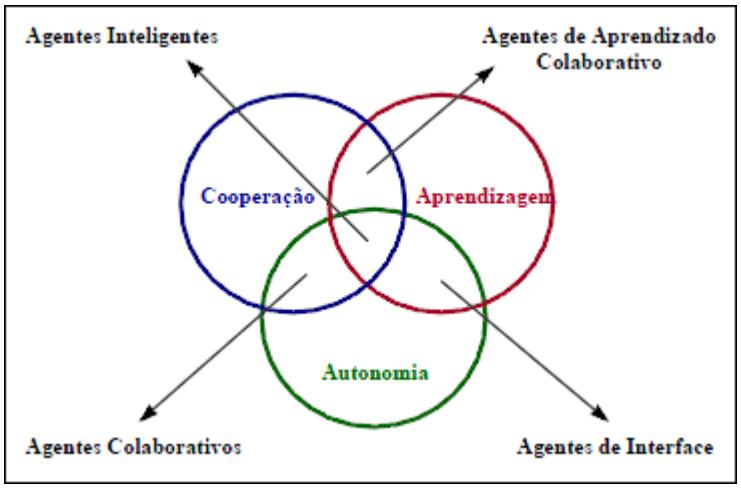

Figura 3: Classificação de agentes quanto a ênfase dada a alguns atributos primários (NWANA, 1996).

Contudo, estas diferenças não são definitivas. Por exemplo, os agentes colaborativos possuem mais ênfase sobre as propriedades de cooperação e autonomia do que sobre aprendizado, mas não significa que eles nunca aprendem.

2.4.6. Classificação Geral dos Agentes

Tendo como base as classes de agentes apresentadas anteriormente, identificamos vários tipos de agentes, que são classificados em (NWANA, 1996):

- Agentes móveis: agentes que tem a capacidade de se mover por uma rede interna local ou pela *web*, transportando-se pelas plataformas levando dados e códigos. Seu uso tem crescido devido a alguns fatos como uma heterogeneidade cada vez maior das redes e seu grande auxílio em tomadas de decisão baseadas em grandes quantidades de informação;
- Agentes situados ou estacionários: são fixos em um mesmo ambiente ou plataforma. Não se movimentam através de uma rede interna e nem pela *web*;
- Agentes competitivos: são agentes que "competem" entre si para a realização de seus objetivos ou tarefas;
- Agentes coordenados ou colaborativos: realizam tarefas específicas, porém coordenando-as entre si de forma que suas atividades se completem;
- Agentes reativos: reagem a estímulos sem ter memória do que já foi realizado no passado e nem previsão da ação a ser tomada no futuro. Não têm representação do seu ambiente ou de outros agentes e são incapazes de prever e antecipar ações. A força de um agente reativo vem da capacidade de formar um grupo e construir colônias capazes de adaptar-se a um ambiente;
- Agentes cognitivos: capacidade de raciocinar sobre as ações tomadas no passado e planejar ações a serem tomadas no futuro. Têm objetivos e planos explícitos, os quais permitem atingir seu objetivo final. Cada agente deve ter uma base de conhecimento disponível, que compreende todos os dados e todo o *"know-how*" para realizar suas tarefas e interagir com outros agentes e com o próprio ambiente. Sua representação interna e seus mecanismos de inferência o permitem atuar independentemente dos outros agentes e lhe proporciona uma grande flexibilidade na forma de expressão de seu comportamento. Devido a sua capacidade de raciocínio baseado nas representações do mundo, são capazes de ao mesmo tempo memorizar situações, analisá-las e prever possíveis reações para suas ações;
- Agentes de interface: são agentes que interagem com o usuário, recebendo especificações do usuário e entregando resultados;

- Agentes autônomos: podem interagir independente e efetivamente com seus ambientes. Não precisam necessariamente do usuário;
- Agentes de informação: acessam fontes de informações, sendo capazes de colecionar e manipular informações obtidas destas fontes para responder consultas solicitadas pelo usuário ou por outros agentes;
- Agentes híbridos: são os agentes que combinam a filosofia de um ou mais tipos de agentes.

2.5. Comunicação entre Agentes

A comunicação é fundamental para permitir que haja colaboração, negociação e cooperação entre agentes. Ela deve ser disciplinada para que os objetivos sejam alcançados, necessitando assim de uma linguagem que seja entendida pelos agentes presentes no ambiente. O principal objetivo da comunicação é a troca do conhecimento com os outros agentes e a coordenação de atividades entre eles.

Agentes podem trocar mensagens diretamente, se comunicar através de um agente facilitador especial, utilizar uma comunicação por difusão de mensagens (*broadcast*) e também podem utilizar o modelo de comunicação através de *blackboard* ou quadro-negro. A seguir são listados cada um destes tipos de comunicação (BAKER, 1997):

- Comunicação direta: comunicação via troca de mensagens direta, ou seja, cada agente comunica diretamente com qualquer outro agente sem qualquer intermédio. É necessário que cada agente envolvido tenha conhecimento da existência dos outros agentes e da forma como endereçar mensagens para eles. A vantagem da comunicação direta é que não existe um agente coordenador da comunicação. Esses agentes podem levar a um "gargalo" ou até ao bloqueio do sistema caso haja um grande número de troca de mensagens. As desvantagens são o custo da comunicação que se torna grande, principalmente quando há um grande número de agentes no sistema, e a própria implementação que se torna muito complexa em comparação às outras formas de comunicação;
- Comunicação assistida: também denominada comunicação por sistemas federados. Os agentes utilizam algum sistema ou agente especial para coordenar suas atividades. É definida uma estrutura hierárquica de agentes e a troca de mensagens

ocorre através de agentes especiais, que são os agentes facilitadores ou mediadores. As vantagens são a diminuição do custo e da complexidade necessária aos agentes individuais na realização da comunicação. Em geral é utilizada quando o número de agentes dentro do sistema é muito grande;

- Comunicação por difusão de mensagens (*broadcast*): utilizada quando a mensagem deve ser enviada para todos os agentes do ambiente ou quando o agente remetente não tem conhecimento do agente destinatário ou seu endereço. Nesse tipo de comunicação todos os agentes recebem a mensagem enviada;
- Comunicação por quadro-negro (*blackboard*): comunicação muito utilizada na Inteligência Artificial como modelo de memória compartilhada. Funciona como um repositório onde os agentes escrevem mensagens a outros agentes e buscam informações sobre o ambiente.

2.6. Padrão FIPA

A FIPA (*Foundation for Intelligent Physical Agents*) foi fundada em 1996 em Genebra. É uma fundação sem fins lucrativos direcionada à produção de padrões para a interoperabilidade de agentes heterogêneos e interativos em sistemas baseados em agentes. Tem como missão facilitar a interligação de agentes e sistemas multi-agentes entre múltiplos fornecedores de ambientes (FIPA, 2010).

É composta por companhias e organizações que se reúnem para discutir e produzir padrões para a tecnologia de agentes. Propõe uma arquitetura básica para agentes inteligentes e uma linguagem de comunicação de agentes chamada FIPA ACL (*Agent Communication Language*), cuja semântica é definida por uma linguagem formal chamada FIPA SL (*Semantic Language*). Protocolos de interação de mensagens FIPA, como o *Contract Net* (FIPA, 2010), estão implementados em plataformas de desenvolvimento de sistemas multi-agentes, chamados FIPA *compliants*, que obedecem ao padrão FIPA, como o JADE (*Java Agent DEvelopment Framework*) (JADE, 2010).

Em FIPA (2010) é apresentada uma especificação de um DTD (*Document Type Definition*) que codifica uma mensagem FIPA em uma mensagem no formato XML (*Extensible Markup Language*) (XML, 2010), ampliando as possibilidades de comunicação, e com isso promovendo a interoperabilidade entre as aplicações. A

vantagem em usar XML se deve ao fato deste se destacar como um formato para integração e comunicação na internet.

2.7. Sistemas Multi-Agentes

Sistemas multi-agentes são sistemas compostos de múltiplos agentes que interagem por meio de uma linguagem de alto nível, com o intuito de atingir os objetivos do sistema (BERGENTI *et al.*, 2002). Podemos destacar algumas justificativas para o uso do paradigma orientado a agentes no desenvolvimento de softwares, de acordo com Jennings e Wooldridge (1998):

- Sistemas abertos: um sistema aberto é aquele no qual sua estrutura é capaz de mudar dinamicamente. Isso resulta na dificuldade de definir em tempo de projeto quais componentes pertencerão ao sistema e como irão interagir. Um exemplo de sistema aberto é a internet, que se expande em tamanho e complexidade. Para operar efetivamente, esses sistemas devem ser capazes de decidir de forma autônoma e flexível;
- Distribuição de dados e controle: em muitos sistemas o controle global é realizado em diversos nós, geralmente distribuídos geograficamente. Com o intuito de trabalharem em conjunto alcançando maior eficiência, esses nós devem possuir a habilidade de interagir autonomamente entre eles;
- Sistemas legados: uma maneira natural em incorporar sistemas legados em modernos sistemas de informação distribuídos, é envolvendo-os em uma camada de agente, permitindo a interação deles com outros agentes.

Podemos destacar também motivos que incentivam o uso de sistemas multi-agentes ao invés de aplicações compostas por um único agente, o que leva à obtenção de um software mais robusto e eficiente (ODELL, 2000):

- Um só agente poderia ser construído para realizar todas as tarefas em um determinado ambiente, mas isso pode representar um gargalo em velocidade, manutenibilidade, confiabilidade, etc. Dividir as funcionalidades do sistema entre vários agentes permite uma aplicação modular, flexível, modificável e extensível;

- O conhecimento especializado geralmente não é disponível em apenas um único agente. O conhecimento sendo distribuído em vários agentes pode ser integrado para uma maior completude quando for necessária flexibilidade.

2.8. Áreas de Aplicações de Sistemas Multi-Agentes

Sistemas orientados a agentes estão sendo cada vez mais utilizados para resolução de problemas no mundo real. A seguir destacam-se algumas áreas de aplicações de sistemas multi-agentes (LUCENA *et al.*, 2004):

2.8.1. Aplicações Industriais

- Fabricação: projeto e configuração de produtos de fabricação, projeto colaborativo, cronograma e controle de operações de fabricação de um robô e determinação de sequências de produção para uma fábrica;
- Telecomunicações: controle de rede, transmissão e chaveamento, gerência de serviço e gerência de rede;
- Controle de tráfego aéreo: agentes utilizados para representar os equipamentos de aviação, bem como diversos sistemas de controle de tráfego aéreo em operação;
- Sistemas de transporte: o domínio de gerência de tráfego e transporte é adequado para uma abordagem baseada em agentes devido a sua natureza geograficamente distribuída.

2.8.2. Aplicações Comerciais

- Gerência de informação: mecanismos de busca são realizados por agentes, que agem autonomamente para buscar na web em benefício de algum usuário. É provavelmente uma das áreas mais ativas para aplicações de agente;
- Comércio eletrônico: agentes têm sido largamente utilizados para automatizar estágios de uma compra na internet, por exemplo, encontrando especificações sobre o produto, apresentando novidades, procurando por ofertas especiais e descontos;

- Gerência de processo de negócio: organizações têm procurado desenvolver sistemas de Tecnologia da Informação para dar assistência a vários aspectos da gerência de processos de negócio.

2.8.3. Aplicações de Entretenimento

Agentes são muito utilizados em jogos de computador, cinema interativo e aplicações de realidade virtual. Estas áreas tendem a ser cheias de caracteres animados autônomos, que podem naturalmente ser implementados como agentes.

2.8.4. Aplicações Médicas

Incluem a área de monitoração de pacientes e plano de saúde, entre outras.

2.9. Considerações Finais

Neste capítulo foram abordados conceitos de agentes de softwares, suas características, classificações e tipos de comunicação entre agentes. Também foi apresentado o padrão FIPA, além de justificativas para o uso do paradigma orientado a agentes. Por fim, foram destacadas algumas áreas em que os sistemas multi-agentes são utilizados.

3. Linguagens de Modelagem e Frameworks de Desenvolvimento de Sistemas Multi-Agentes

3.1. Introdução

Atualmente existem várias linguagens de modelagem de sistemas multi-agentes e frameworks para o desenvolvimento destes. O objetivo é que estas ferramentas possam representar as informações que os agentes possuem sobre o seu ambiente, suas metas, tarefas que devem realizar, além do modo como devem interagir entre si e com o seu ambiente.

Serão apresentadas a seguir algumas destas ferramentas, que são de suma importância no processo de modelagem e desenvolvimento de sistemas multi-agentes. As linguagens de modelagem descritas são AUML e ANote, enquanto os frameworks de desenvolvimento abordados são JADE e Cougaar.

3.2. Linguagens de Modelagem de Sistemas Multi-Agentes

Devido à necessidade de modelagem das entidades de um sistema multi-agente e pelo fato da UML não ter sido idealizada de forma a apoiar a modelagem de sistemas multi-agentes, houve o surgimento de linguagens de modelagem como a AUML e ANote.

3.2.1. AUML

A AUML (*Agent Unified Modelling Language*) estende a UML (*Unified Modelling Language*) para dar suporte ao desenvolvimento de sistemas orientados a agentes (ODELL, 1999). Fornece modelos para capturar o aspecto dinâmico das interações inter-agente, mas ainda não fornece extensões para capturar a organização estrutural de agentes.

A linguagem AUML provê uma representação de troca de mensagens através do seu diagrama de protocolo de interação, que estende os diagramas de sequência e de estado da UML em vários pontos. É considerado o seu principal diagrama, pois com ele é possível identificar os tipos de mensagens (padrão FIPA-ACL) que estão sendo trocadas entre os agentes, o que facilita a compreensão de como os agentes estão se comunicando no sistema. A figura 4 apresenta um exemplo do diagrama de protocolo de interação, especificando a troca de mensagens entre o “Agente 1” e o “Agente 2”:

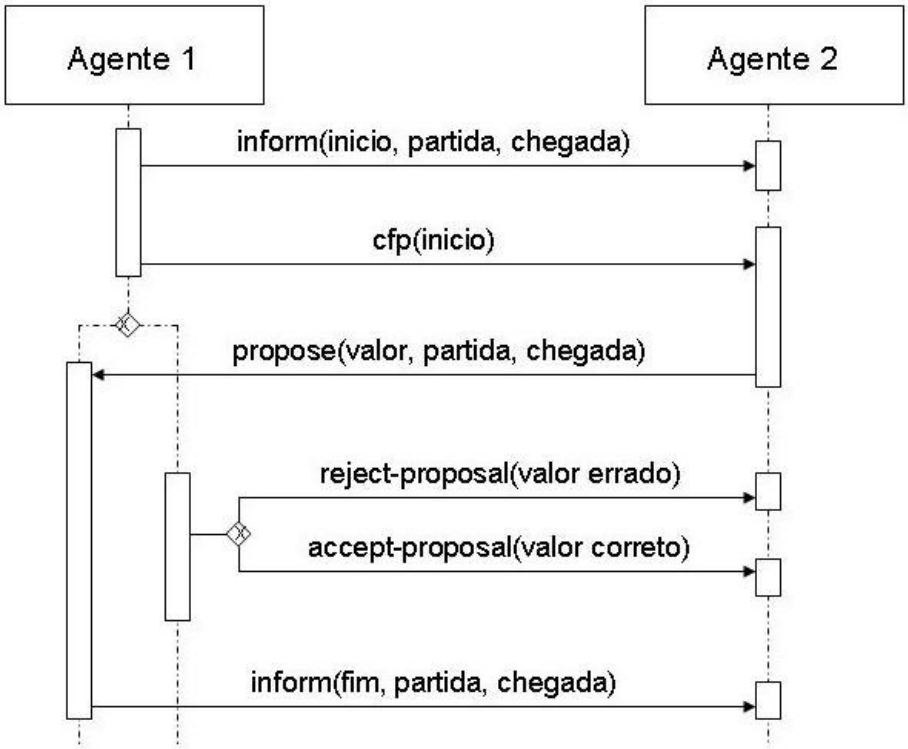

Figura 4: Diagrama de protocolo de interação da AUML (SANTOS, 2007).

Outro diagrama que faz parte da AUML é o diagrama de classes, o qual representa um agente dentro do sistema. São descritos os protocolos de comunicação que ele utilizará, a lista de agentes que podem iniciar uma colaboração e também a descrição da ontologia que o agente pode utilizar durante a comunicação com outros agentes. Uma ontologia, de forma geral, define termos de um vocabulário controlado para facilitar a comunicação. A figura 5 exemplifica este diagrama:

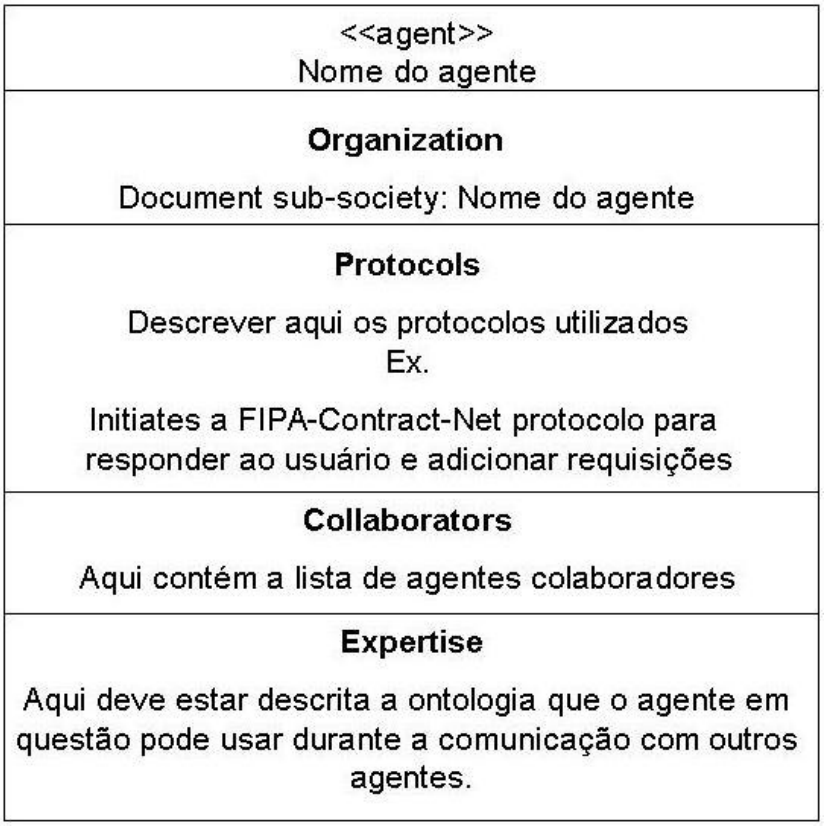

Figura 5: Diagrama de classe da AUML (SANTOS, 2007).

Por fim, temos o diagrama de *deployment* ou de posicionamento, que possibilita uma visão de como o sistema deve ser estruturado, mostrando qual a plataforma que está sendo trabalhada, onde se localiza o servidor, etc. Com ele é possível representar o conhecimento que um agente tem de outro agente ou o conhecimento que um agente tem de outro artefato do sistema através da *tag* <<*acquaintance*>>. A figura 6 apresenta o diagrama de *deployment*:

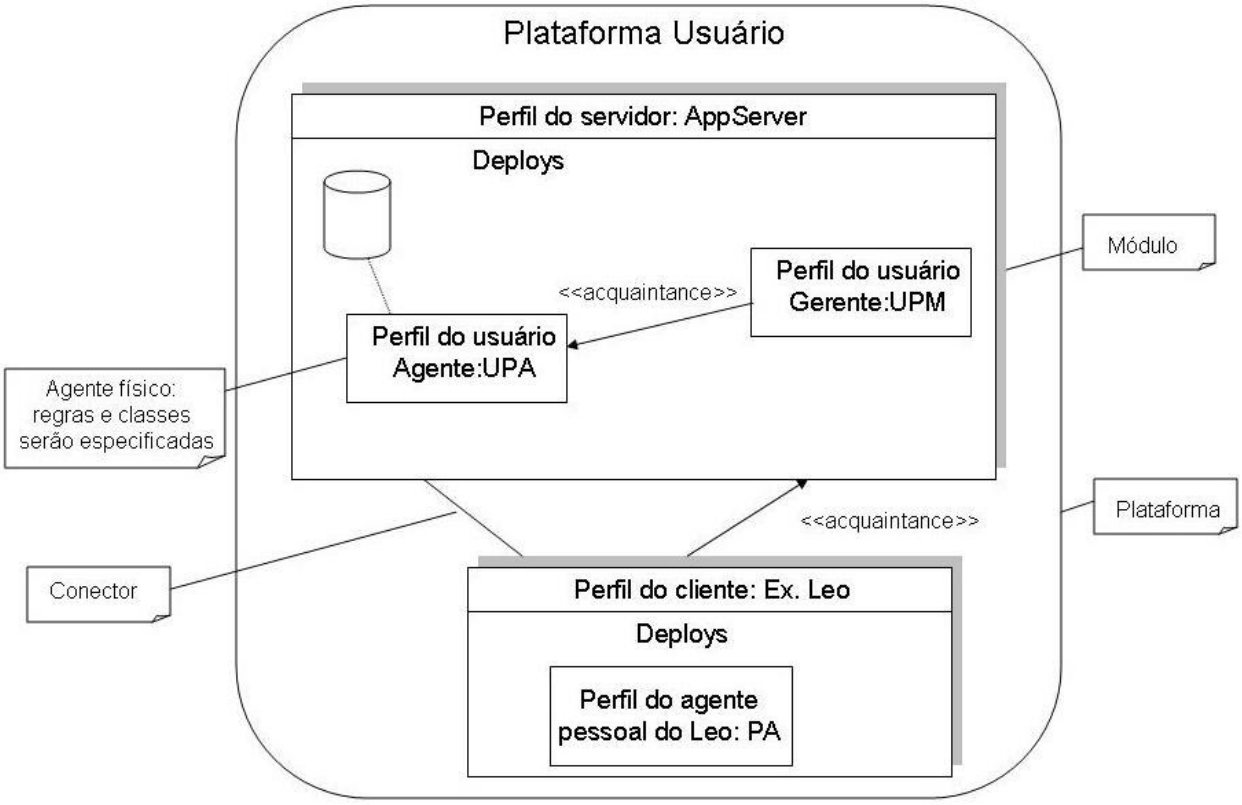

Figura 6: Diagrama de *deployment* da AUML (SANTOS, 2007).

3.2.2. ANote

A linguagem de modelagem ANote foi desenvolvida para oferecer uma maneira padrão de descrever os conceitos relacionados ao processo de modelagem de sistemas multi-agentes. Oferece uma estrutura de software orientada a agente, ou seja, tem o agente como o principal elemento de modelagem.

O ANote possui um meta-modelo que funciona como guia para a modelagem do sistema. Um meta-modelo é uma representação dos tipos de entidades que podem existir em um modelo, suas relações e restrições de aplicações (CHOREN e LUCENA, 2005). A figura 7 apresenta o meta-modelo:

Figura 7: Meta-modelo do ANote (CHOREN e LUCENA, 2005).

O ANote apresenta sete diagramas que possibilitam a modelagem de sistemas multi-agentes. Cada um dos diagramas representa uma visão específica do sistema e a combinação destas visões fornece um conhecimento extensivo sobre a especificação do sistema.

Os diagramas são classificados em diagramas estáticos e dinâmicos. Os diagramas estáticos são os diagramas de objetivo, agente e ambiente; os diagramas dinâmicos são os de cenário, planejamento e interação. O diagrama de organização não se encaixa nesta classificação já que é o responsável pela definição da estrutura do sistema que não é definida por agente de software. A seguir são apresentados os sete diagramas (CHOREN e LUCENA, 2005):

- Diagrama de objetivo: são especificados os objetivos do sistema, que são definidos como serviços e funcionalidades. Um objetivo é representado por um retângulo com as pontas arredondadas e são interligados através de setas que vão do objetivo mais específico para o mais geral.

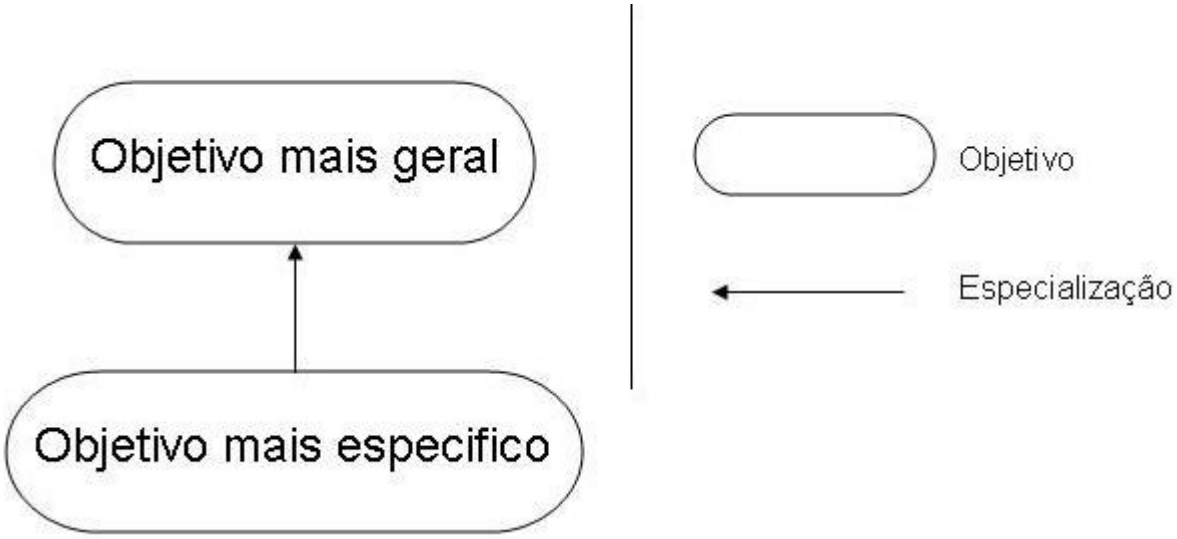

Figura 8: Elementos do diagrama de objetivo do ANote (SANTOS, 2007).

- Diagrama de agente: especifica os agentes que existem no sistema e as relações entre eles. O principal elemento neste diagrama é a classe agente que é representada por um retângulo e uma *tag* A. Os agentes são interligados através de uma linha.

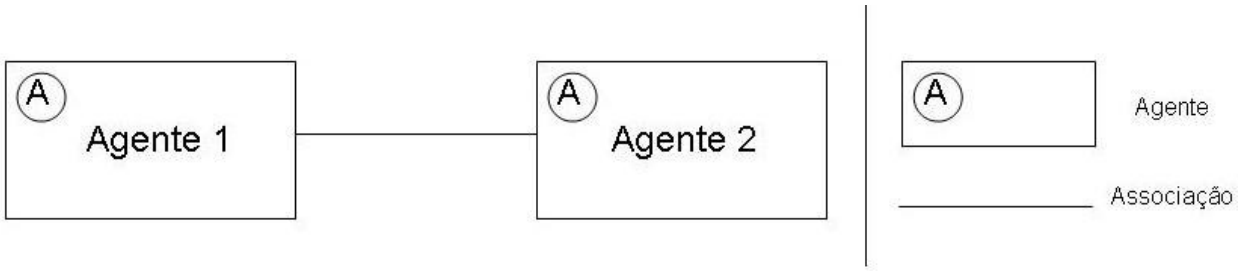

Figura 9: Elementos do diagrama de agente do ANote (SANTOS, 2007).

- Diagrama de ambiente: especifica componentes do sistema que não são agentes, representando o ambiente de componentes onde os agentes interagem. Para isso modela a visão utilizando a UML. Se o sistema interage com outros programas ou banco de dados, este diagrama pode representá-lo através do diagrama de componentes da UML.

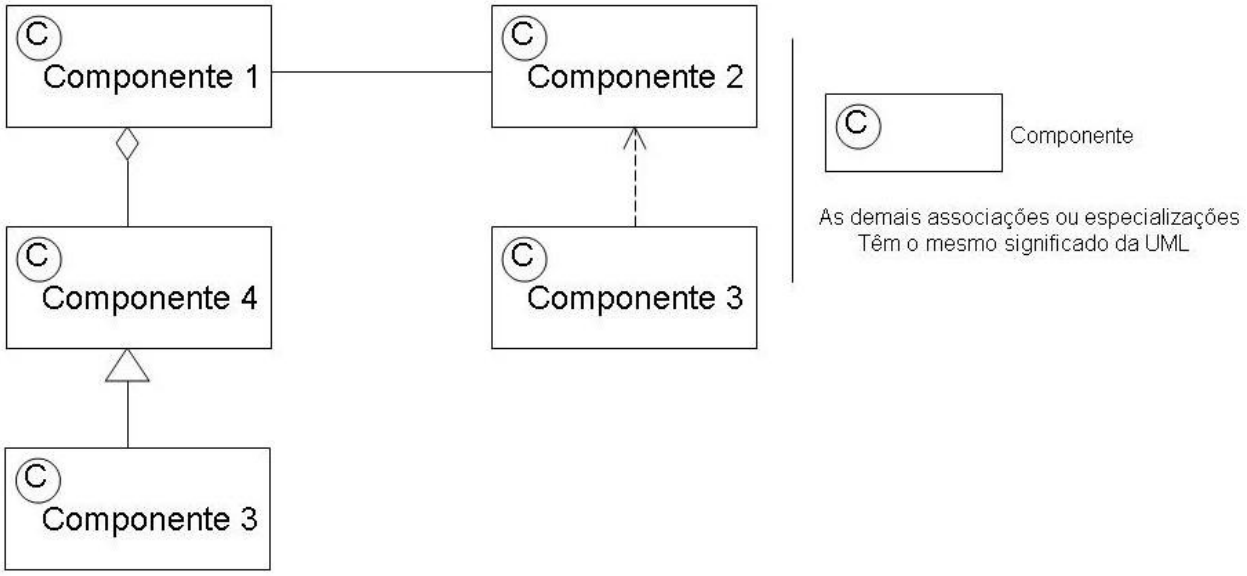

Figura 10: Elementos do diagrama de ambiente do ANote (SANTOS, 2007).

- Diagrama de organização: permite que o desenvolvedor pense no sistema multi-agente como um conjunto de componentes ou como a implementação física da unidade. Este diagrama é representado como cubos, onde estes representam organizações de agentes.

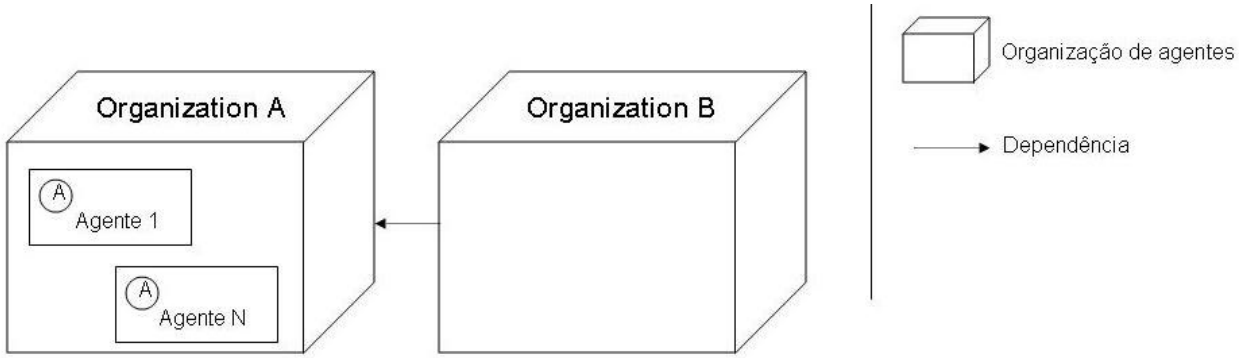

Figura 11: Elementos do diagrama de organização do ANote (SANTOS, 2007).

- Diagrama de cenário: é o diagrama que captura o comportamento do agente em um contexto específico. Diferente das demais, esta visão é constituída apenas de uma descrição. Este diagrama ajuda o desenvolvedor a elaborar os caminhos através dos objetivos dos agentes.

Objetivo do agente	
Lead Agent	Nome do agente principal
Precondition	Precondições para que o objetivo possa ser cumprido
Main Action Plan	Seqüência de ações (principais) que podem ser executadas
Interactions	Nome do agente que interage com o agente principal
Variant Plan	Seqüência de ações (secundárias) que podem ser executadas

Figura 12: Elementos do diagrama de cenário do ANote (SANTOS, 2007).

- Diagrama de planejamento: especifica o fluxo em que as ações de um agente devem ser executadas. Este diagrama é similar ao diagrama de atividades da UML.

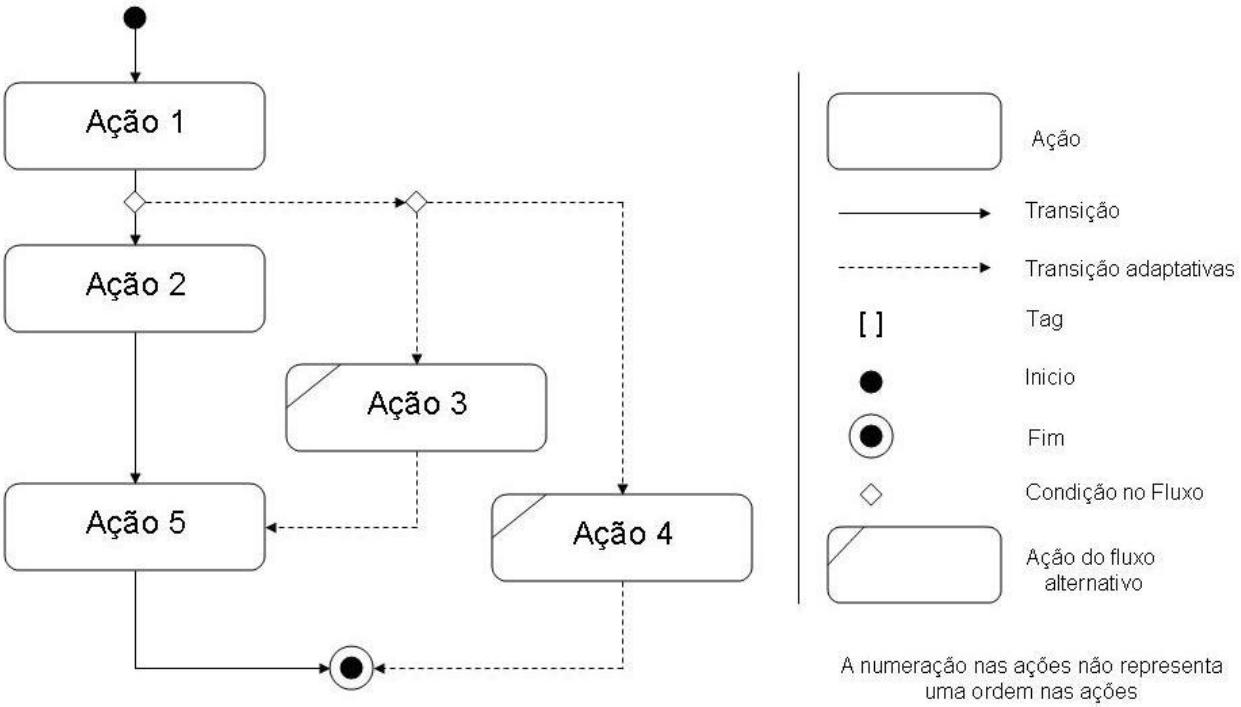

Figura 13: Elementos do diagrama de planejamento do ANote (SANTOS, 2007).

- Diagrama de comunicação: representa os agentes que enviam e recebem mensagens enquanto estão executando o plano das ações.

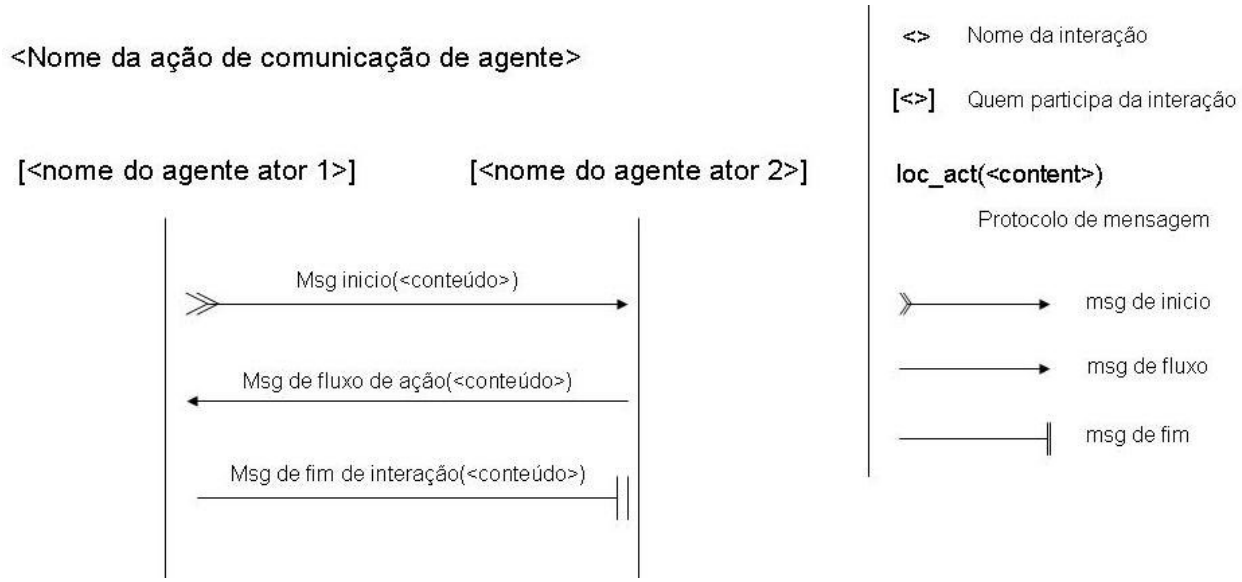

Figura 14: Elementos do diagrama de comunicação do ANote (SANTOS, 2007).

3.2.3. Considerações Sobre as Linguagens de Modelagem

A AUML é uma linguagem que representa muito bem a troca de mensagem entre os agentes de um sistema, inclusive respeitando o protocolo FIPA-ACL. O diagrama de interação da AUML é o responsável por modelar a troca de mensagens entre os agentes em um sistema multi-agentes. Os demais diagramas da AUML mostram uma modelagem mais alto nível determinando características do agente e do sistema. Um ponto negativo da AUML é o fato de que não é possível através dela representar as variabilidades que podem ocorrer no desenvolvimento de sistemas multi-agentes. As aplicações que devem ser instanciadas não são representadas em seus diagramas.

O ANote é uma linguagem que visa representar um sistema multi-agente de uma forma geral e abrangente de modo que todas as características de um agente possam ser representadas. Esta linguagem de modelagem de agentes apresenta sete diagramas e através destes a modelagem é capaz de expressar tudo o que se passa num sistema multi-agente. O ANote não tem uma modelagem atrelada à uma linguagem de programação ou um framework específico, o que possibilita uma representação bastante genérica. A elaboração do diagrama de cenários do ANote permite a realização de uma análise

qualitativa de cada um dos objetivos funcionais, possibilitando uma nova distribuição de acordo com o conjunto de agentes.

3.3. Frameworks de Desenvolvimento de Sistemas Multi-Agentes

A construção de sistemas multi-agentes pode ser realizada em qualquer linguagem de programação e de forma livre por seus desenvolvedores. No entanto, o uso de *frameworks* pode auxiliar nesta tarefa, permitindo aos projetistas se focarem no problema que o sistema multi-agente deve resolver. Detalhes internos como onde os agentes irão atuar e a comunicação entre eles ficam sob a responsabilidade do ambiente de execução. A seguir são apresentados os *frameworks* JADE e Cougaar.

3.3.1. JADE

O JADE (*Java Agent DEvelopment Framework*) é um *framework* que apóia o desenvolvimento de sistemas multi-agentes seguindo o padrão FIPA, realizando o papel de um *middleware*. Disponibiliza um ambiente de execução, com apoio de ferramentas gráficas, onde os agentes podem ser criados e suas ações gerenciadas. Permite também o desenvolvimento de agentes baseados em sua tecnologia, através do uso de sua biblioteca de classes (JADE, 2010).

Dentre as características de sistemas multi-agentes, o JADE fornece serviço de páginas amarelas para apoio no registro e busca por serviços oferecidos pelos agentes. Este sistema de gerenciamento de agentes provê o serviço de nomes (cada agente deve ter um nome exclusivo na plataforma em que habita). Além disso, permite criar e destruir agentes em *containers* remotos via requisição.

Quanto à troca de mensagens, o JADE utiliza uma linguagem de comunicação entre agentes com suporte ao padrão de protocolos de interação FIPA. Aliado a isto, pode-se utilizar ontologias como apoio a compreensão e aceitação das mensagens recebidas.

Cada instância do ambiente de execução JADE é chamada de *container*, o qual pode conter diversos agentes. Cada *container* é um objeto RMI (*Remote Method Invocation*) que gerencia um conjunto de agentes localmente, controlando o ciclo de vida dos agentes (criação, suspensão, destruição).

O conjunto de *containers* ativos é vinculado em uma plataforma. O *container Main* precisa estar sempre ativo em uma plataforma e todos os *containers* devem ser registrados a este. Ele é responsável pelo sistema de gerenciamento de agentes e o serviço de páginas amarelas, através dos agentes AMS (*Agent Manager System*) e DF (*Directory Facilitator*), respectivamente. A figura 15 ilustra a estrutura de *containers* e plataformas:

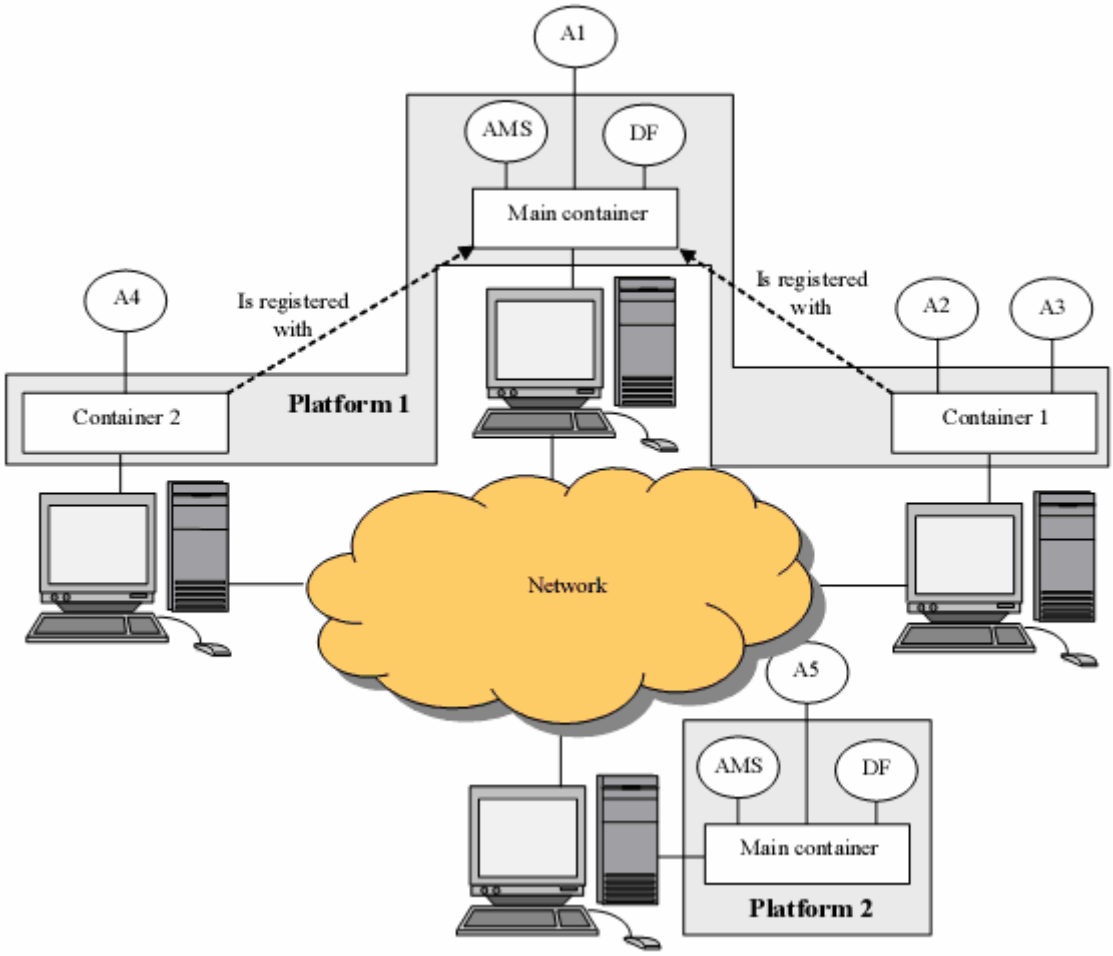

Figura 15: *Containers* e plataformas JADE (SILVA, 2007).

De acordo com a especificação FIPA, o ciclo de vida de um agente apresenta os seguintes estados [24]:

- Iniciado (*Initied*): o agente é construído, porém ainda não foi registrado no AMS e nem apresenta nome ou endereço, o que impossibilita que ele se comunique com outros agentes;
- Ativo (*Active*): o agente está registrado no AMS e possui nome e endereço, podendo acessar todas as características do JADE;

- Suspenso (*Suspended*): as atividades do agente são suspensas, onde nenhum comportamento pode ser executado;
- Aguardando (*Waiting*): o agente é bloqueado, pois está aguardando alguma ação, geralmente o recebimento de alguma mensagem;
- Excluído (*Deleted*): o agente é finalizado e não possui mais registro com o AMS;
- Em trânsito (*Transit*): um agente móvel entra neste estado quando está em processo de migração a outra localidade. O sistema continua a armazenar as mensagens que serão encaminhadas a sua nova localidade.

Na figura 16, temos a representação do ciclo de vida de agentes definido pela FIPA e seus possíveis estados:

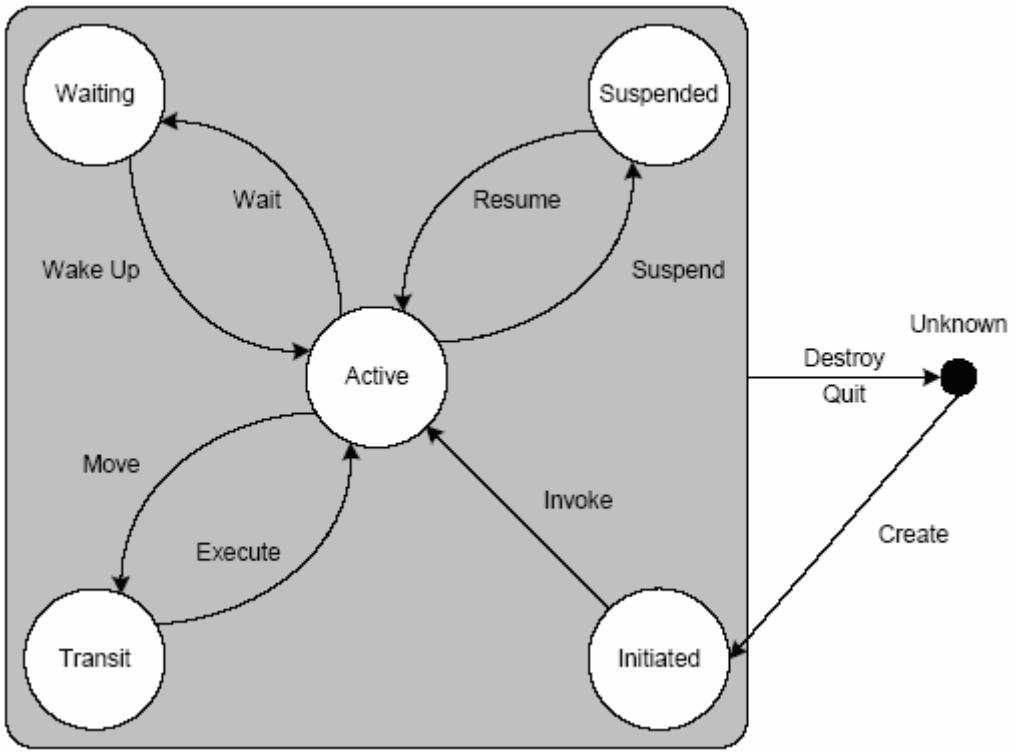

Figura 16: Ciclo de vida de agentes definido pela FIPA (SILVA, 2007).

O JADE disponibiliza um conjunto de ferramentas de suporte ao gerenciamento de sistemas multi-agentes, as quais simplificam a administração da plataforma. Cada ferramenta conta com um agente para o apoio na atividade. As ferramentas são especificadas a seguir (SILVA, 2007):

- Agente de gerenciamento remoto (RMA – *Remote Management Agent*): é uma interface gráfica para controle e gerenciamento de plataforma. O JADE mantém coerência entre múltiplos RMAs. O console RMA permite iniciar outras ferramentas JADE;
- Agente *Dummy*: ferramenta gráfica de monitoramento e *debugging*. Usando a interface gráfica é possível criar mensagens ACL (*Agent Communication Language*) e enviar a outros agentes. Também é possível mostrar uma lista de mensagens ACL enviadas e recebidas;
- Agente *Sniffer*: é um agente que pode interceptar mensagens ACL enquanto elas estão sendo transmitidas e as exibe em uma interface gráfica usando uma notação similar ao diagrama de sequências UML. É útil para testar sociedades de agentes, observando como eles trocam mensagens;
- Agente *Introspector*: é um agente que permite monitorar o ciclo de vida de outros agentes, tal como a troca de mensagens e comportamentos em execução;
- Agente *Directory Facilitator*: fornece serviços de páginas amarelas. Agentes podem registrar os seus serviços no DF, ou consultar o DF;
- Agente *LogManager*: é um agente que permite registrar informação de log em execução;
- Agente *SocketProxy*: atua em modo bidirecional entre a plataforma JADE e a conexão TCP/IP.

Uma das mais importantes características que os agentes JADE provêem é a habilidade de se comunicar. O paradigma de comunicação adotado é a troca de mensagens assíncronas. Cada agente tem uma caixa de correio (a fila de mensagens do agente) onde o JADE recebe as mensagens enviadas por outros agentes. Onde quer que uma mensagem seja enviada na fila de mensagens, o agente receptor é notificado (SILVA, 2007).

Enviar uma mensagem para outro agente exige apenas o preenchimento dos campos do objeto *ACLMessage*, e então executar o método *send* (). Um agente pode capturar mensagem de sua fila de mensagens através do método *receive* (). Este método retorna a primeira mensagem na fila (removendo-a) ou *NULL* se a fila estiver vazia. Outra possibilidade é pré-selecionar as mensagens que serão analisadas pelo agente. Para isto,

utiliza-se a classe *MessageTemplate*, onde um conjunto de critérios podem ser definidos (SILVA, 2007).

3.3.2. Cougaar

Cougaar (*Cognitive Agent Architecture*) (COUGAAR, 2010) é um *framework* baseado em Java para a construção de aplicações baseadas em agentes distribuídos, o qual foi desenvolvido pela DARPA (*Defense Advanced Research Projects Agency*).

Agentes Cougaar são executados em uma JVM (*Java Virtual Machine*), a qual se localiza em um host. JVMs provêem os serviços que os agentes locais utilizam. Estes serviços são definidos por componentes e podem envolver bibliotecas que não pertencem ao Cougaar.

Um agente é composto de um ou mais *plugins*, que definem o comportamento do agente. Caso o agente não tenha nenhum *plugin*, ele não será capaz de realizar nenhuma tarefa. Cada agente tem um *blackboard* (quadro-negro) privado que é utilizado por seus *plugins*, os quais devem comunicar entre si apenas através dele. Este serviço de quadro-negro intra-agente é chamado *BlackboardService*. A figura 17 apresenta a arquitetura Cougaar:

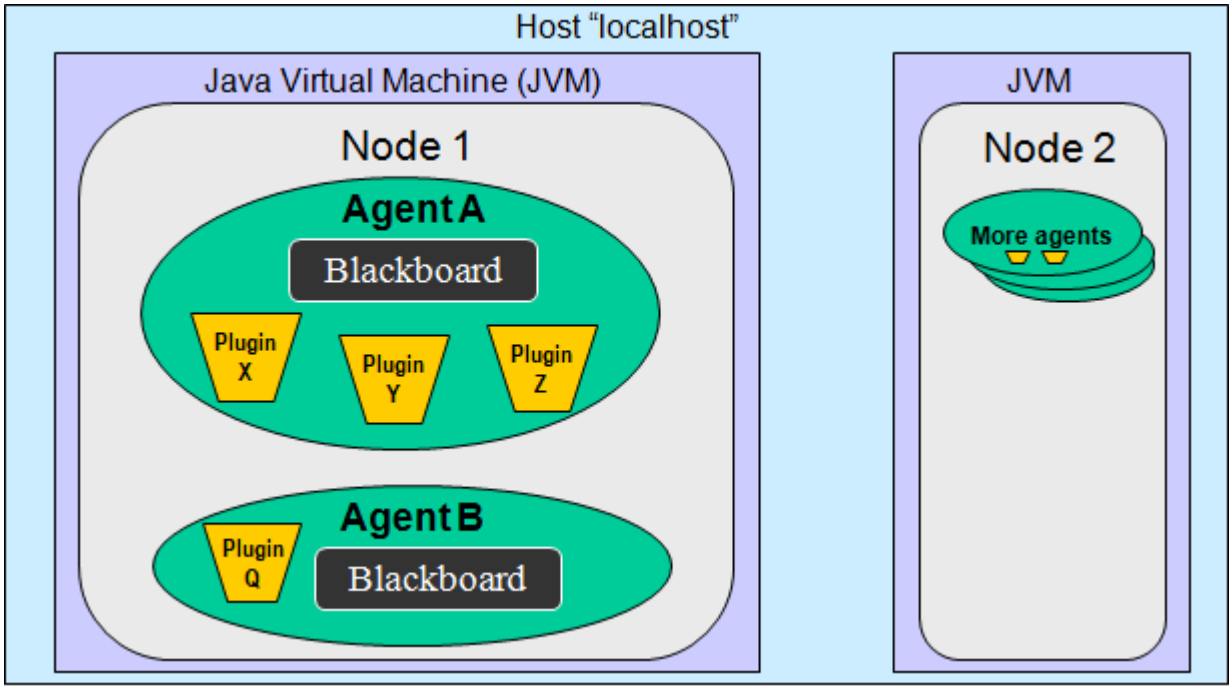

Figura 17: Representação da arquitetura Cougaar (COUGAAR, 2010).

Plugins também podem agir como *Servlets*, aguardando conexões URL remotas. Conhecido como *ServletService*, este é mais um dos serviços disponibilizados pela plataforma Cougaar. A figura 18 exemplifica este serviço:

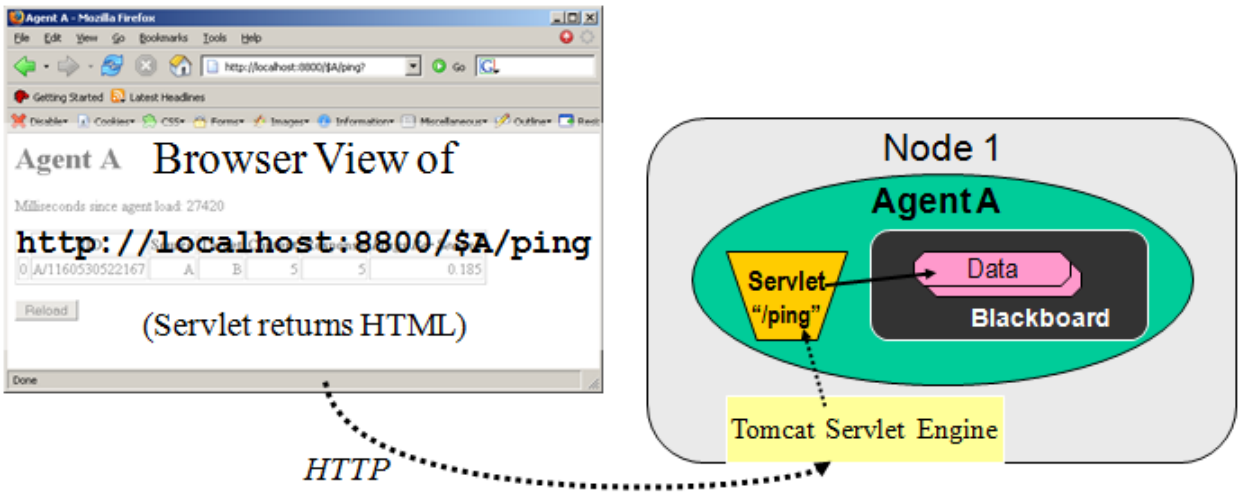

Figura 18: *ServletService* (COUGAAR, 2010).

Na figura anterior, o agente A tem um *servlet*/*plugin* que irá aguardar pelas requisições do *browser* para "http://localhost:8800/$A/ping". O *servlet* consulta o *blackboard* e disponibiliza os dados como HTML, para visualização no *browser*.

3.3.3. Comparativo entre os Frameworks JADE e Cougaar

A partir do estudo e análise das duas ferramentas pode-se concluir que o JADE é mais adequado para desenvolver sistemas multi-agentes relativamente simples, enquanto o Cougaar é mais indicado para o desenvolvimento de sistemas multi-agentes complexos e de larga escala, que necessitam de maior robustez e que sejam altamente seguros.

JADE segue os padrões da FIPA e apresenta uma linguagem de comunicação de agentes, a FIPA ACL. O desenvolvedor não precisa entrar no mérito de como ocorre a comunicação entre os agentes, já que o próprio JADE faz este controle. Em Cougaar as mensagens de interação entre os agentes são customizadas pelo desenvolvedor, não seguindo nenhum padrão.

Por apresentar ferramentas gráficas para a criação e depuração de agentes, JADE simplifica a criação de sistemas multi-agentes, proporcionando uma melhor compreensão para o desenvolvedor de como utilizar a ferramenta. Já o Cougaar não apresenta ferramentas gráficas, o que dificulta o entendimento por parte do desenvolvedor.

Ambas as ferramentas são *open-source* e no site oficial de cada uma encontra-se toda a documentação, manuais, novidades e a ferramenta para *download*.

3.4. Considerações Finais

Neste capítulo foram apresentadas as linguagens de modelagem e *frameworks* de desenvolvimento de sistemas multi-agentes analisados. A linguagem de modelagem escolhida para modelar o sistema proposto foi o ANote, pois este permite uma melhor especificação dos agentes se comparado com a AUML. Para o desenvolvimento do sistema foi escolhido o JADE, o qual apresenta ferramentas gráficas que facilitam o processo de desenvolvimento. Outra vantagem apresentada é o controle da comunicação entre os agentes que é feito pelo JADE, permitindo a visualização gráfica da sequência das mensagens trocadas pelos agentes.

4. Agentes para Repositórios de Componentes

4.1. Introdução

O termo repositório pode ser utilizado para diversos propósitos. Entre eles destacam-se (HENNINGER, 1997):

- Armazenamento de modelos: um processo de desenvolvimento envolve a geração de vários artefatos que normalmente são armazenados em locais diferentes. Um repositório possibilita a centralização do armazenamento desses artefatos e o controle de cada uma das suas versões;
- Integração de ferramentas: muitas vezes os modelos gerados durante um processo de desenvolvimento originam-se de ferramentas distintas. Assim, para a troca de informações entre elas é necessário que haja uma "linguagem" comum através da qual as mesmas possam se comunicar. O repositório pode servir para um armazenamento temporário e para a intermediação entre as ferramentas, ou seja, pode transformar o modelo de entrada de uma delas num modelo compatível para a outra;
- Manutenção de sistemas legados: um repositório pode ser útil também para o armazenamento de dados relacionados a sistemas antigos. Esses dados podem posteriormente ser analisados para se determinar os possíveis impactos gerados por modificações.

Analisadas tais funcionalidades, observa-se que o repositório é utilizado para o armazenamento de metadados. Essa restrição é o que os diferenciam dos bancos de dados.

Um importante termo no contexto é o ativo de software, o qual segundo Ezran *et al.* (1999) é qualquer artefato, passível de reutilização, produzido em qualquer fase de um processo de desenvolvimento de um sistema computacional, que deve ser utilizado seguindo certas regras de uso, sendo possível os artefatos possuírem algum ponto de customização. Assim, pode-se considerar um ativo de software artefatos de documentação, padrões de análise e projeto, normas de codificação, entre outros.

4.2. Características de um Repositório

As principais funcionalidades de um repositório são a pesquisa e a recuperação dos ativos de software. No entanto, a presença de algumas outras funcionalidades pode facilitar e, com isso, incentivar a sua utilização. Conforme proposto por Ezran *et al.* (1999), serão apresentadas, a seguir, algumas das principais funções que um repositório pode disponibilizar a seus usuários. Vale ressaltar que essas funções não são obrigatórias e variam dependendo do contexto no qual ele está inserido e das necessidades da organização ao utilizar um repositório.

- Identificação e descrição: para se descrever um ativo é possível que a ele seja atribuído um conjunto de características tais como, nome, domínio, palavras-chaves, dentre outras que os identificam e os diferenciam dos demais ativos que compõem esse mesmo repositório. Para cada um dos ativos armazenados deve ser possível identificá-lo homogeneamente dentro de um repositório, ou seja, os ativos de um mesmo tipo devem apresentar o mesmo conjunto de características. Isso não significa, contudo, que ativos diferentes devam ter os mesmos valores para esse conjunto de características;
- Inserção: um repositório deve permitir que usuários autorizados insiram novos ativos ou ainda novas versões dos mesmos. A inserção significa adicionar ao repositório o corpo e a descrição do ativo;
- Exploração do catálogo: aos usuários de um repositório deve ser permitido que explorem o catálogo de ativos para que possam conhecer e analisar as características dos ativos disponíveis;
- Pesquisa textual: um repositório deve permitir que seus usuários façam pesquisas mais específicas na descrição dos ativos. Como resultado da pesquisa será obtido um ou mais ativos que satisfaçam as condições desejadas. Observados os resultados, pode-se decidir por um maior detalhamento ou generalização dos critérios anteriores;
- Recuperação: após a identificação do ativo desejado, um repositório deve permitir que seus usuários recuperem esse ativo para que possam posteriormente utilizá-lo num processo de reuso;

- Organização e pesquisa: a funcionalidade de exploração de catálogos não é suficiente à medida que a quantidade de ativos disponíveis aumenta. Além disso, a pesquisa textual, muitas vezes, demanda muito tempo. Devido a esses fatores, alguns repositórios podem adotar novas formas de organizar as características dos ativos de modo a permitir que a pesquisa no repositório seja baseada em outros critérios;
- Histórico: é importante para o gerenciamento de um repositório que ele possa armazenar informações de uso, modificações, criação e exclusão de cada um dos ativos disponíveis. Essas informações devem montar uma base histórica que facilitará a análise e a reutilização dos mesmos;
- Mensuração: um repositório pode coletar estatísticas que facilitem o seu gerenciamento. Algumas das principais estatísticas que podem ser adotadas são: frequência de acesso ao repositório, quantidade de ativos disponíveis, taxa de recuperação, porcentagem de pesquisas bem-sucedidas, frequência com que um determinado ativo é analisado ou modificado, dentre outras;
- Controle de acesso: um repositório pode adotar uma política de segurança para que determinadas funcionalidades só estejam acessíveis a pessoas autorizadas. Por exemplo, pode-se definir para uma empresa específica uma política de segurança onde a pesquisa textual esteja disponível para todos os funcionários, mas a recuperação de ativos esteja disponível apenas aos funcionários da área de desenvolvimento;
- Gerenciamento de versões: um repositório pode conter várias versões de um mesmo ativo e, sendo assim, é recomendável que haja algum mecanismo para controlar essas versões e estabelecer o relacionamento entre elas;
- Controle de modificações: é recomendável que sejam providas algumas funcionalidades para se fazer o gerenciamento de modificações dos ativos num repositório. Essas funcionalidades incluem procedimentos para solicitação de alterações, discussões e permissão para as mesmas. É de extrema importância para a manutenção da consistência de um repositório que quaisquer intenções de modificar os ativos sejam comunicadas àquele responsável pela sua administração para que o mesmo, juntamente com outros responsáveis possam analisar as alterações solicitadas e manter a coerência entre os diversos ativos de software;

- Notificação de mudanças: é possível que a qualquer momento um ativo seja modificado ou, até mesmo, que o próprio repositório sofra algumas alterações em suas principais funcionalidades. Sendo assim, um repositório pode prover funções que notifiquem os seus usuários das modificações recentemente ocorridas tais como, inserção ou exclusão de ativos, alterações em documentos, disponibilização de novas funcionalidades, novas políticas de segurança, entre outras.

Apresentadas as principais funcionalidades de um repositório faz-se necessário analisar alguns dos aspectos não-funcionais que o caracterizam. Esses aspectos podem também, se adequadamente utilizados, possibilitar maiores incentivos para a utilização do repositório.

- Quantidade: o repositório pode ser único ou pode existir um conjunto deles (EZRAN *et al.*, 1999). O único é indicado para organizações pequenas onde não há uma grande divisão departamental e, assim, os tipos de ativos não são completamente distintos e são compreendidos por todos os seus usuários. Para organizações maiores ou geograficamente distribuídas sugere-se o uso de vários repositórios, relacionados as áreas de atuação da empresa. Nesse caso é importante que haja um gerenciamento centralizado e eficiente para evitar inconsistência nas duplicações existentes entre alguns repositórios. A utilização de vários deles é defendida por aqueles que consideram que informações desnecessárias para um determinado grupo de usuários, mas de grande valia para outro, prejudicam e desestimulam a prática do reuso nas organizações;
- Acesso pela rede: como a maioria das empresas está migrando seus modelos computacionais para modelos centralizados, um requisito típico é que o repositório possa ser acessado de qualquer ponto numa rede. Isso é particularmente importante quando seus usuários estão distribuídos geograficamente. O que normalmente se tem observado na implementação dos repositórios é a utilização de modelos cliente-servidor em 3 ou mais camadas (EZRAN *et al.*, 1999). No modelo de 3 camadas, uma é a responsável pelo armazenamento dos ativos, a outra pelo processamento das principais funcionalidades do sistema tais como pesquisa e recuperação e a terceira corresponde à aplicação cliente, por onde o repositório poderá ser acessado. A quantidade de camadas e o modo de implementação das mesmas poderão variar entre as diversas organizações.

4.3. Aplicação de Agentes para Repositório de Componentes

O objetivo da aplicação de agentes para repositórios de componentes desenvolvida neste trabalho é proporcionar um apoio ao gerenciamento de repositórios de componentes não-centralizados, porém pertencentes a uma mesma entidade, possibilitando que seus usuários sejam notificados quando houver a inserção de um novo componente em algum dos repositórios.

Para a aplicação desenvolvida foram criados dois repositórios que foram mantidos em máquinas diferentes. A criação destes repositórios ocorreu porque não foi encontrado um repositório aberto, o qual possibilitaria a execução de testes da aplicação de agentes. Os repositórios criados são bases de dados *Access* que mantêm arquivado o nome, a descrição e o conteúdo do componente.

O escopo do sistema pode ser expandido para apoiar o gerenciamento de repositórios que armazenem qualquer tipo de artefato produzido durante um projeto de software. Isto é possível pois a comunicação realizada entre os agentes é a mesma, independente de qual são as informações transmitidas entre eles. A figura 19 ilustra a aplicação:

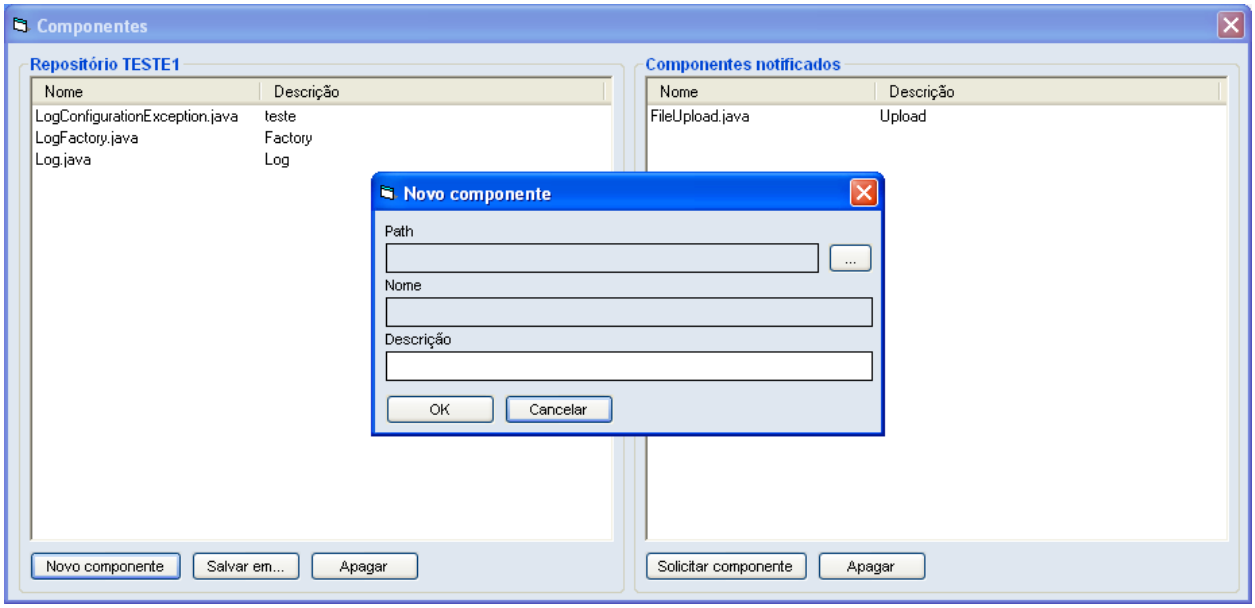

Figura 19: Aplicação de Agentes para Repositório de Componentes.

Ao clicar em “Novo componente”, o usuário deve localizar o componente que deseja adicionar no repositório, preencher sua descrição e clicar em “OK”. Após a

inserção, o novo componente é adicionado ao Repositório TESTE1 e uma notificação é feita ao outro repositório.

A notificação ocorre logo após a inserção do componente, sendo realizada através da troca de mensagens entre os agentes desenvolvidos. A inserção é permitida apenas para componentes que ainda não estão registrados no repositório. Caso o componente já exista no repositório, o usuário receberá uma mensagem de alerta. A figura 20 apresenta esta mensagem de alerta:

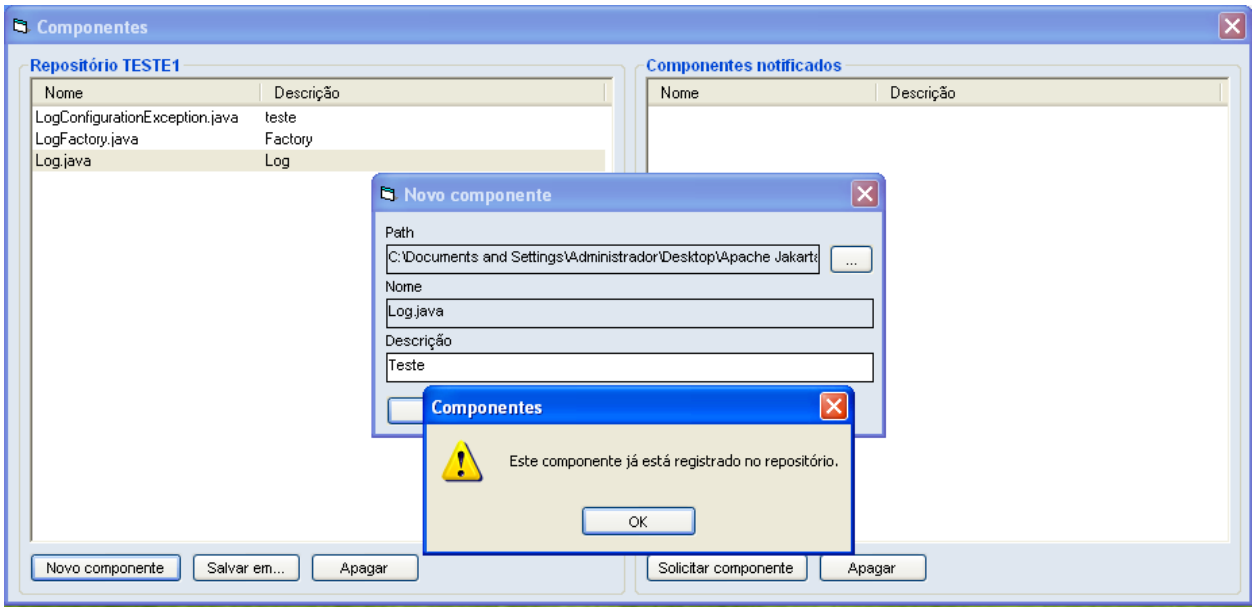

Figura 20: Mensagem de alerta sobre componente já registrado no repositório.

Na figura 19, temos um componente que foi notificado (“FileUpload.java”) ao Repositório TESTE1 a partir de um outro repositório. Caso este seja um componente de interesse, o mesmo poderá ser solicitado pelo usuário do Repositório TESTE1 a qualquer momento.

Feita a solicitação, caso o componente ainda exista no repositório ao qual pertence, ele será inserido no Repositório TESTE1, e será removido da lista de componentes notificados. Se o componente foi removido, o usuário do Repositório TESTE1 receberá uma mensagem de alerta sobre a remoção, e após clicar em “OK” o componente será removido da lista de componentes notificados. O processo de envio dos componentes entre os repositórios é feito através dos agentes. A figura 21 apresenta a imagem de alerta recebida pelo usuário:

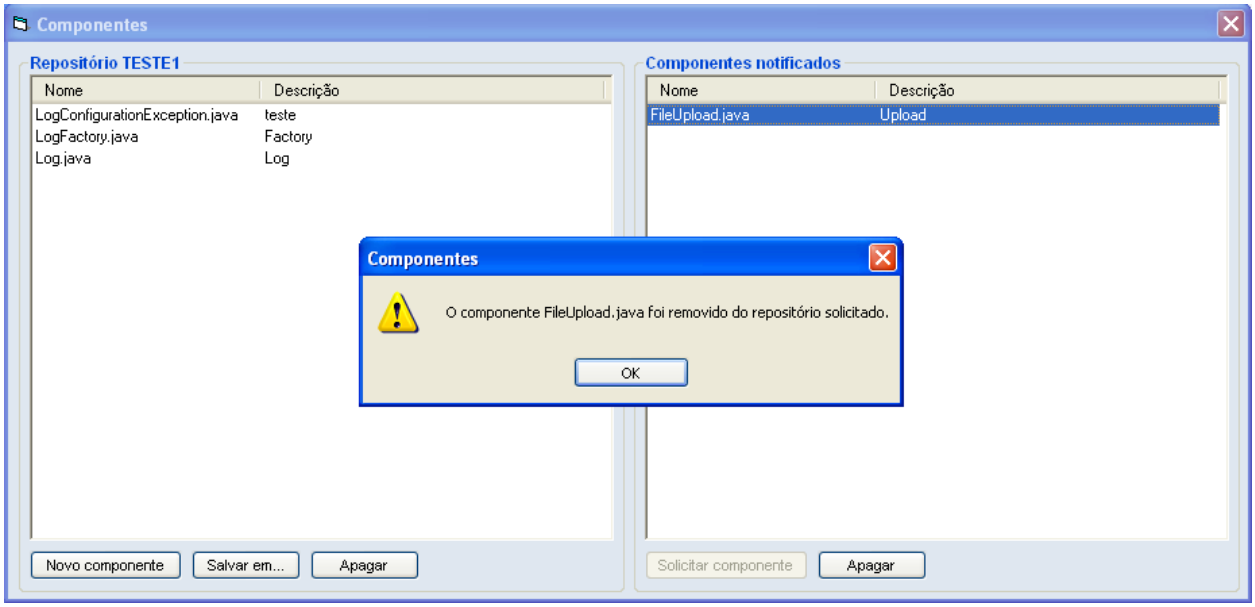

Figura 21: Mensagem de alerta sobre a remoção do componente solicitado.

O usuário também tem a possibilidade de apagar os componentes de seu repositório ou da lista de componentes notificados, clicando no botão "Apagar" referente a cada uma das situações. Após selecionar o componente que deseja remover e clicar no botão "Apagar", uma mensagem de confirmação da remoção do componente é exibida ao usuário, conforme mostra a figura 22:

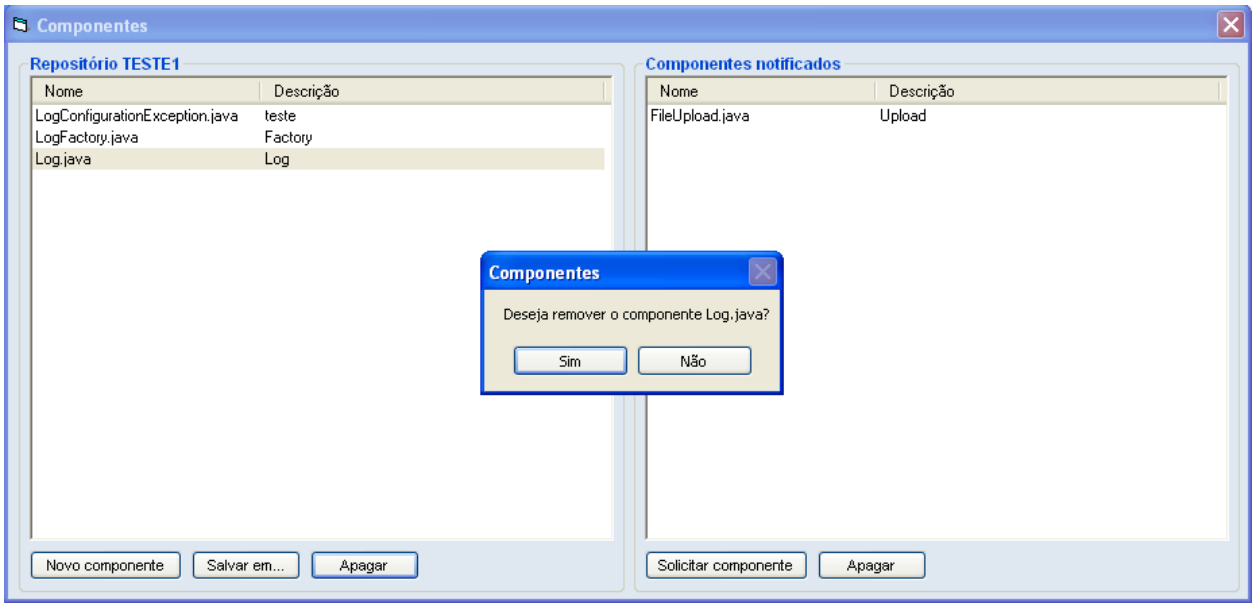

Figura 22: Mensagem de confirmação da remoção do componente.

Caso o usuário confirme a remoção, o componente é então removido do Repositório TESTE1, senão a remoção é cancelada. O usuário tem ainda a opção de salvar

o componente selecionado no local que desejar, clicando no botão “Salvar em...” e selecionando em seguida a pasta em que o componente será salvo, conforme ilustrado na figura 23:

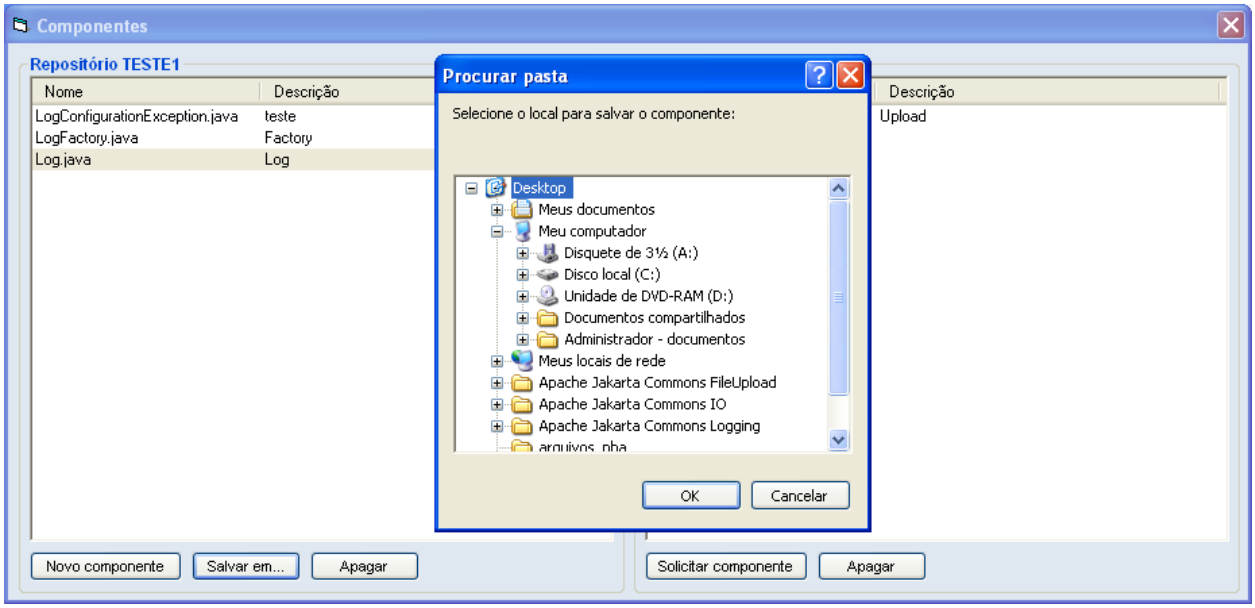

Figura 23: Escolha do local em que o componente será salvo.

Após clicar em “OK”, o componente selecionado é copiado do Repositório TESTE1 para o local escolhido pelo usuário.

4.4. Considerações Finais

Neste capítulo foram apresentadas as definições e as características de um repositório. Também foi destacado o objetivo da aplicação desenvolvida, e realizada uma apresentação das funcionalidades presentes na aplicação.

5. Implementação: Agentes para Repositório de Componentes

5.1. Introdução

A utilização de repositórios de componentes por uma organização permite que esta usufrua do reuso de *software*, o qual representa um ganho de produtividade e redução de custos de produção significativos para a empresa.

Para casos em que uma determinada empresa possua várias unidades distribuídas geograficamente, o que ocorre é que nem sempre os componentes e informações pertinentes à uma das unidades são interessantes para outra. Com isso, a proposta do sistema desenvolvido é que após a inserção de um novo componente em uma das unidades, as outras unidades recebam uma notificação daquele novo componente, podendo então decidir se irão adquirí-lo e mantê-lo em seu repositório local. Isso possibilita um gerenciamento do repositório de acordo com os interesses e necessidades de cada unidade.

Para efeito de implementação do sistema, foi considerada a existência de apenas duas unidades de uma organização, cada qual com seu repositório de componentes, como comentado no capítulo 4. Outra consideração a ser feita é que foram utilizados componentes *open source* do *Apache* (APACHE, 2010) durante os testes realizados com os repositórios. Um componente foi considerado como sendo qualquer arquivo ".java", ".xml" ou ".html" presentes nos componentes do *Apache*.

Os detalhes sobre o desenvolvimento do sistema são enunciados nos tópicos a seguir.

5.2. Implementação Computacional do Sistema

As seguintes tecnologias foram utilizadas para a implementação do sistema:

- Linguagem Java versão 6 (JAVA, 2010): linguagem de programação de propósito geral. Foi a linguagem utilizada para a implementação dos agentes do sistema;
- JADE versão 3.7 (*Java Agent DEvelopment Framework*) (JADE, 2010): *framework* que oferece um ambiente de desenvolvimento de aplicações baseadas em agentes de acordo com os padrões de interoperabilidade entre sistemas multi-agentes;

- Visual Basic 5.0: ambiente de desenvolvimento integrado totalmente gráfico, facilitando a construção da interface das aplicações. Foi utilizado para a construção da interface gráfica e de funcionalidades do repositório;
- VISDATA: aplicativo pertencente ao Visual Basic 5.0, que possibilita a criação e manipulação de bases de dados. As bases de dados *Access* utilizadas como repositório de componentes do sistema foram criadas com a utilização deste aplicativo.

5.3. Repositório de Componentes do Sistema

Para que testes do sistema multi-agentes desenvolvido fossem realizados, foi gerada uma base de dados *Access* que representa o repositório de componentes do sistema. A base de dados é constituída por duas tabelas, "Componentes" e "Notificacao".

A tabela "Componentes" mantém arquivados os componentes do repositório e é composta pelos seguintes campos:

- Comp_ln_Counter: campo do tipo long, que serve como um contador;
- Comp_tx_Nome: campo do tipo texto, que guarda o nome referente ao componente;
- Comp_me_Conteudo: campo do tipo memo. Mantém gravado o conteúdo do componente;
- Comp_dt_DataHora: campo do tipo date/time. Contém a data e hora da inserção do componente no repositório;
- Comp_tx_Hash: campo do tipo texto. Armazena o *hash* referente ao componente. O *hash* é gerado de acordo com o conteúdo do componente;
- Comp_tx_Descricao: campo do tipo texto. Armazena uma descrição do componente, adicionada pelo usuário.

A tabela "Notificacao" guarda a informação dos componentes que foram notificados, ou seja, componentes que foram inseridos no outro repositório. Ela é composta pelos mesmos campos da tabela "Componentes", exceto o campo que armazena o conteúdo do componente, o qual não existe na tabela.

5.4. Agentes do Sistema

5.4.1. Descrição dos Agentes

Os agentes pertencentes ao sistema e suas respectivas ações são enunciados a seguir:

- AgenteReceptor: agente responsável por receber notificação de novo componente inserido em outro repositório e por receber solicitação de componente que pertence ao repositório que se encontra na mesma máquina que o agente. Também é responsável por receber o componente propriamente dito. Quando a aplicação é encerrada ele recebe uma mensagem e é finalizado em seguida;
- AgenteNotificador: agente responsável pela notificação quando um novo componente é inserido no repositório pelo usuário. Após a notificação o agente é finalizado;
- AgenteRequest: agente responsável pela solicitação de um componente feita por um usuário. Após a solicitação o agente é finalizado;
- AgenteSender: agente responsável por enviar o componente solicitado pelo AgenteRequest. Após o envio do componente o agente é finalizado;
- AgenteExit: agente responsável por enviar uma mensagem ao AgenteReceptor, para causar a sua finalização. Após enviar a mensagem para o AgenteReceptor, o AgenteExit é finalizado.

Os agentes “AgenteNotificador”, “AgenteRequest”, “AgenteSender” e “AgenteExit” possuem as seguintes características:

- Autonomia: os agentes agem sem intervenção humana;
- Habilidade social: interagem com outros agentes e em alguns casos com seres humanos por algum tipo de linguagem de comunicação;

O agente “AgenteReceptor” possui as características destacadas anteriormente, além das características a seguir:

- Reatividade: fica em estado de espera do qual se desperta assim que percebe alguma solicitação de um agente para execução de algum serviço;

- Racionalidade: o agente possui um conjunto de regras e uma base de conhecimento para executar suas ações.

5.4.2. Métodos e Classes do JADE

A seguir são descritos métodos utilizados pelos agentes do sistema e que pertencem à classe *Agent* do JADE:

- *protected void setup* (): Esse método protegido é indicado para códigos inicializadores da aplicação. Os desenvolvedores podem sobrescrever esse método fornecendo comportamentos necessários ao agente. Quando este método é chamado, o agente já está registrado no AMS (*Agent Manager System*) e está apto a enviar e receber mensagens. Porém, o modelo de execução do agente ainda é sequencial e nenhum escalonamento de comportamento foi efetivado ainda. Logo, é essencial adicionar pelo menos um comportamento para o agente neste método, além de tarefas comuns de inicialização como registro no DF (*Directory Facilitator*), para torná-lo apto a fazer alguma coisa;
- *public void addBehaviour* (*Behaviour* comportamento): Esse método adiciona um novo comportamento ao agente. Ele será executado concorrentemente com outros comportamentos. Geralmente é usado no método *setup* () para disparar algum comportamento inicial, porém pode ser usado também para gerar comportamentos dinamicamente;
- *public final void send* (*ACLMessage* mensagem): Envia uma mensagem ACL para outro agente. Esse agente destino é especificado no campo receiver da mensagem no qual um ou mais agentes podem ser definidos como receptores;
- *public final ACLMessage receive* (): Recebe uma mensagem ACL da fila de mensagens do agente. Este método é não-bloqueante e retorna a primeira mensagem da fila caso haja alguma. Retorna *null* caso não haja mensagens;
- *public void doDelete* (): causa a destruição do agente. Esse método pode ser chamado pela plataforma de agentes ou pelo próprio agente. Chamar *doDelete* () em um agente já destruído não causa efeito nenhum.

JADE possui várias classes de comportamentos de agentes prontas para uso pelo desenvolvedor, adequando-as de acordo com a necessidade específica do agente. A seguir são descritas as classes referentes aos comportamentos utilizados pelos agentes do sistema:

- *Behaviour*: tem como finalidade prover a estrutura de comportamentos para os agentes JADE implementarem. O principal método da classe *Behaviour* é o *action* (). É nele que serão implementadas as tarefas ou ações que este comportamento irá tomar. Outro importante método é o *block* (), o qual permite a suspensão de apenas um comportamento. O comportamento é desbloqueado caso uma mensagem seja recebida.
- *CyclicBehaviour*: Comportamento atômico que deve ser executado sempre. Essa classe abstrata pode ser herdada para criação de comportamentos que se manterão executando continuamente;
- *TickerBehaviour*: Comportamento que executa periodicamente tarefas específicas. Ou seja, é uma classe abstrata que implementa um comportamento que executa periodicamente um pedaço de código definido pelo usuário em uma determinada freqüência de repetições. No caso, o desenvolvedor redefine o método *onTick* () e inclui o pedaço de código que deve ser executado periodicamente. O período de "*ticks*" é definido no construtor da classe em milisegundos.

Toda a troca de mensagens realizada no JADE é feita através de métodos próprios e com o uso de instâncias da classe *ACLMessage*. Esta classe possui um conjunto de atributos que estão em conformidade com as especificações da FIPA, implementando a linguagem FIPA-ACL. Assim, um agente que pretenda enviar uma mensagem deve instanciar um objeto da classe *ACLMessage*, preenchê-los com as informações necessárias e chamar o método *send* () da classe *Agent*. Caso for receber mensagens, o método *receive* () ou *blockingReceive* () da classe Agent deve ser chamado.

A classe *ACLMessage* implementa, como o próprio nome diz, uma mensagem na linguagem FIPA-ACL complacente às especificações da FIPA. Todos seus atributos podem ser acessados via métodos *set* e *get* (por exemplo, *getContent* e *setContent*). Além disso, a *ACLMessage* define um conjunto de constantes (ACCEPT_PROPOSAL, AGREE, CANCEL, CFP, CONFIRM, INFORM, entre outras) que são usadas para se referir às performativas da FIPA. Essas constantes são referidas no construtor da classe com o

objetivo de definir o tipo de mensagem. A seguir são descritos os métodos da classe *ACLMessage* utilizados:

- *public void addReceiver* (AID idAgente): Adiciona o AID de um agente como receptor ou destinatário da mensagem. Em outras palavras, determina quem receberá a mensagem;
- *public void setOntology* (*String* ontologia): Adiciona uma ontologia referente à mensagem;
- *public String getOntology* (): Extrai a ontologia referente à mensagem e adiciona em uma *string*;
- *public void setByteSequenceContent* (*byte*[] conteúdo): Adiciona uma sequência de *bytes* no conteúdo da mensagem;
- *public byte[] getByteSequenceContent* (): Extrai o conteúdo da mensagem e adiciona no *array byte[]*;
- *public void addUserDefinedParameter* (*String* chave, *String* valor): Adiciona um novo parâmetro definido pelo usuário;
- *public String getUserDefinedParameter* (*String* chave): Procura pelo parâmetro definido pelo usuário com a chave especificada. O método retorna *null* se o parâmetro não é encontrado;

Para selecionar as mensagens que um agente irá receber, foi utilizada a classe *jade.lang.acl.MessageTemplate*. Esta classe permite definir filtros para cada atributo da mensagem *ACLMessage* e estes filtros podem ser utilizados como parâmetros do método *receive* (). Nesta classe se define um conjunto de métodos estáticos que retornam como resultado um objeto do tipo *MessageTemplate*. As opções para filtragem utilizadas foram:

- *MatchOntology* (*String*): permite que mensagens com uma determinada ontologia sejam lidas;
- *Or* (*MessageTemplate1, MessageTemplate2*): realiza um "OU" lógico entre dois filtros.

5.4.3. Bibliotecas de Pacotes do JADE

JADE oferece inúmeros pacotes de classes que visam auxiliar tanto na implementação quanto na monitoração de sistemas multiagentes. A seguir são descritos os pacotes de classe utilizados para o desenvolvimento do sistema:

- *jade.core*: Esse pacote implementa o "*kernel*" do sistema. Ou seja, é o "coração" do JADE que inclui classes importantes como a classe *Agent* e AID. Ele fornece uma passagem de mensagens simples com protocolo de mensagens variados e um ambiente de execução *multithreading* para softwares de agentes;
- *jade.core.behaviours*: Esse subpacote do pacote *jade.core* contém a classe *Behaviour*. Essa classe, que próprio nome já diz, implementa os comportamentos básicos de um agente. Ou seja, suas tarefas, intenções ou objetivos;
- *jade.lang.acl*: Neste pacote contém todo o suporte necessário para a implementação de uma mensagem no formato FIPA-ACL , incluindo a classe *ACLMessage*, o analisador (*parser*), o codificador (*encoder*), e uma classe para representar *templates* (padrões) de mensagens ACL;
- *jade.domain*: Este pacote e seus sub-pacotes contêm todas as classes Java que representam as entidades gerenciadoras de agentes (*Agent Management*) definidas pela FIPA, como por exemplo o *AMS Agent*, que gerencia o ciclo de vida dos agentes no ambiente e guarda informações sobre eles e o *DF Agent*, que mantêm as páginas amarelas de informações sobre os agentes.

5.5. Modelagem do Sistema

A modelagem do sistema foi realizada com a utilização de diagramas da linguagem de modelagem de sistemas multi-agentes, o ANote. Os diagramas são apresentados a seguir.

5.5.1. Diagrama de Objetivos

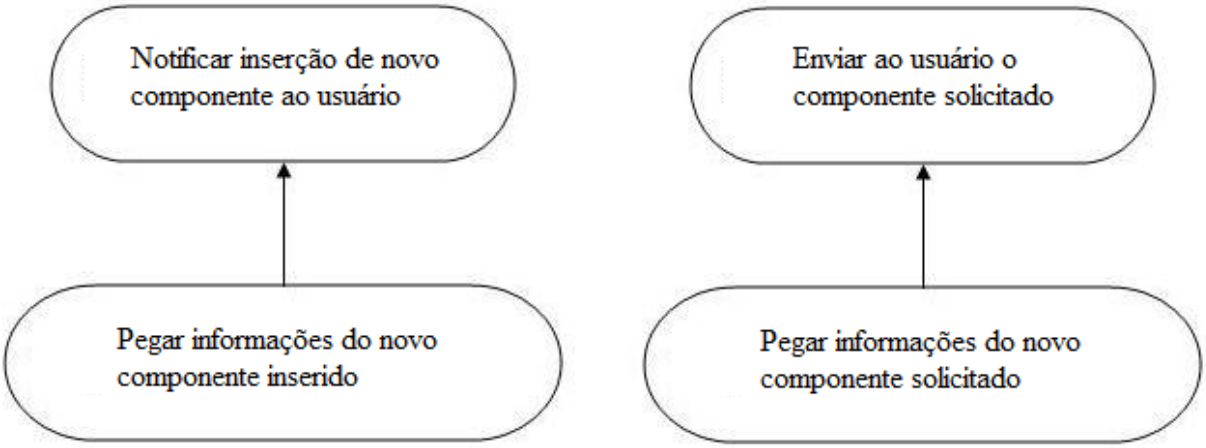

Figura 24: Diagrama de Objetivos do Sistema.

Através deste diagrama podemos visualizar os objetivos principais do sistema, que são "Notificar inserção de novo componente ao usuário" e "Enviar ao usuário o componente solicitado".

5.5.2. Diagrama de Agentes

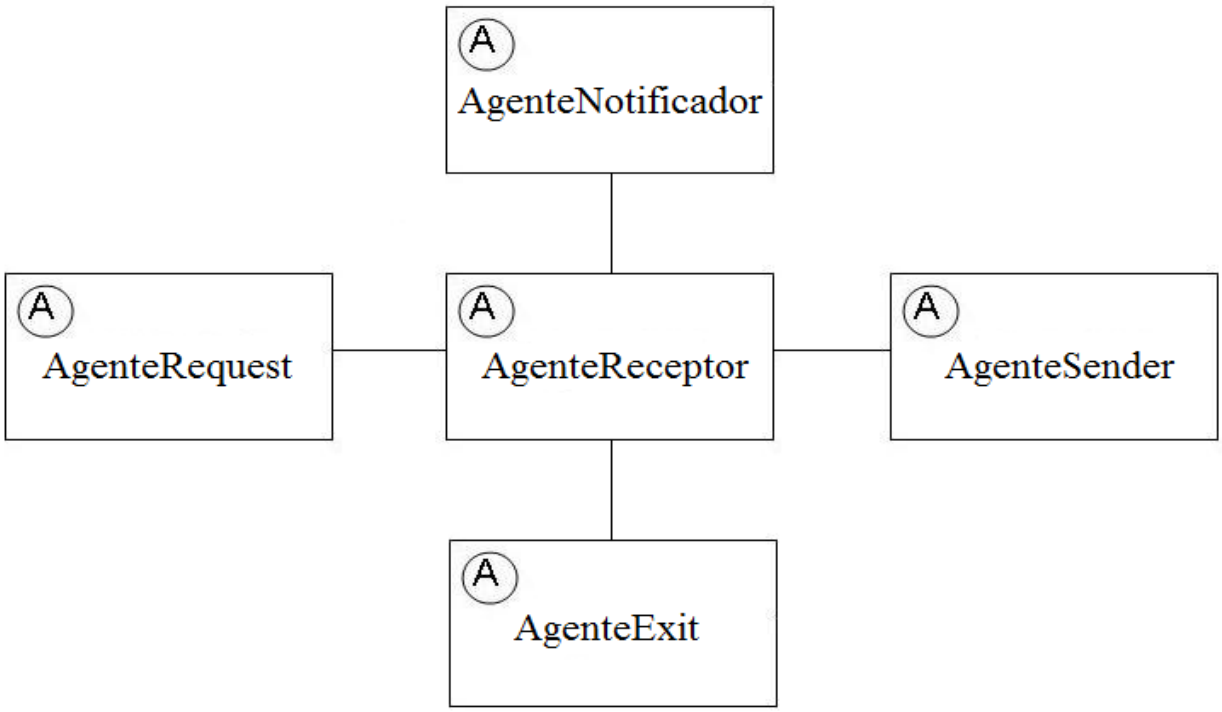

Figura 25: Diagrama de Agentes do Sistema.

Este diagrama nos indica quais são os agentes pertencentes ao sistema e as relações existentes entre eles. Pode-se notar que o "AgenteReceptor" se relaciona com todos os demais agentes.

5.5.3. Diagrama de Organização

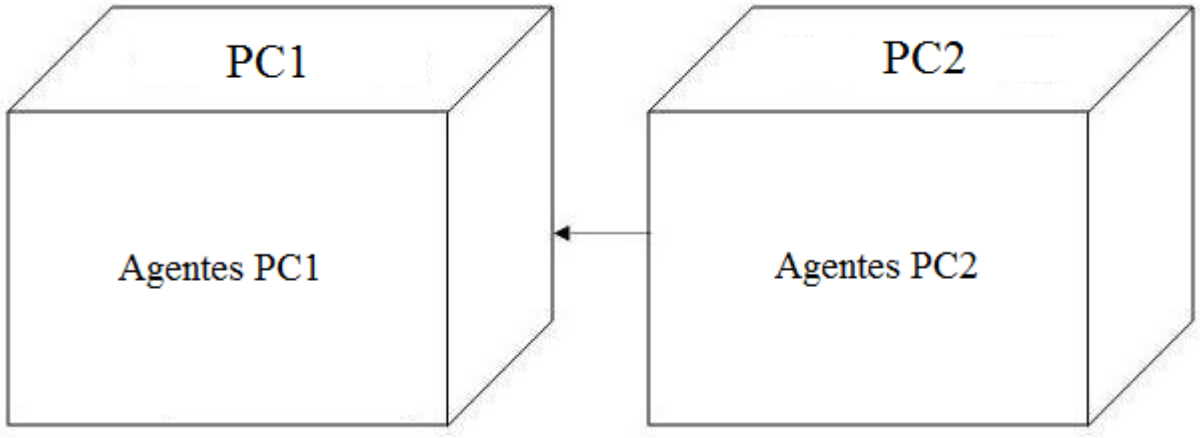

Figura 26: Diagrama de Organização do Sistema.

Neste diagrama podemos notar duas organizações de agentes que são interligadas. Cada uma das organizações é constituída pelos agentes apresentados na figura 25. A única diferença entre os agentes presentes em cada uma das organizações é quanto à ontologia das mensagens enviadas e recebidas pelos agentes. Considera-se como ontologia no sistema desenvolvido os termos que cada um dos agentes reconhece ou envia para outros agentes. Estes termos são inseridos dentro de cada mensagem enviada pelos agentes.

Por exemplo, no PC1, o "AgenteReceptor" só recebe mensagens com as seguintes ontologias: "NovoPC2", "RequestPC2" e "ComponentePC2", que pertencem a mensagens vindas de agentes presentes no PC2. Ele também aceita a ontologia "ExitPC1", que pertence a uma mensagem vinda de um agente que se localiza no próprio PC1. Já o "AgenteReceptor" do PC2 aceita as ontologias "NovoPC1", "RequestPC1", "ComponentePC1" e "ExitPC2".

5.5.4. Diagramas de Cenário

Considerando que a análise dos cenários ocorre no PC1 (de acordo com a figura 26), temos os seguintes diagramas de cenário no sistema:

Receber notificação	
Lead Agent	AgenteReceptor
Precondition	A ontologia da mensagem deve ser igual a "NovoPC2"
Main Action Plan	Grava a mensagem de notificação em um arquivo texto na pasta "Notificado" do sistema
Interactions	AgenteNotificador
Variant Plan	

Figura 27: Diagrama do Cenário "Receber notificação".

Receber solicitação	
Lead Agent	AgenteReceptor
Precondition	A ontologia da mensagem deve ser igual a "RequestPC2"
Main Action Plan	Grava a mensagem de solicitação em um arquivo texto na pasta "RequestIn" do sistema
Interactions	AgenteRequest
Variant Plan	

Figura 28: Diagrama do Cenário "Receber solicitação".

Receber componente	
Lead Agent	AgenteReceptor
Precondition	A ontologia da mensagem deve ser igual a "ComponentePC2"
Main Action Plan	Grava o componente na pasta "Recebido" do sistema
Interactions	AgenteSender
Variant Plan	

Figura 29: Diagrama do Cenário "Receber componente".

Finalizar execução	
Lead Agent	AgenteReceptor
Precondition	A ontologia da mensagem deve ser igual a "ExitPC1"
Main Action Plan	Finaliza sua execução
Interactions	AgenteExit
Variant Plan	

Figura 30: Diagrama do Cenário "Finalizar execução".

Enviar notificação	
Lead Agent	AgenteNotificador
Precondition	O usuário deve inserir um novo componente no repositório
Main Action Plan	Pega o conteúdo do arquivo texto presente na pasta "Novo" do sistema e em seguida envia para o AgenteReceptor. O conteúdo é o hash, o nome e a descrição do componente inserido. A ontologia da mensagem será "NovoPC1"
Interactions	AgenteReceptor
Variant Plan	

Figura 31: Diagrama do Cenário "Enviar notificação".

Enviar solicitação	
Lead Agent	AgenteRequest
Precondition	O usuário deve solicitar um componente que foi inserido no outro repositório
Main Action Plan	Pega o conteúdo do arquivo texto presente na pasta "RequestOut" do sistema e em seguida envia para o AgenteReceptor. O conteúdo é o hash do componente solicitado. A ontologia da mensagem será "RequestPC1"
Interactions	AgenteReceptor
Variant Plan	

Figura 32: Diagrama do Cenário "Enviar solicitação".

Enviar componente	
Lead Agent	AgenteSender
Precondition	É acionado sempre que um novo arquivo de solicitação é inserido na pasta "RequestIn" do sistema
Main Action Plan	Pega o componente solicitado, que se encontra na pasta "Enviado" do sistema e em seguida envia para o AgenteReceptor. A ontologia da mensagem será "ComponentePC1"
Interactions	AgenteReceptor
Variant Plan	

Figura 33: Diagrama do Cenário "Enviar componente".

Alertar finalização da execução	
Lead Agent	AgenteExit
Precondition	É acionado sempre que o usuário finaliza a aplicação
Main Action Plan	Envia mensagem ao AgenteReceptor com a ontologia "ExitPC1"
Interactions	AgenteReceptor
Variant Plan	

Figura 34: Diagrama do Cenário "Alertar finalização da execução".

5.5.5. Diagrama de Comunicação

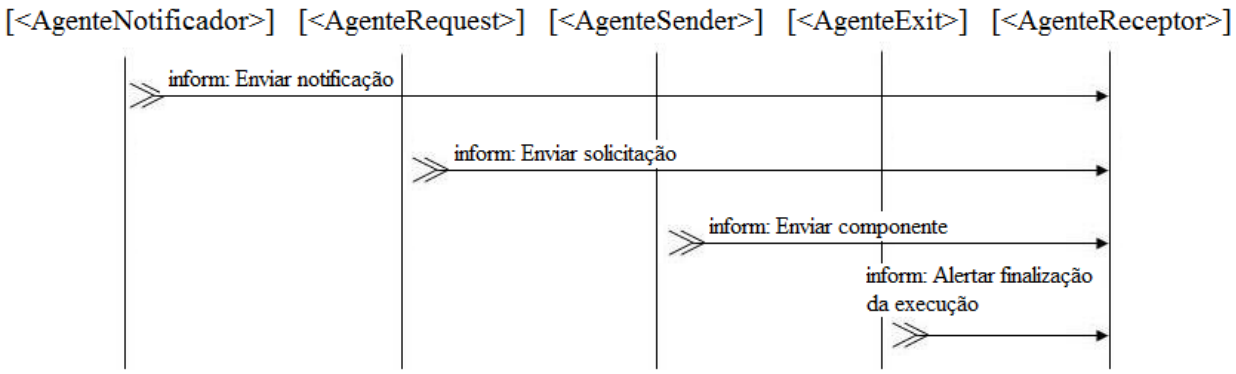

Figura 35: Diagrama de Comunicação entre Agentes do Sistema.

Este diagrama evidencia qual é o tipo da mensagem enviada e para qual agente a mensagem é enviada.

5.6. Visualização da Interação entre os Agentes

Para visualizar a interação entre os agentes foi utilizada a ferramenta *SnifferAgent* do JADE. Trata-se de uma ferramenta que mostra a troca de mensagens entre agentes de forma similar a um diagrama de sequências UML. Toda mensagem enviada de um agente para outro é rastreada e disponibilizada em uma interface gráfica para o usuário.

Considerando que os testes foram realizados numa máquina com o nome “sqlbi”, o resultado obtido na figura 36 é referente à inserção de um novo componente em um repositório que está em uma máquina diferente, a qual será denotada como “teste1”. Agentes em vermelho são os agentes ainda ativos, e agentes em amarelo são os agente inativos.

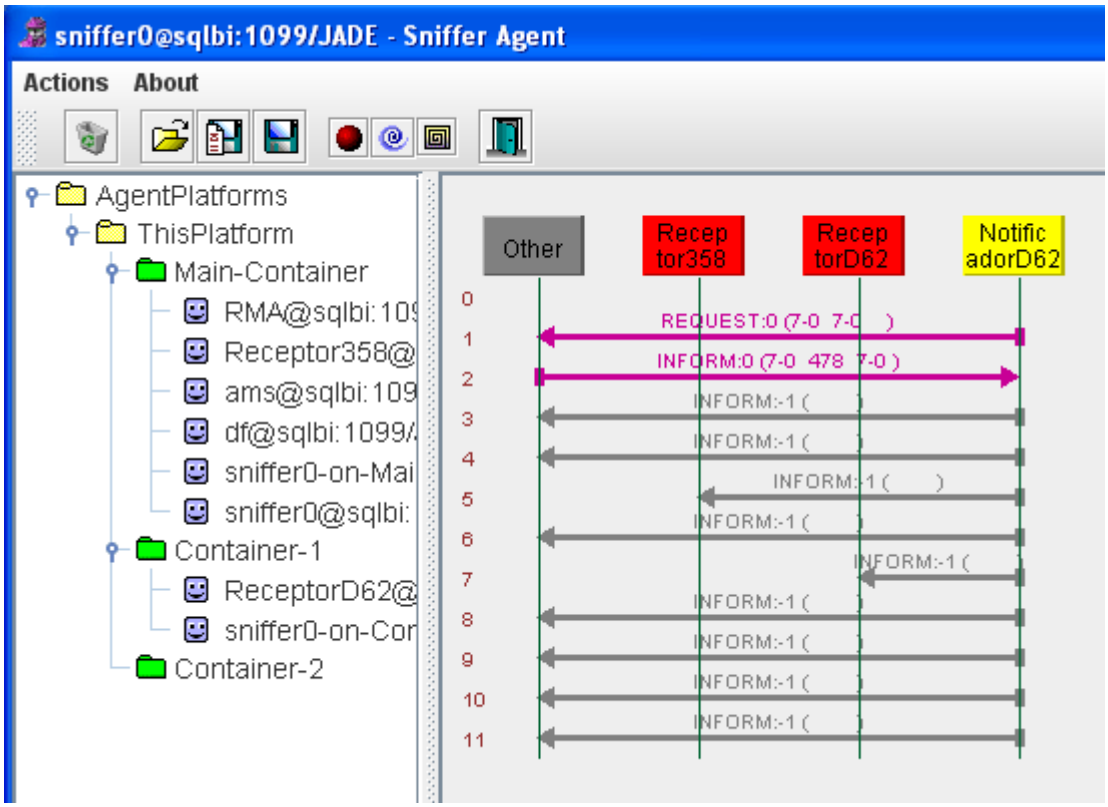

Figura 36: *Sniffer Agent* após inserção de novo componente.

Analisando o resultado do diagrama gerado pelo *Sniffer Agent*, nota-se que o agente "NotificadorD62" se registrou no sistema logo após sua inclusão (mensagem REQUEST). Em seguida ele recebe uma resposta (INFORM), confirmando seu registro. A seguir ele executa sua ação e envia para os agentes a mensagem de notificação de novo componente. Porém apenas o agente "Receptor358", que está na máquina "sqlbi", irá aceitar a mensagem devido à ontologia. Em seguida o agente "NotificadorD62" é finalizado.

A figura 37 exemplifica o diagrama gerado pelo *Sniffer Agent* após o usuário da máquina "sqlbi" solicitar um componente que foi adicionado na máquina "teste1".

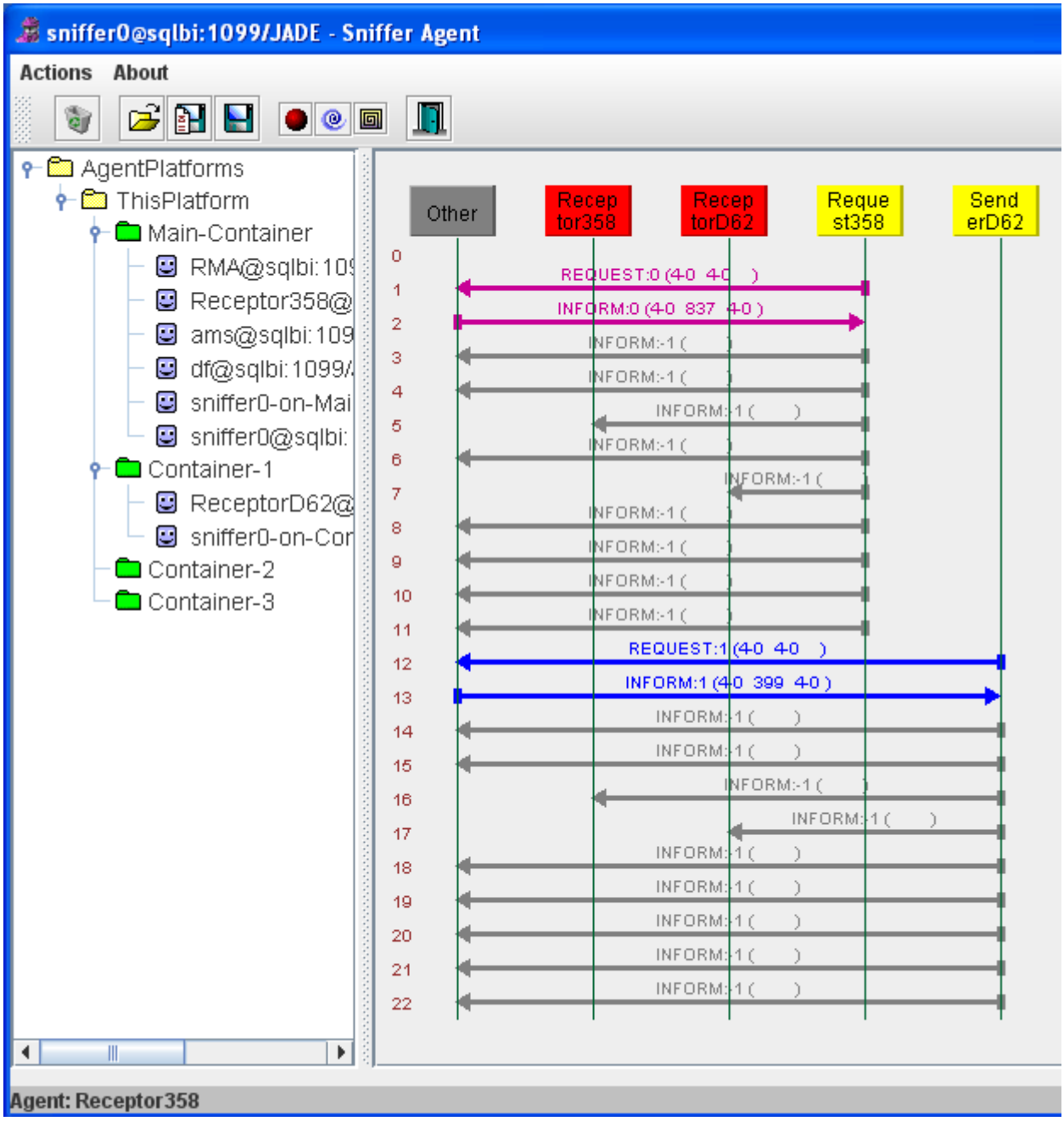

Figura 37: *Sniffer Agent* após solicitação de componente.

O agente “Request358” envia mensagem de solicitação aos demais agentes, porém somente o agente “ReceptorD62” (que está na máquina “teste1”), irá aceitar a mensagem. Após isso este agente é finalizado.

Em seguida, o agente “SenderD62” (que se localiza na máquina “teste1”) irá enviar o componente solicitado, e somente o agente “Receptor358” irá receber o componente. Após o envio, o agente “SenderD62” é finalizado.

5.7. Considerações Finais

Neste capítulo foram apresentados detalhes sobre a implementação do sistema proposto, como o repositório de componentes do sistema, a descrição dos agentes desenvolvidos e suas ações, além da modelagem do sistema e uma visualização da interação entre os agentes, através do *Sniffer Agent*.

6. Considerações Finais

Neste trabalho foi apresentada uma aplicação desenvolvida com a proposta de apoiar o gerenciamento de repositórios de componentes não-centralizados, utilizando uma solução orientada a agentes. Para o desenvolvimento desta aplicação, foram abordados os conceitos relacionados a repositórios de componentes, agentes de software, linguagens de modelagem de agentes e *frameworks* de desenvolvimento de sistemas multi-agentes.

A partir do embasamento teórico foi realizada a modelagem dos agentes de software e foram especificados detalhes da implementação. Há também uma demonstração do funcionamento da aplicação e a descrição da interação entre os agentes do sistema.

A aplicação possibilita que o usuário de um determinado repositório que está interligado a outro, sendo que ambos são pertencentes a uma mesma organização, possa decidir quais os componentes que ele deseja manter, de acordo com o seu interesse. Isso evita que componentes irrelevantes ao usuário sejam mantidos no repositório. Este contexto favorece a utilização de técnicas de reuso de software, uma vez que existe um maior apoio ao gerenciamento dos repositórios.

A abordagem orientada a agentes, utilizada para desenvolver a aplicação, permitiu um gerenciamento dos repositórios de componentes sem que fosse necessária a intervenção do usuário. A autonomia nas ações e a capacidade de comunicação entre os agentes são fatores que favoreceram sua utilização neste domínio.

Trabalhos futuros podem considerar a utilização de mais do que dois repositórios não-centralizados, além da realização de busca de componentes entre todos eles. Essa busca poderia ser resolvida através da troca de mensagens entre os agentes do sistema, que comunicariam entre si sobre a busca em questão, e pesquisariam no repositório a que pertencem. Outra possibilidade de trabalho seria a expansão do sistema para repositórios de qualquer tipo de artefato produzido durante um projeto de software. Como por exemplo, poderiam ser utilizados repositórios de componentes para manter qualquer tipo de componente, separados por conjunto ou característica, como interface, métodos numéricos, entre outros.

Referências Bibliográficas

ALMEIDA, A. C. BIOANOT: UM SISTEMA MULTI-AGENTES PARA NOTIFICAÇÃO DE (RE)ANOTAÇÕES DE SEQÜÊNCIAS EM BANCOS DE DADOS GENÔMICOS, 2006. Dissertação de mestrado - IME.

APACHE. 2010. "Apache Commons". Disponível em: http://commons.apache.org. Acesso em: junho 2010.

AUML. 2010. "Agent UML". Disponível em: http://www.auml.org. Acesso em: maio 2010.

AYRES, L. Estudo e Desenvolvimento de Sistemas Multiagentes usando JADE: Java Agent Development framework, 2003. Trabalho de conclusão para obtenção do grau de Bacharel - UNIFOR.

BAKER, A. JAFMAS - A java-based agent framework for multiagent systems. Development and Implementation. Cincinnati: Department of Electrical & Computer Engineering and Computer Science University of Cincinnati, 1997. Doctoral thesis.

BERGENTI, F.; SHEHORY, O.; ZAMBONELLI, F. Agent-Oriented Software Engineering. European Agent Systems Summer School (EASSS 2002).

CAFARATE, L. Utilização da engenharia de software orientada a agentes na modelagem de um sistema de seleção de pessoas, 2008. Trabalho de conclusão para obtenção do grau de Bacharel - UFSM.

CARVALHO, R. W. Um ambiente de suporte para uma linguagem de modelagem de sistemas multi-agentes, 2005. Dissertação de mestrado - PUC-Rio.

CHOREN, R.; LUCENA, C. Modeling Multi-agent Systems with ANote. Journal on Software and Systems Modeling (SoSyM) 4(2), 2005,p.199-208.

COUGAAR. 2010. "Cougaar Agent Architecture". Disponível em: http://www.cougaar.org. Acesso em: maio 2010.

EZRAN, MICHEL, ET AL. Practical Software Reuse: The Essencial guide. 1999.185 p.

FIPA. 2010. "The Foundation for Intelligent Physical Agents". Disponivel em: http://www.fipa.org. Acesso em: maio 2010.

GIRARDI, R. Engenharia de Software baseada em Agentes. Itajai, Santa Catarina, Brasil: Anais do IV Congresso Brasileiro de Ciência da Computação (CBCOMP 2004). Ed. Univali, pp. 913-937, 2004.

HENNINGER, SCOTT. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. In: :ACM Transactions on Software Engineering and Methodology No 2, abril 1997. v. 6, p.111-140.

JADE. 2010. "Java Agent DEvelopment Framework". Disponível em: http://jade.tilab.com. Acesso em: maio 2010.

JAVA. 2010."Developer Resources for Java Technology". Disponível em: http://java.sun.com/. Acesso em: junho 2010.

JENNINGS, N. R. Agent Software. Proceedings UNICOM Seminar on Agent Software, Londres, UK, p. 12-27, 1995.

JENNINGS, N. R. On agent-based software engineering. Artificial Intelligence 117, 2000, p. 277-296.

JENNINGS, N. R.; WOOLDRIDGE, M. Applications of Intelligent Agents in Agent Technology: Foundations, Applications, and Markets (eds. N. R. Jennings and M. Wooldridge. London, 1998. Disponível em: http://agents.umbc.edu/introduction/jennings98.pdf. Acesso em: maio 2010.

LUCENA, C.; GARCIA, A.; ROMANOVSKY, A.; CASTRO, J.; ALENCAR, P. Software Engineering for Multi-Agent Systems II: Research Issues and Practical Applications. Lecture Notes in Computer Science - LNCS Vol. 2940, State-of-the-Art Survey. 278 pages. 2004.

NWANA, H. S. Software Agents: An Overview. Knowledge Engineering Review Press, vol. 11, nº 3, p. 1-40, 1996.

ODELL, J. Agent Technology - Green Paper in Agent Working Group OMG Document. Needham, 2000. Disponível em: http://citeseer.ist.psu.edu/cache/papers/cs/16218/http:zSzzSzjamesodell.comzSzec2000-08-01.pdf/group00agent.pdf. Acesso em: maio 2010.

ODELL, J. Representing Agent Interaction Protocols in UML in AOSE. Limerick, 1999. Disponível em: http://citeseer.ist.psu.edu/cache/papers/cs/32308/http:zSzzSzwww.ca.sandia.govzSzFIPAPDMzSzodell.pdf/odell99representing.pdf. Acesso em: maio 2010.

PEROTTO, M. Estudo de uma metodologia orientada a agentes: um protótipo para um ambiente virtual, 2002. Dissertação de mestrado - UFSC.

QUINTÃO, H. C. Especificação de um sistema multiagente de recomendação de ações em caso de falhas de sistemas de automação e controle industriais, 2008. Dissertação de mestrado - UFMA.

RUSSEL, S.; NORVIG, P. Artificial Intelligence: A Modern Approach. s.l.: Prentice Hall, 1995.

SANTOS, G. Introduzindo Variabilidade no Desenvolvimento de Sistemas Multi-Agentes, 2007. Dissertação de mestrado - PUC-Rio.

SILVA, J. Desenvolvimento de um Framework para Objetos Inteligentes de Aprendizagem Aderente a um Modelo de Referência para Construção de Conteúdos de Aprendizagem, 2007. Dissertação de mestrado - UFSC.

SOSA, I. Sistemas Multiagentes para Controle Inteligente da Caldeira de Recuperação, 2007. Dissertação de mestrado - Escola Politécnica da Universidade de São Paulo.

WOOLDRIDGE, M. Intelligent Agents In G. Weiss, editor: Multiagent Systems, The MIT Press, April, 1999.

XML. 2010. "Extensible Markup Language". Disponível em: http://www.w3.org/XML. Acesso em: maio 2010.