Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Bacharelado em Ciências da Computação

METODOLOGIA PARA ANÁLISE E PROJETO DE SISTEMAS MULTI-AGENTES.

por
Abel Augusto de Carvalho Ferreira

Florianópolis, 2008

Abel Augusto de Carvalho Ferreira

METODOLOGIA PARA ANÁLISE E PROJETO DE SISTEMAS MULTI-AGENTES.

Monografia apresentada como requisito para obtenção do grau de Bacharel em Ciência de Computação pela Universidade Federal de Santa Catarina

Área de Concentração: Engenharia de Software e Sistemas Multi-Agentes.

Orientador:

Prof. Ricardo Azambuja Silveira, Dr.

Florianópolis, 2008

METODOLOGIA PARA ANÁLISE E PROJETO DE SISTEMAS MULTI-AGENTES.

por

Abel Augusto de Carvalho Ferreira

Monografia apresentada como requisito para obtenção do grau de Bacharel em Ciência de Computação pela Universidade Federal de Santa Catarina

Área de Concentração: Engenharia de Software e Sistemas Multi-Agentes.

Data de aprovação: _____

_____

Prof. Dr. Ricardo Azambuja Silveira

Universidade Federal de Santa Catarina

_____

Prof. Dr. Mauro Roisenberg

Universidade Federal de Santa Catarina

_____

Prof. Dr. Jovelino Falqueto

Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço a todos os excelentes professores que fizeram parte da minha formação, aos meus pais e familiares que mesmo longe sempre me apoiaram, e a todas as pessoas que trabalham para o bem comum.

"Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves."

Yukihiro "Matz" Matsumoto

RESUMO

O trabalho apresenta algumas das metodologias de engenharia de software baseadas em agentes e compara-as com a Engenharia de Software Orientada a Objeto. Propõe o desenvolvimento de um sistema de organização de torneios de jogos em uma aplicação de realidade virtual ou mesmo em um ambiente real. Esse projeto é utilizado como estudo de caso para aplicação de uma metodologia de Sistema Orientado a Agente. A abordagem de desenvolvimento de sistemas Multi-Agentes nos fornece um suporte maior à abstração de dados. O trabalho tem foco no desenvolvimento da modelagem do sistema, seguindo uma metodologia sugerida, explicando-a na prática.

ABSTRACT

This project presents some of the methodologies of software engineering agent-based and comparing them to Object Oriented Software Engineering. This paper proposes the development of the organizational system of computer game tournaments in a virtual reality application or actually in a real environment. This project is used as a case study to apply the methodology of Agent-Oriented System. Multi-Agent systems approach to provide us with more possibilities to abstract data. This project focuses on the development of system modeling, following a suggested methodology thereby explaining it and putting into practice.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Engenharia de software baseada em Agentes é um campo relativamente novo na Ciência da Computação significando um avanço na Engenharia de Sofware além das metodologias existentes. A proposta deste trabalho de conclusão de curso é realizar um estudo investigativo nas metodologias de análise e projeto de sistemas multi-agentes, comparando as metodologias existentes e sua abordagem diferenciada em relação à análise e projeto de sistemas orientados a objetos.

Ambientes de aplicações reais costumam ser compostos de um grande número de subproblemas que, por sua vez, podem ainda ser divididos em subproblemas, a fim de facilitar a análise e a implementação de soluções. Sistemas Multi agente (SMA) é uma estratégia da Inteligência Artificial Distribuída (IAD) que se mostra adequada para manipular problemas complexos. Este enfoque baseia-se no estudo e desenvolvimento de agentes autônomos em um universo multi agente, isto é, onde múltiplos agentes convivem em um ambiente comum operando cooperativamente na resolução de problemas.

Para os SMA, o termo autônomo designa o fato de que os agentes têm uma existência própria, independente da existência de outros agentes. Usualmente, cada agente possui um conjunto de capacidades comportamentais que definem sua competência, um conjunto de objetivos, e a autonomia necessária para utilizar suas capacidades comportamentais, a fim de alcançar seus objetivos. Um agente é uma entidade computacional com um comportamento autônomo que lhe permite decidir suas próprias ações. Podemos dizer que estes agentes são heterogêneos em sua natureza quando a arquitetura utilizada na implementação de cada agente for diferente.

A motivação para esse trabalho vem do interesse crescente em Engenharia de Software, assim como em agentes inteligentes. As experiências obtidas nos últimos estágios, realizados como disciplina optativa do curso, em empresas de desenvolvimento de software fez aumentar o reconhecimento que a falta da engenharia de software acarreta deficiências no processo do desenvolvimento e, por conseguinte, na qualidade final do software, independente da área do desenvolvimento.

O enfoque desse trabalho é apresentar um estudo sobre a metodologia de desenvolvimento de sistemas multi agentes para facilitar o projeto deste tipo de arquitetura de sistemas.

É necessário deixar claro que a computação trabalha a serviço do usuário. Os sistemas computacionais vieram para facilitar a vida das pessoas e não complicá-las, como acontece em muitos casos. O desenvolvedor de software também é um usuário do computador, e facilitar a vida dele é importante. Um desenvolvimento que não segue metodologias de processo carece de qualidade para manutenção e até mesmo em todo ciclo de vida do desenvolvimento, atrasando e depreciando a produtividade do desenvolvedor, a flexibilidade, a reusabilidade, e o aumento da demanda de homens/hora em um projeto.

A modelagem é uma parte fundamental do projeto, pois com ela fica mais fácil obter desde uma visão geral de um sistema, mais abstrata, até a visão detalhada, menos abstrata, sem ter que apelar para um processo de engenharia reversa. O simples ato de ter que ler um código implementado por outra pessoa sem que haja uma documentação adequada que associe o código à modelagem do sistema, já define a necessidade de engenharia reversa para sua manutenção. Isto se refere tanto ao código escrito por outra pessoa, mas também um próprio código antigo que esteja sendo utilizado e não possua uma documentação adequada mostrando a modelagem do sistema para entendimento imediato do código, acarretando um esforço desnecessário.

O esforço do projeto pode ser amenizado pela utilização de ferramentas CASE (*Computer-Aided Software Engineering*), frameworks, e outras ferramentas de apoio ao desenvolvimento. É apresentado também no trabalho o framework JADE (Java Agent Development), que é utilizado para facilitar o desenvolvimento de agentes baseados no modelo FIPA utilizando tecnologia da linguagem Java.

Com o apoio de todas essas ferramentas, o desenvolvimento de sistemas multi-agente torna-se mais produtivos e fácil de se projetar.

Com a finalidade de apresentar um estudo de caso da aplicação de uma metodologia de Engenharia de Software orientada a agentes é apresentado o projeto "Organizador de partidas de games", OPG, um sistema orientado a agentes que será adicionado dentro de uma aplicação de jogo MMORPG.

MMORPG é um jogo de computador que permite a milhares de jogadores criarem personagens em um mundo virtual dinâmico ao mesmo tempo na Internet. Cada um desses milhares de jogadores estará apto de utilizar dos serviços do OPG enquanto estiverem imersos no jogo.

1.1 Objetivos

Estudar diversas metodologias de desenvolvimento de sistemas multi-agentes, aprofundar o conhecimento em metodologias de cunho geral e aprender a desenvolver um trabalho científico.

1.1.2 Objetivos Específicos

O desenvolvimento de um aplicativo para diversos ambientes que vêem a quantidade de usuários/jogadores crescer, dificultando o gerencialmente entre esses usuários e a aplicabilidade de ferramentas de comunicação; como estudo de caso para exemplificar a utilização de uma metodologia diferente baseada nas existentes sem o intuito de reinventar a roda.

1.2 Contribuições Esperadas

A monografia tem o intuito de gerar informações para que outras pessoas interessadas na área de atuação desse trabalho possam ter como referência os tipos de metodologias existentes e sobre as tecnologias abordadas.

1.3 Organização do trabalho

No capítulo 2 será abordado o embasamento teórico sobre agentes, arquitetura de agentes e programação orientada a agentes.

No capítulo 3 será mostrado algumas das diversas metodologias de sistemas Multi-Agentes existentes.

No Capítulo 4 iremos abordar a metodologia e as tecnologias utilizadas no trabalho.

No Capítulo 5 será introduzido o estudo de caso, Organizador de Partidas de Games, e iniciado sua análise e projeto.

2 AGENTES E OBJETOS

2.1 Agentes e Sistemas Multi-Agentes

O termo *Agente* tem sido amplamente usado em vários segmentos: na inteligência artificial, banco de dados, sistemas operacionais e redes de computadores. Por isso, não há uma única definição para agentes, porém todas as definições concordam que um agente é essencialmente um componente de software especial que tem autonomia, que fornece uma interface de interoperabilidade para um sistema qualquer e possui comportamentos como um agente humano.

O termo possui diversos significados para diferentes pessoas. Entretanto, num olhar mais focado, pode-se dizer que dois empregos do termo agente podem ser identificados: A noção fraca e a noção forte de agente (Wooldridge e Jennings, 1995). A noção fraca é a que os pesquisadores mais concordam, enquanto a noção forte é mais controversa e um assunto para atividade de pesquisa.

A noção fraca denota um sistema de computador baseado em software com as seguintes propriedades (Wooldridge e Jennings, 1995):

- Autônomo: agentes que operam sem intervenção direta de humanos ou outros, e que tem algum controle de suas ações e estados internos.
- Habilidade social: agentes interagem com outros agentes (e possivelmente humanos) via algum tipo de linguagem de comunicação.
- Reativo: respondem a estímulos em tempo que uma mudança ocorra.
- Pró-atividade: além de agir em resposta ao ambiente, agentes são capazes de exibir comportamento direcionado a resultados tomando iniciativas.

A noção forte é uma extensão da notação fraca, e adiciona propriedades humanas e mental tais como crença, desejo e intenções (Shoham, 1993).

Consistente com a noção fraca, agentes de software são aplicações que se comunicam em uma linguagem de comunicação expressiva (Genesereth at al, 1994). Embora pareça simplista essa definição, ela nos permite claramente identificar o que constitui um sistema multi agente Agentes são peças de códigos autônomos, capazes de comunicar entre si usando uma linguagem de comunicação de agentes.

Assumimos, então, nesse trabalho a seguinte definição de agentes:

Agentes residem em uma plataforma que, consistente com a visão apresentada, fornece um agente com um mecanismo peculiar para comunicar por nomes, não considerando a complexidade e natureza do ambiente subjacente.

Quando o sistema possui mais de um agente, consiste então de um sistema multi agente Esses sistemas (SMA) podem tratar sistemas complexos e introduzir possibilidades de os agentes terem objetivos comuns ou conflituosos. Esses agentes podem se comunicar direta ou indiretamente, atuando no ambiente comum ou via mecanismos explícitos de comunicação e negociação, seja pela cooperação ou pela competição de seus interesses.

Assim, um agente é *autônomo*, por que opera sem intervenção humana ou qualquer outra e tem controle de suas ações e estados internos. Um agente é social, porque interage com outros agentes ou com humanos para chegar ao seu objetivo, e é reativo, pois tem percepção do ambiente e responde pelas mudanças lá ocorridas.

Um Agente é pró-ativo, pois ele não age em resposta ao seu ambiente somente, mas também é capaz de exibir comportamento direcionado a objetivos tomando iniciativas. Além disso, se necessário, um agente pode ter a *habilidade de se mover* entre diferentes nós de uma rede de computadores. Pode ser *honesto,* tendo certeza que nunca enviará falsas informações. Pode ser benevolente, sempre tentando fazer o que lhe foi pedido. Pode ser racional, sempre agindo em direção ao seu objetivo, e pode ser bem informado, adaptando-se a mudança no ambiente e aos desejos dos usuários.

2.2 Arquitetura dos Agentes

Agentes de software possuem diferentes tipos de arquiteturas, com diferentes tipos de comportamentos, do puramente reativo, que reage a estímulos, ao baseado no modelo de crenças, desejos e intenções (BDI: belief, desire, intention). Podemos dividir as arquiteturas de agentes em quatro principais grupos: baseado em lógica, reativo, BDI e arquiteturas de camadas.

A Arquitetura b*aseado em lógica* (simbólico) baseia-se na tradicional técnica de sistemas baseados em conhecimento na qual um ambiente é simbolicamente representado e manipulado usando mecanismo de raciocínio.

A arquitetura *reativa* implementa a ação do agente como um mapeamento direto da situação para a ação e é baseada em mecanismo de estimulo-resposta, controlada por

sensores de eventos. Diferente da arquitetura baseada em lógica, esse tipo de agente não possui um modelo simbólico central e não utiliza qualquer raciocínio simbólico complexo.

O modelo *BDI* representa uma arquitetura cognitiva baseada em estados mentais, e tem sua origem no modelo de raciocínio prático humano. O nome atribuído ao modelo é justificado pelos seus estados mentais: crença, desejo e intenção (Belief, Desire and Intention). Uma arquitetura baseada no modelo BDI representa seus processos internos através dos estados mentais acima citados, e define um mecanismo de controle que seleciona de maneira racional o curso das ações.

Arquiteturas por *camadas* permitem ambos os comportamentos reativos e deliberativos. Para permitir essa flexibilidade, subsistemas organizados em camadas de hierarquia são utilizados para acomodar ambos os tipos de comportamento de agentes. Existem dois tipos de fluxo de controle dentro de uma arquitetura por camada: horizontal (Ferguson, 1991) e vertical (Muller at al*.,* 1995). As camadas horizontais são diretamente conectadas do sensor de entrada para as ações de saída. Essencialmente, cada camada age como um agente.

A principal vantagem é que com a simplicidade desse esquema, se o agente necessita de n diferentes tipos de comportamento, então a arquitetura vai requerer somente n camadas. Entretanto, uma vez que cada camada se comporta de uma forma diferente, suas ações podem ser inconsistentes necessitando de uma função de mediação para controlar as ações. Outra complexidade é a alta quantidade de possíveis interações entre camadas horizontais - m^n (onde *m* é o número de ações por camadas). A arquitetura de camada vertical elimina algum desses problemas como o sensor de entrada e a ação de saída que são tratados por no máximo uma camada cada (não criando ações inconsistentes). A arquitetura de camada vertical pode ser subdividida em arquiteturas de controle de *um passo* ou *dois passos*.

A figura 1 mostra este tipo de arquitetura

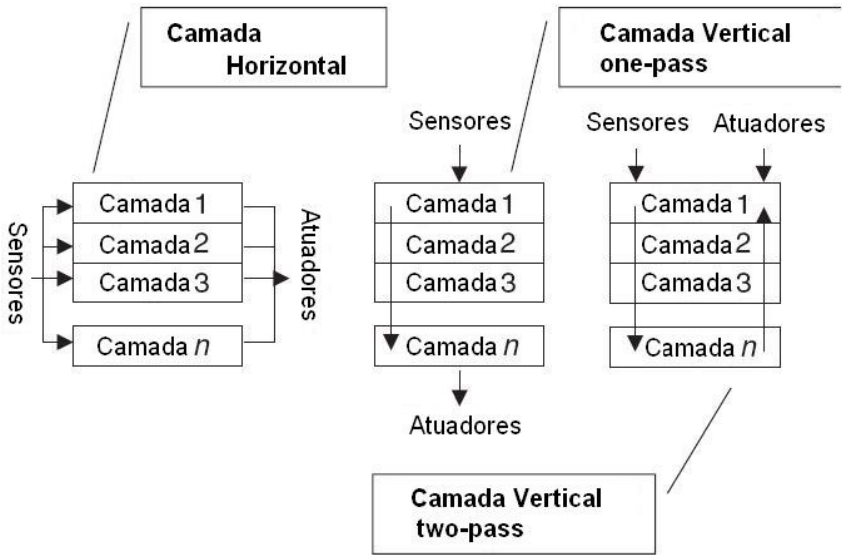

Figura 1 – Fluxo de dados e controle na arquitetura em camadas

Na arquitetura de *one-pass* o fluxo de controle começa da camada inicial que pega os dados dos sensores até a camada final que gera ações de saída. Na arquitetura de *two-pass,* o fluxo de dados passa pela seqüência de camadas e controle e então faz o caminho de volta. A principal vantagem da arquitetura por camada vertical é que as interações entre camadas são reduzidas significantemente para m2(n-1). A principal desvantagem é que a arquitetura depende de todas as camadas e não é tolerante a falhas, assim se uma camada falha, o sistema inteiro falha.

2.3 Programação Orientada a Agentes

Os paradigmas de programação têm evoluído para aumentar a capacidade de expressar idéias. Com isso, surgiram diversos tipos de novos paradigmas. É difícil para as pessoas conceberem estruturas que não podem descrever, verbalmente ou por escrito.

O surgimento da programação orientada a objetos veio como uma nova forma de expressar a idéia e abstrair o mundo real e a programação orientada a agentes traz uma nova abordagem para o desenvolvimento de sistemas com características especificas: software autônomo e sistema distribuído, podendo ser pensado como uma extensão da programação orientada a objetos (Odell, 2000). Porém a tecnologia de agentes fornece

suporte efetivo para resolver problemas em certas áreas de aplicação, onde outras tecnologias podem ser deficientes ou enfadonhos.

Atualmente há uma carência de metodologias baseadas em agentes. Essa deficiência tem sido uma barreira à utilização em larga escala de tecnologia de agentes (Luck, McBurney, Preist, 2003). Assim, a continuação do desenvolvimento e refinamento de metodologias para o desenvolvimento de sistemas multi-agentes é imperativo, uma área de tecnologia de agente que merece significante atenção.

Gaia (Wooldridge at al, 2000), MESSAGE (Caire at al, 2002), e Cassiopeia (Collinot, 1996) são algumas das metodologias de agentes existentes, porém a maioria das atuais metodologias tentam adaptar metodologias de análise e projetos de sistemas orientados a objetos para o projeto baseado em agentes (Wooldridge, 2002). Assim ocorrem diversas desvantagens, principalmente pelo fato de objetos e agentes serem diferentes abstrações, e como resultado, deve ser pensado em diferentes níveis (Odell, 2000).

3 METODOLOGIAS DE SISTEMAS BASEADO EM AGENTE

Nesse capitulo serão apresentados algumas das metodologias mais conhecidas para construir sistemas baseados em agentes.

As técnicas de engenharia de software convencional não se encaixam apropriadamente para projetar sistemas baseados em agentes devido às suas peculiaridades.

Engenharia de Software baseado em agentes pode ser separada em dois grupos distintos (Caire, 2001): O primeiro estende metodologias existentes sustentando a idéia de simplicidade e facilidade de usar técnicas já consolidadas e bem conhecidas. O segundo grupo propõe novas metodologias baseadas na teoria de agentes; esse grupo alega que esse tipo de abordagem captura da melhor forma as características especiais do paradigma de agente.

3.1 GAIA

GAIA (Wooldridge at al, 2000) é uma metodologia para análise e projeto de sistemas multi agentes, cobrindo a arquitetura e a sociedade de agentes que é vista como uma organização computacional de diversas regras de interação entre agentes. Essa abordagem objetiva capturar aspectos de sistemas multi agentes como uma estrutura organizacional, interação e comportamento dirigido a objetivo, de uma forma mais eficiente do que a abordagem orientada a objetos.

Essa metodologia é independente da fase subseqüente de implementação do projeto. GAIA inicia o processo do ponto onde tradicionais engenharias de software terminam.

O relacionamento de modelo da metodologia GAIA é mostrado na figura 2, abaixo.

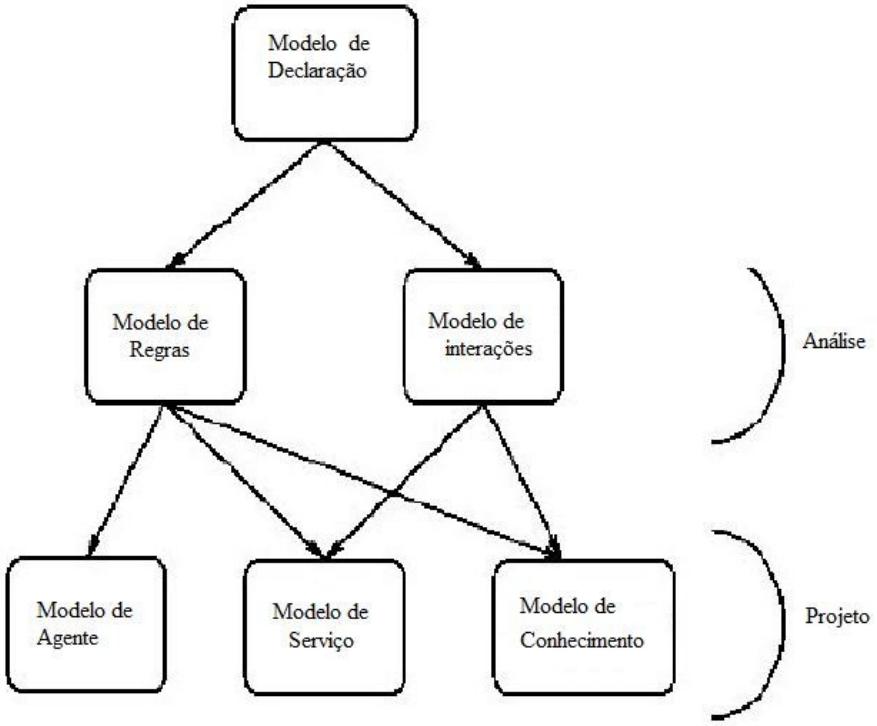

Figura 2 – Relação de modelos GAIA.

O objetivo do processo da análise é promover a compreensão do sistema e sua organização e estrutura abstraindo detalhes de implementação. Nessa primeira etapa deve-se capturar a organização do sistema, que é representado pelo conjunto de papéis.

Cada papel interage com outros e podem ser visto como uma entidade abstrata que se torna um personagem concreto quando é utilizado por um agente. Um agente pode utilizar mais de um papel e cada papel pode ser utilizado por muitos agentes. Um papel é definido por quatro diferentes atributos:

- Responsabilidade: é o atributo chave ou a regra que define quais as funções do papel.
- Permissões: define os recursos disponíveis que um papel pode usar ou gerenciar.
- Atividades: define uma computação interna que o papel pode realizar sem interação com outros papéis. Eles são as ações privadas dos papéis.
- Protocolos: descreve as habilidades de comunicação de um papel e define o conjunto de protocolos realizados pelos papéis.

O modelo dos papéis identifica a chave dos papéis do sistema que pode ser visto como uma abstração de funções associadas a entidades. A responsabilidade e

permissões associadas para cada regra são representadas por uma notação formal baseado no processo da metodologia orientada a objeto chamada fusion (Coleman at al, 1994).

Devido a dependência da relação entre os papéis devemos definir um modelo de interação. Esse modelo define um conjunto de protocolos que interagem entre os papeis e define a classe de interações e seu propósito.

O processo de análise da metodologia GAIA pode ser sumarizado nos seguintes passos:

- Identificar os papéis utilizados internamente no sistema e construir um protótipo de modelos de papeis a partir da descrição informal de cada papel.
- Identificar os protocolos associadas a esses papeis e construir um modelo de interação.
- Construir modelos de papeis formais baseado no modelo de interação.
- Voltar ao primeiro passo interagindo com cada modelo para refiná-los.

O objetivo da fase de projeto da metodologia GAIA é transformar os modelos abstratos derivados da fase de análise em modelos com baixo nível de abstração podendo ser facilmente implementado usando um método de projeto de engenharia de software como a metodologia orientada a objeto.

A fase de projeto proposta pela metodologia GAIA possui três modelos formais:

- A intenção do modelo de agente GAIA é documentar vários tipos de agentes que serão usados em cada sistema e as instâncias de agente que irão realizar esses tipos de agentes em tempo de execução. Um tipo de agente é mais bem pensado como um conjunto de papeis de agente.
- Um modelo de agente é definido usando uma árvore de tipos de agente simples, cada nodo folha corresponde a papeis, (como definido pelo modelo de regras), e outros nodos correspondem a tipos de agentes. Se um tipo de agente t1 tem filhos t2 e t3, então isso significa que t1 é composto de papeis que perfazem t2 e t3.

O objetivo dos modelos e serviços do GAIA é identificar os serviços associados com cada papel de agente, e especificar as principais propriedades desses serviços.

Um serviço pode ser visto como uma função de um agente. Em termos de Orientação a Objeto, um serviço corresponderia a um método, entretanto, não significa que serviços estão disponíveis a outros agentes do mesmo modo que métodos de objetos são disponibilizados para ser invocados por outros objetos. Um serviço é simplesmente um bloco de atividade que um agente irá empenhar.

Modelos de conhecimento simplesmente definem quais ligações de comunicação existirá entre os tipos de agentes. Eles não definem que mensagem será enviada ou quando uma mensagem será enviada. Eles simplesmente indicam qual caminho de comunicação existe. O propósito de um modelo de conhecimento é identificar qualquer potencial gargalo de comunicação, que pode causar problemas em tempo de execução.
É uma boa prática assegurar que sistemas sejam fracamente acoplados, e o modelo de conhecimento pode ajudar a fazer isso.

O modelo de conhecimento de agente é simplesmente um grafo: os nodos no grafo correspondem a um tipo de agente e arcos no grafo correspondem a caminhos de comunicação.

Os estágios da fase de projeto de GAIA podem ser sumarizados assim:

- Criar um modelo de agente: agregar papéis em tipos de agentes, e refinar para formar um tipo de agente hierárquico e documentar as instâncias de cada tipo de agente usando anotações de instância.
- Desenvolver um modelo de serviços, examinando atividades, protocolos.
- Desenvolver uma modelo de conhecimento a partir do modelo de interação e modelo de agente.

3.2 MaSE

A metodologia Multiagent System Engineering (MaSE) (Wood e Deloach, 2000), pega a especificação inicial do sistema e produz um conjunto de documentos formais em um estilo baseado em gráfico. O principal foco da MaSE é guiar o projetista através do ciclo de vida do software de uma especificação natural para um sistema de agente implementado.

MaSE é independente de uma arquitetura de sistema multi-agente particular, arquitetura de agente, linguagem de programação, ou forma de envio de mensagem de um sistema. A Figura 3 apresenta uma visão geral da metodologia e dos modelos MaSE.

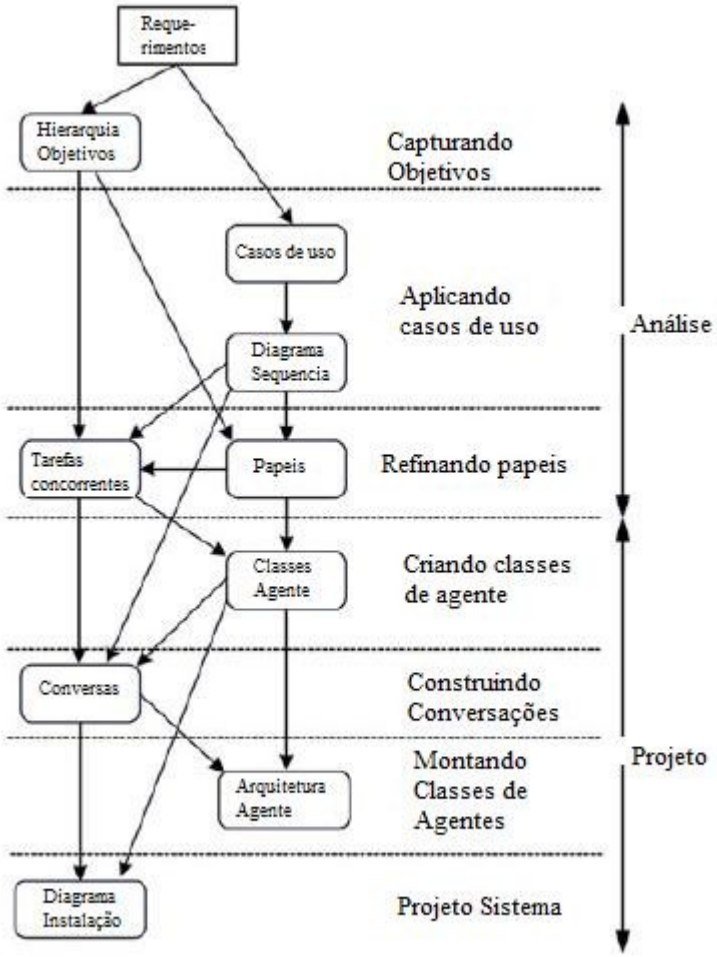

Figura 3 – Metodologia MaSE.

Um sistema projetado utilizando a metodologia pode ser implementado de diferentes formas a partir de um mesmo design. MaSE também oferece a habilidade de rastrear mudanças por toda parte do processo. Todo objeto projetado pode ser rastreado de frente ou de trás através de diferentes fases da metodologia e da construção correspondente.

A primeira fase pega a especificação inicial do sistema e transforma em uma conjunto de estrutura de objetivo como mostrado no Diagrama de Hierarquia de Objetivo. Na metodologia MaSE, os objetivos são sempre definidos como nível de sistema. Há duas partes na fase de capturar os objetivos: Identificar e estruturar os objetivos.

A principal seqüência de detalhes de interação e de subordinação deve ser distinguida uma da outra. Cada nível da hierarquia contém objetivos que são iguais em escopo e todos sub-objetivos se relacionam funcionalmente com seus pais.

A aplicação da fase de Caso de Uso captura casos de uso de um requerimento inicial do sistema e reestrutura-os como um diagrama de seqüência. O diagrama de seqüência descreve uma seqüência de mensagem entre múltiplos papeis de agente.

Os casos de usos são esboçados de um requerimento de sistema como descrições narrativas de seqüência de eventos que define o comportamento desejado do sistema. Além disso, um Diagrama de Seqüência é usado para determinar o conjunto mínimo de mensagem que deve ser passado entre regras. Se uma mensagem é passada entre dois papeis, deverá haver um caminho de comunicação correspondente entre eles.

O terceiro passo da metodologia MaSE é transformar os objetivos estruturados do diagrama de Hierarquia de objetivos em uma forma mais útil para construir sistemas Multi-Agentes.

Papéis são os blocos de construção usados para definir classes de agente e capturar objetivos do sistema durante a fase de projeto. Esse passo deve garantir que os objetivos do sistema são contados para assegurar que todo objetivo seja associado com um papel e que todo papel seja desempenhado por uma classe de agente. Um papel é uma descrição abstrata de uma função esperada de uma entidade. Papeis são criados para fazer alguma coisa. Eles são similares a noção de um ator em um teatro ou um escritório dentro de uma organização.

Cada objetivo do sistema deve ser mapeado par um papel, de um para um. Entretanto, existem várias situações excepcionais onde é útil combinar alguns objetivos. Objetivos similares ou relacionados podem ser combinados dentro de um único papel por motivo de eficiência ou mesmo conveniência.

Definições de papéis são capturadas em um Modelo de Papéis que incluem informações de interação entre tarefas de papeis. Nesse modelo, linhas entre papeis denotam possíveis caminhos de comunicações entre papeis.

As classes de agente são identificadas a partir de componentes de papeis. O produto dessa fase é um Diagrama de Classe de Agente que representa classes de agente e conversações entre eles. Classes de Agente consiste de dois componentes: papeis e conversações. As conversações de uma classe de agente são aquelas em que ele participa.

A principal diferença entre um Diagrama de Classe de Agente e um similar diagrama de objeto é a semântica das relações entre as classes de agentes. Nesses diagramas, as relações definem conversas que são mantidas entre classes de agentes. A proposta dessa fase é identificar a classe de agente que estaca cada lado de uma conversa.

Uma conversação MaSE define um protocolo de coordenação entre dois agentes. Uma conversação consiste de dois Diagramas de Classe de Comunicação, uma para o iniciador da conversa e outro para o destinatário da conversa. Um Diagrama de Classe de Comunicação é um par de máquinas de estados finitos que definem os estados de conversação de duas classes de agentes participantes.

Na fase de Montagem de Agente a arquitetura interna de uma classe de agente é criada. Existem cinco diferentes estilos de modelo de arquiteturas: BDI, reativo, planejado, baseado em conhecimento, e uma arquitetura definida pelo usuário. Cada modelo de arquitetura tem um conjunto específico de componentes.

O projetista pode definir um componente do zero ou usar componentes pré-existentes. Além do mais, componentes podem ter sub-arquiteturas contendo componentes. Componentes são ligados com conectores de agentes interno ou externo. Conectores de agentes internos definem as visibilidades entre os componentes enquanto conectores externos definem conexões com recursos externos como outros agentes, sensores, atuadores e banco de dados.

A fase final da metodologia MaSE pega as classes de agente e instancia-os como atuais agentes. É usado um diagrama de instalação para mostrar o número, tipo e localização dos agentes dentro do sistema.

3.3 AALAADIN

AALAADIN é uma metodologia baseada em papéis e organizações. Essa metodologia apresenta uma abordagem top-down mais normativa para gerenciar as relações sociais e os modelos de organização.

A proposta da AALADIN é baseada em três conceitos básicos: agente, grupo, e propósito.

- Agente: vista como uma entidade ativa que pode interagir com outras entidades e desempenhar um papel dentro de um grupo de agentes
- Grupo: um conjunto de agentes. Cada agente pode ser membro de muitos grupos.
- Papel: é uma representação abstrata de um conjunto de características desempenhado por um agente

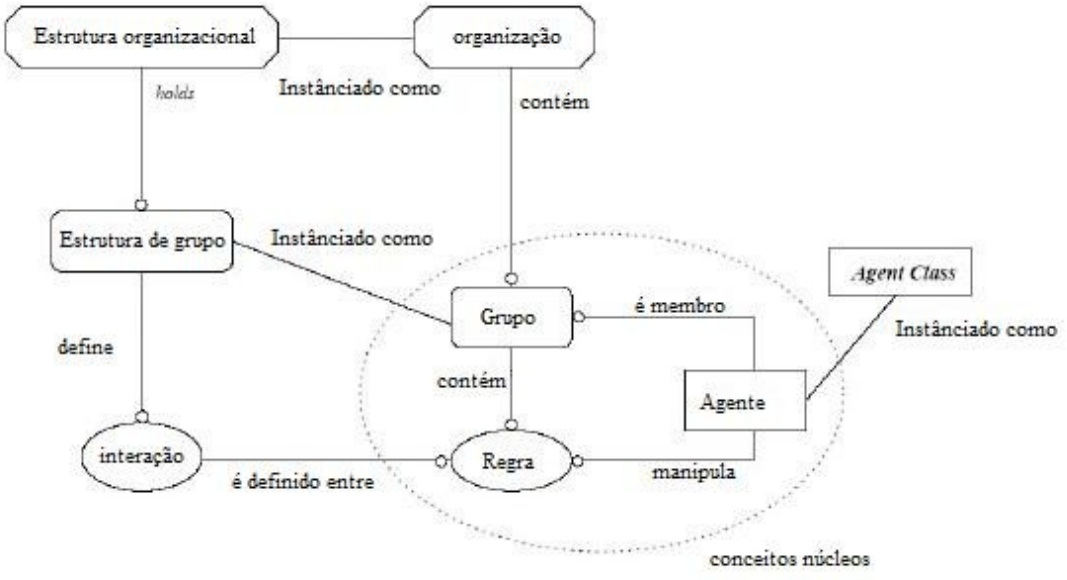

Figura 4 – Descrição da organização dos agentes

O nível de abstração mostrado na figura 4 define o conjunto de papéis e interações válidas entre esses papéis, a estrutura do grupo e a organização. O conceito introduzido nesse nível funciona como uma ferramenta para o processo de análise e projeto.

- Estrutura de grupo: identifica todo conjunto de papeis dentro de um grupo e as interações entre eles.
- Estrutura organizacional: define um esquema organizacional de um sistema multi agente

O problema dessa metodologia é que as conseqüências dessas relações sociais não podem ser modeladas e, agentes individuais que são praticamente caixas pretas. Conseqüentemente a metodologia força que os agentes se comportem mais como agentes reativos do que deliberativos.

3.4 Agent UML

O Agente UML (AUML) é resultado de um trabalho cooperativo entre a FIPA (Fecundativo of Intelligent Physical Agents) e a OMG (Object Management Group).
A abordagem foca em melhorar o uso da engenharia de software orientado a agente usando uma extensão dos diagramas UML.

Dois novos diagramas foram incluídos na notação UML para dar suporte as características da abordagem orientada agente (Bauer et al, 2001): o diagrama de protocolo e o diagrama de classe de agente. O diagrama de protocolo estende o diagrama de seqüência do UML e o diagrama de classe de agente estende o diagrama de classe. O diagrama de protocolo descreve um padrão de comunicação incluindo todas as seqüências de mensagem entre os agentes quando eles estão atuando seus papeis.

As ações comunicativas definem os tipos e o conteúdo das mensagens. Os protocolos de interação são representados por um diagrama de protocolo que define o comportamento cooperativo de um grupo de agentes.

O diagrama de protocolo inclui os conceitos de papéis de agente e as threads de interação correspondente ao ciclo de vida dos agentes.

AUML propõe uma extensão do diagrama de classe UML para representar os agentes (Bauer, 2002). A figura 5 apresenta alguns tipos de classes de agentes: a primeira representa a classe de agente e a segunda alguma classe de agente que satisfaz algumas regras, e a terceira algumas instancias de classe de agente.

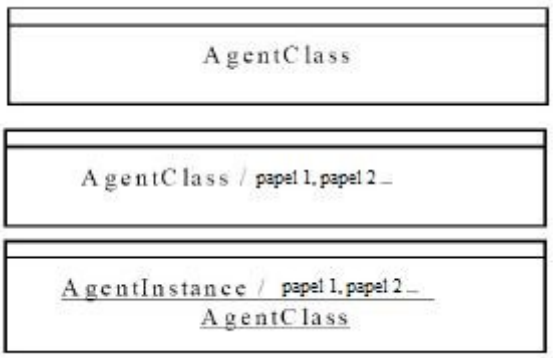

Figura 5 - Classe de agente AUML

No diagrama de classe de agente é definido o nome das classes e seus papeis associados. O diagrama representa as ações de comunicação autômato e associativa que definem os tipos de mensagem, o conteúdo de seu remetente e de seu destinatário. A ação autômato define o comportamento dos agentes por um mapeamento de ações comunicativas de envio de acordo as ações comunicativas recebidas (comportamento reativo) ou de acordo com algumas condições (comportamento proativo). A figura 6 apresenta o diagrama de classe de agente.

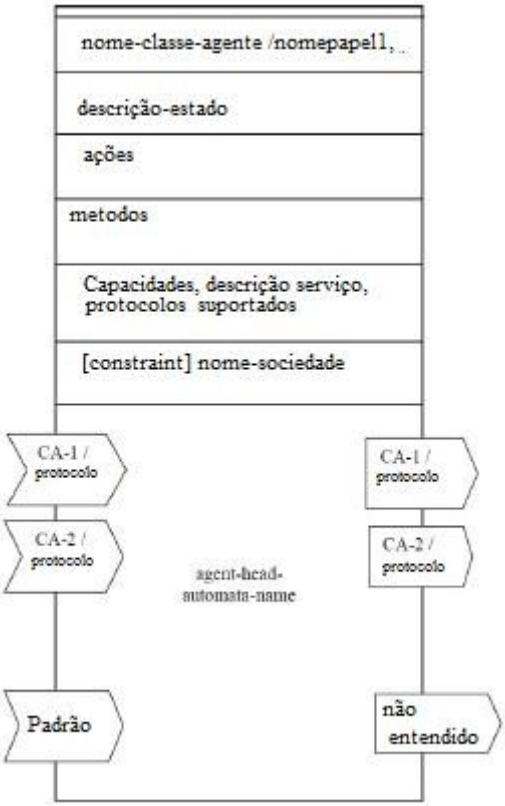

Figura 6 – Diagrama de classe de agente

3.5 MESSAGE/UML

O AUML é uma abordagem que não considera a noção forte de agente. Agentes são classes em AUML. Diagramas de estado de transição modelam seus comportamentos e diagramas de interação, estendido para suportar ações comunicativas, modelam suas comunicações. Como AUML suporta somente noção fraca de agente, podendo ser usado em conjunto com todas as outras metodologias para modelar comunicação de Sistemas Multi-Agentes.

A MESSAGE/UML é outra metodologia relacionada com UML, fornecendo boas ferramentas de projeto para especificar a arquitetura de software de agentes individuais.

Existe um nível de organização, mas esse nível não fornece boas abstrações como a usada para modelos de agentes individuais. A metodologia MESSAGE/UML (Caire et al, 2001) é focado para aplicações industriais. Essa metodologia almeja capturar aspectos positivos de metodologias orientadas a objeto e teoria de agentes.

A MESSAGE/UML estende os diagramas de classe e de atividade da UML para capturar características e propriedades de sistemas multi-agentes. Esse nível de abstração é chamado de nível de conhecimento. A notação UML é usada para descrever detalhes dos níveis estruturais. Cinco visões do modelo são usadas durante a fase de projeto. Essa definição é feita baseada no conceito de nível de conhecimento que possui três categorias: entidades concretas, entidades de estado metal e atividade. As entidades concretas são:

- Agente: uma entidade autônoma com características funcionais. Sua característica é descrita de acordo com os serviços. Um serviço é um pouco similar a métodos na abordagem orientada a objetos. As ações do agente é guiado por seu propósito devido a sua autonomia;
- Organização: é um grupo de agentes compartilhando os mesmos propósitos;
- Papéis: descreve características externas de um agente em algum contexto especifico;
- Recursos: é usado para representar qualquer entidade sem autonomia (banco de dados, programa externo, etc);
- Tarefa: é executado quando alguma pré condição ocorre. É esperado que após uma tarefa uma nova situação ocorra (pós-condição). Uma tarefa pode ser composta de sub tarefas ou ações.
- Interação: anseia obter uma visão consistente de um domínio e como o problema pode ser resolvido baseado em protocolos que define ações comunicativas.

Entidade de informação é outro conceito que consiste em um objeto capturar informações sobre eventos e recursos. Cada visão de um modelo anseia descrever um subconjunto de entidades e suas relações:

- Visão organizacional: mostra as entidades concretas, o ambiente e suas relações;

- Visão de tarefa: apresenta os objetivos, situações de tarefas e dependências entra eles. A tarefa e o objetivo é associado a situação e representa conexões entre eles;
- Visão de Agente/Papel: descreve os agentes individuais e seus papeis;
- Visão de domínio: apresenta conceitos específicos de um domínio de aplicação.

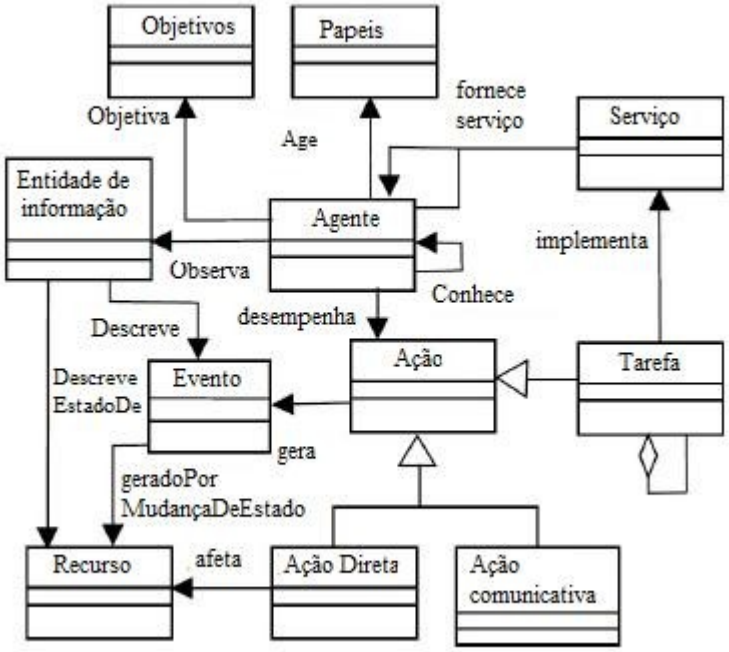

Figura 7 - Conceitos da Metodologia MESSAGE/UML

4 METODOLOGIA DO TRABALHO

Uma metodologia serve como guia para o desenvolvedor ao projetar um sistema. Em geral, uma metodologia de desenvolvimento de software consiste em:

•Um processo, seqüência de fases e passos que guia o desenvolvedor ao construir um sistema.

•Um conjunto de regras de heurísticas que dá apóio ao desenvolvimento para escolhas relevantes.

•Um número de artefatos: diagramas, esquemas, representação de documento graficamente ou textualmente.

•Um conjunto de padrões que podem ser aplicados para resolver situações comuns.

•Uma ou mais ferramentas que: automatiza o melhor possível, as fases e passos especificados no processo; que mantêm consistência entre os modelos produzidos, geração de código e documentação, etc.

O processo descrito cobre a fase de análise e design. A fase de análise é geral por natureza e independente da plataforma adotada, diferentemente da fase de design que nesse trabalho assumimos o uso do framework JADE (http://jade.cselt.it) como plataforma de implementação e que é focado diretamente nas classes e conceitos providos pelo JADE.

Não há uma fronteira entra as fases de análise e design, permitindo ao projetista mover entre as fases deliberadamente.

Ao final da fase de design, deve-se estar hábil seguir o processo direto para a implementação, a criação do código. Considerando ainda que muito desse processo pode ser feito com ferramentas de ajuda para automatização do processo de implementação. Entretanto, no momento, e do mesmo modo, não está endereçado na corrente versão dessa metodologia proposta, deixando esse tópico para futuros trabalhos.

Uma questão está incluída nessa metodologia proposta: O projetista fez uma escolha racional ao usar uma solução baseada em agentes? Se a resposta for sim, o projetista move-se para a etapa da análise, se a resposta for não, o projetista deve procurar por uma solução alternativa.

Como em toda metodologia, algumas suposições devem ser feitas:

- A definição de agente definida no Capitulo 2 é assumida.
- A plataforma escolhida para implementação é o JADE.
- O número de agentes é relativamente pequeno.
- A estrutura organizacional do sistema é estático, significando que comportamentos emergenciais em tempo de execução não é esperado.
- Segurança não é levado em consideração.

4.1 Adequação do uso de agentes

Para responder essa questão temos que levar em consideração todos os aspectos da abordagem orientada a agente.

O European Institute for Research and Strategic Studies in Telecommunications (EUROSCOM) ao iniciar um projeto que explorava o uso de tecnologias de agentes dentro da indústria de telecomunicação da Europa definiu uma estratégia para abordar esse problema. Um dos três objetivos do projeto foi definir orientação para a identificação de áreas de aplicação onde a abordagem baseada em agentes é melhor condizente do que outras abordagens (MESSAGE, 2000). O consórcio produziu as seguintes diretrizes para ajudar o projetista decidir quando uma abordagem baseada em agente é interessante, ou não:

- Uma abordagem baseada em agente é benéfica em situações onde são necessários diversos tipos de comunicação.
- Uma abordagem baseada em agente é benéfica quando o sistema deve se comportar bem em situações onde não é prático ou possível especificar seu comportamento utilizando a forma básica caso-por-caso.
- Uma abordagem baseada em agente é benéfica em situações envolvendo negociação, cooperação e competição de diferentes entidades.
- Uma abordagem baseada em agente é benéfica quando o sistema deve atuar autonomamente.
- Uma abordagem baseada em agente é benéfica quando sabe-se de antemão que o sistema irá ser expandido, modificado ou quando o sistema proposto espera ser modificado.

Se o sistema a ser projetado encaixa-se em uma ou mais dessas diretrizes então a abordagem baseada em agente pode ser utilizada.

4.2 Java Agent Development Framework

Jade é um framework totalmente implementado na linguagem Java, simplificando o desenvolvimento de sistemas multi-agentes através do middle-ware que está de acordo com as especificações FIPA (http://www.fipa.org/) e com um conjunto de ferramentas gráficas que suportam debugging e frases de organização.

A plataforma de agentes pode ser distribuída através de máquinas (que nem mesmo precisam compartilhar do mesmo sistema operacional) e a configuração pode ser controlada por uma interface gráfica remota.

A configuração pode até mesmo ser mudada em tempo de execução movendo agentes de uma máquina a outra, como e quando necessário. O único requisito para utilizar o framework Jade é o Java SDK 1.4 ou superior.

JADE é software livre e é distribuído pela Telecom Itália, em um código fonte aberto sobre os termos da licença LGPL (Lesses General Public License Version 2).

Algumas das empresas presentes como membros do JADE Board:
Telecom Itália, Motorola, Whitestein Technologies AG, Profactor GmbH, France Telecom R&D.

O primeiro desenvolvimento de software, que eventualmente se tornou a plataforma JADE, foi iniciado pela Telecom Itália em 1998, motivado pela necessidade de validar as especificações FIPA. Com o JADE não precisamos nos preocupar com a linguagem FIPA e suas especificações.

No inicio, desenvolver uma plataforma não era o objetivo dos criadores, porém, graças ao suporte financeiro do European Commission (FACTS Project, ACTS AC317) e a boa vontade e capacidade do time de desenvolvimento, em um certo ponto foi decidido mover alem de uma aplicação de simples validação de especificações FIPA para o desenvolvimento de um *middleware* inteiro. A visão era de fornecer serviços para desenvolvedores de aplicações e isso foi prontamente acessível e usável por desenvolvedores com algum conhecimento das especificações FIPA, assim como recém-chegados ao ambiente. Ênfase dada na simplicidade e usabilidade de APIs de softwares.

O código fonte foi aberto em 2000 e foi distribuído pela Telecom Itália com a licença LGPL (Library Gnu Public Licence). Essa licença assegura todos os direitos básicos para facilitar a utilização do software incluído em produtos comerciais: "o direito de fazer cópias de software e distribui-las, o direito de ter acesso ao código fonte, e o direito de mudar o código e fazer melhorias nele" (Free Software Foundation, 2007). Diferente da licença GPL, a LGPL não põe nenhuma restrição no software que usa JADE, e isso permite softwares proprietários serem misturados com softwares cobertos pela licença LGPL. De outro lado, a licença requer também que trabalhos derivados em JADE, ou qualquer outro trabalho baseado nele, deve ser retornado para a comunidade com a mesma licença.

Para facilitar o envolvimento da indústria no projeto JADE, em Maio de 2003 a Telecom Itália Lab e a Motorola Inc. definiram um acordo de colaboração e formou o JADE Governing Board, uma organização sem fins lucrativos de empresas designadas ao desenvolvimento e promoção da plataforma JADE. Atualmente fazem parte do JADE Board, Telecom Itália, Motorola, France Teleco R&D, Whitestein Technologies AG e Profactor GmbH.

Quando JADE tornou-se público pela Telecom Itália, era utilizado quase que exclusivamente pela comunidade FIPA, mas como suas características cresceram para além das especificações da FIPA encadeou distribuição para uma comunidade global de desenvolvedores. É interessante notar que o JADE contribuiu para a difusão da especificação FIPA fornecendo um conjunto de ferramentas que escondem as próprias especificações, programadores podem implementar de acordo com as especificações sem precisar estudá-las. Isso é uma das principais qualidades do JADE em relação à FIPA.

Uma das primeiras extensões do núcleo do JADE foi fornecido por LEAP [LEAP] (IST 1999-10211), um projeto parcialmente financiado pelo European Commission que contribuiu significantemente entre 2000 e 2002 para portar JADE para a plataforma para o ambiente Java Micro Edition e Wireless Network. Esse trabalho, em grande parte, foi liderado por Giovanni Caire. Hoje, a possibilidade de executar o JADE para as plataformas J2ME-CLDC e CDC é considerado como uma das características do JADE.

4.2.1 JADE e o paradigma de agentes

Jade é uma plataforma de software que fornece funcionalidades básicas de camada - midleware que são independentes de aplicações especificas e que simplifica a realização de aplicações distribuídas que se aproveita da abstração de agente de software (Wooldridge and Jennings, 1995). Um dos grandes méritos do JADE é que ele implementa essa abstração em uma bem conhecida linguagem orientada a objetos, Java, fornecendo uma API simples e amigável. As seguintes escolhas de projeto foram influenciadas pela abstração de agente (Bellifemine, 2007):

Um agente é autônomo e proativo: Como dito no Capitulo 2, um agente é proativo. Um agente deve ter sua própria *thread* de execução, usando isso para controlar seu próprio ciclo de vida e decidir autonomamente quando deve executar as ações.

Agentes podem dizer 'não', e eles são fracamente acoplados: Comunicação baseado em mensagem assíncrona é a forma básica de comunicação entre agentes no *framework* JADE; um agente que deseja se comunicar deve enviar uma mensagem para um destinatário identificado (ou um conjunto deles). Não existe dependência temporal entre remetente e destinatário: um destinatário pode não estar disponível quando o remetente enviar a mensagem. Também não é necessário obter a referência do objeto do agente destinatário, somente o nome do agente destinatário. É possível também que algum destinatário seja desconhecido pelo remetente, que pode definir um conjunto de destinatário usando um grupo intencional (todos os agentes que fornecem o serviço "*Book-Selling*") ou sendo mediado por um agente de proxy (propagando essa mensagem para todos os agentes no domínio "selling.book.it"). Além disso, essa forma de comunicação permite aos destinatários selecionar quais mensagens processar e quais descartar, como definido em sua tabela de prioridades (ex. leia primeiro as mensagens de tal domínio... etc). Isso também permite que o remetente controle sua *thread* de execução e assim não ser bloqueado até que o destinatário processe a mensagem. Finalmente, isso fornece uma vantagem interessante quando implementado comunicação *multi-cast* como uma operação atômica do que N consecutivas chamadas de métodos (uma lista de múltiplas mensagens em vez de diversas chamadas de método para cada objeto remoto na qual se deseja comunicar).

O sistema é peer-to-peer: Cada agente é identificado por um nome único global (o AgentIdentifier, ou AID, como definido pela FIPA). Pode entrar e sair de uma

plataforma hospedeira a qualquer tempo e pode descobrir outros agentes através dos serviços de páginas amarelas e paginas brancas (serviço fornecido em JADE pelo AMS e pelo agente DF como definido também pelas especificações FIPA).

JADE foi implementado para fornecer aos programadores as seguintes funcionalidades, fáceis de usar e personalizar:

- Um sistema distribuído habitado por agentes, cada um rodando em uma *thread* separada, potencialmente em maquinas remotas, e capazes de comunicar transparentemente uns com os outros, a plataforma JADE abstrai a infra-estrutura de comunicação.
- Segue inteiramente as especificações FIPA: Como já dito anteriormente a plataforma JADE segue rigorosamente todo o padrão de especificação FIPA.
- Eficiente transporte de mensagens assíncronas
- Implementação dos serviços *yellow pages* e *white pages*.
- Um gerenciamento do ciclo de vida dos agentes simples e efetivo: Quando um agente é ele criado é automaticamente vinculado a um nome global único e a um endereço de transporte que é usado para se registrar com o serviço de paginas branca de sua plataforma. Simples API e ferramentas gráficas são fornecidas para gerenciar o ciclo de vida dos agentes, tanto localmente quanto remotamente, criar, suspender, migrar, clonar e matar.
- Suporte a agentes móveis. Código de agentes e, com algumas restrições, estados de agentes podem migrar entre processos e máquinas. A migração dos agentes é feita de forma transparente comunicando outros agentes que podem continuar interagir mesmo durante o processo de migração.
- Um conjunto de ferramentas gráficas para dar suporte na monitoração e depuração. Isso é particularmente importante e complexo quando se fala em *multi-threads*, multi-processos e multi-maquinas tais como uma típica aplicação JADE.
- Suporte à ontologias e conteúdo de linguagens. Verificação de ontologias e codificação do conteúdo é feito automaticamente pela plataforma.
- Bibliotecas de protocolos de interação que modelam padrões típicos de comunicação orientado para realização de uma ou mais metas.
- Integração com várias tecnologias Web incluindo JSP, servlets, applets e Web services.

- Suporte para a plataforma J2ME e ambientes sem fio. JADE inclui suporte para plataformas J2ME-CDC e –CLDC através de um conjunto uniforme de APIs cobrindo os ambientes J2SE e J2ME.

4.2.2 Arquitetura JADE

A plataforma JADE é composta de *conteiners* de agentes que podem ser distribuídos através da rede. Agentes vivem em *conteiner*s que são processos Java fornecido pelo *run-time* JADE e todos os serviços necessários para hospedagem *e execução de agentes.* Existe um *contêiner* especial, chamado de *contêiner* principal, que representa o ponto de *bootstrap* da plataforma: é o primeiro contêiner pra ser lançado e todos os outros *conteiners* devem se juntar ao contêiner principal para se registrarem com ele.

A figura 8, extraída da documentação JADE, mostra os principais elementos arquitetônicos da plataforma JADE.

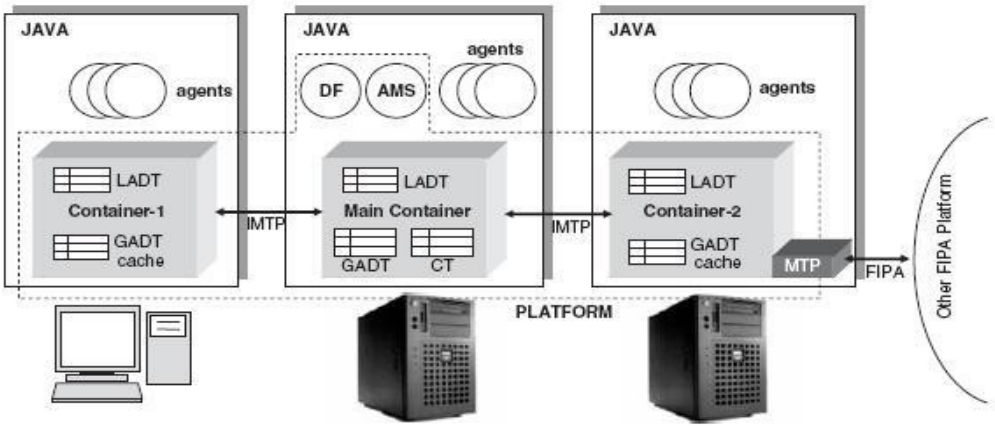

Figura 8 - Os principais elementos arquitetônicos da plataforma JADE. (Jade, 2008)

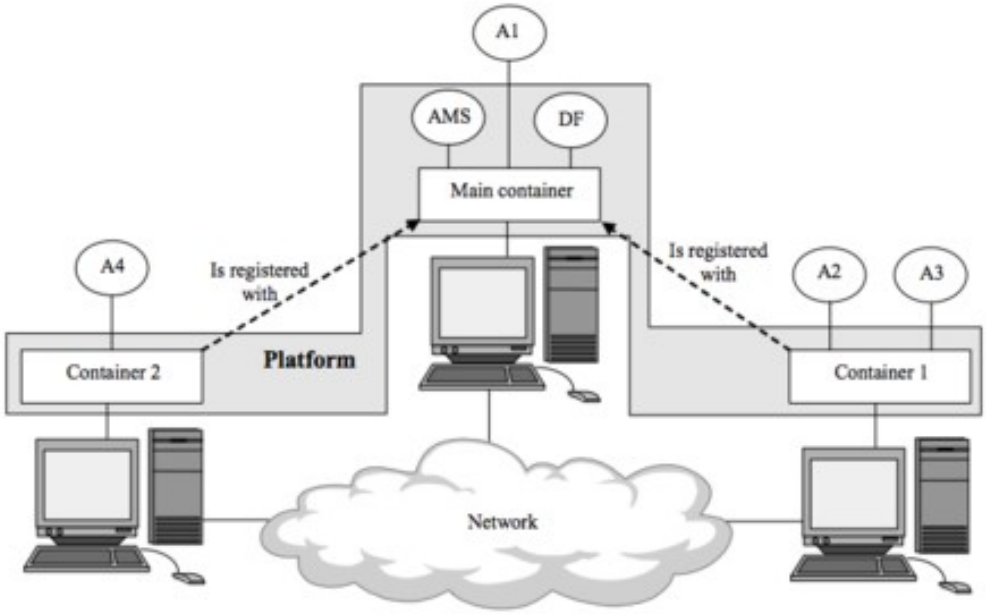

Figura 9 – Visão geral da arquitetura JADE (Jade, 2008)

A figura 10 esquematiza a relação entre os principais elementos arquitetônicos.

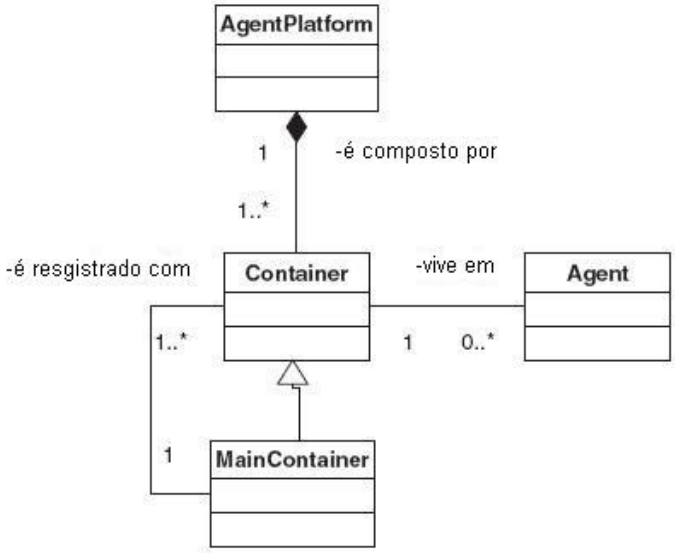

Figura 10- Relação entre os principais elementos arquitetônicos.

O programador identifica os contêineres usando um nome lógico, por padrão o contêiner principal é nomeado de ‘Main Container’ enquanto os outros são chamados de ‘Container-1’, ‘Container-2’, etc.

Como um ponto de bootstrap, o contêiner principal tem as seguintes responsabilidades especiais:

- Gerenciar a tabela de contêineres, que é o registro das referencias de objetos e dos endereços de transporte de todos os contêineres nó que compõe a plataforma.
- Gerenciar a tabela global de descrição dos objetivos dos agentes, que é o registro de todos os agentes presentes na plataforma, incluindo seu estado corrente e sua localização.
- Hospedar o AMS e o DF, os dois agentes especiais que fornecem a administração dos agentes e o serviço de paginas brancas, e o serviço padrão de paginas amarelas da plataforma, respectivamente.

Com essas informações, o projeto de modelagem a seguir fica mais focado, ou seja, a modelagem proposta no capitulo seguinte terá em mente a utilização da plataforma JADE. A plataforma JADE consegue provar que é uma eficiente plataforma para desenvolvimento de aplicações orientada a agentes distribuídos.

5 ORGANIZADOR DE PARTIDAS DE GAMES: OPG.

Nesse capitulo será apresentado o processo de análise, projeto e implementação do sistema proposto, como um estudo de caso da metodologia adotada.

A metodologia será ilustrada aplicando-a a um cenário chamado de Sistema Organizador de Partidas e Campeonatos de Games. Neste cenário, é assumido que uma operadora de telefone celular do mundo MMORPG queira fornecer aos seus assinantes, os usuários do jogo, um serviço que irá dar suporte a grupos de amigos que queriam marcar partidas de jogos. O serviço deve permitir ao assinante: convidar amigos para iniciar uma partida, coletar preferências dos que convidam e dos que são convidados, e sugerir a opção de jogo que melhor casa com a preferência do grupo.

O MMORPG é um jogo que envolve diversos países e localizações e seus usuários podem navegar entre esses países, planetas e localizações.

O serviço deve ser acessível pelos usuários através de seus telefones celulares. É assumido também que a operadora de celular tem as informações com os jogos disponíveis, dependendo da região virtual do usuário. Tem também informações de campeonatos e jogos no respectivo local do usuário.

Os usuários podem se mover livremente de locais em locais e o sistema deve levar isso em conta. A localização do usuário deve ser baseada na localização do sistema da operadora móvel. O sistema permite recuperar a posição do telefone celular dado o seu numero de telefone e notificações de relevantes mudanças na posição do celular.

5.1 Análise

A fase da análise almeja clarificar o problema sem qualquer interesse na solução (Dennis and Wixom, 2000). Na metodologia utilizada, iremos dividir a parte de análise em diversos passos.

Os passos da análise da metodologia seria:

- Passo 1: Caso de Uso.
- Passo 2: Identificação inicial dos tipos de agente
- Passo 3: Identificação das responsabilidades
- Passo 4: Identificação das ligações de conhecimento

- Passo 5: Refinamento de Agentes
- Passo 6: Organização de informação dos agentes
- Iterar entre os passos 1 à 6.

5.1.1 Casos de Uso

Casos de usos é um modo efetivo de capturar os potenciais requerimentos funcionais de um novo sistema. Cada caso de uso apresenta um ou mais cenários que demonstram como o sistema deve interagir com o usuário final, ou outros sistemas, para completar um objetivo especifico. Existem numerosas quantidades de padrões para especificar casos de uso. O mais popular é a especificação Unified Modeling Language (UML), que é definida em notação gráfica (como alternativa, é possível produzir casos de uso escritos). Casos de uso são usados extensivamente em praticantes da programação orientada a objetos, porém suas aplicabilidades não estão restritas a sistemas OO, porque casos de uso não são orientados a objetos em natureza (Hamptom at al, 1997). Por isso, também é possível aplicar casos de usos (sem modificações) para capturar requerimentos funcionais de sistemas multi-agentes.

Baseado na descrição do OPG e após entrevista com potenciais usuários do sistema, é possível construir uma lista preliminar de possíveis cenários. Com isso, produzimos o caso de uso mostrado na figura 11, abaixo.

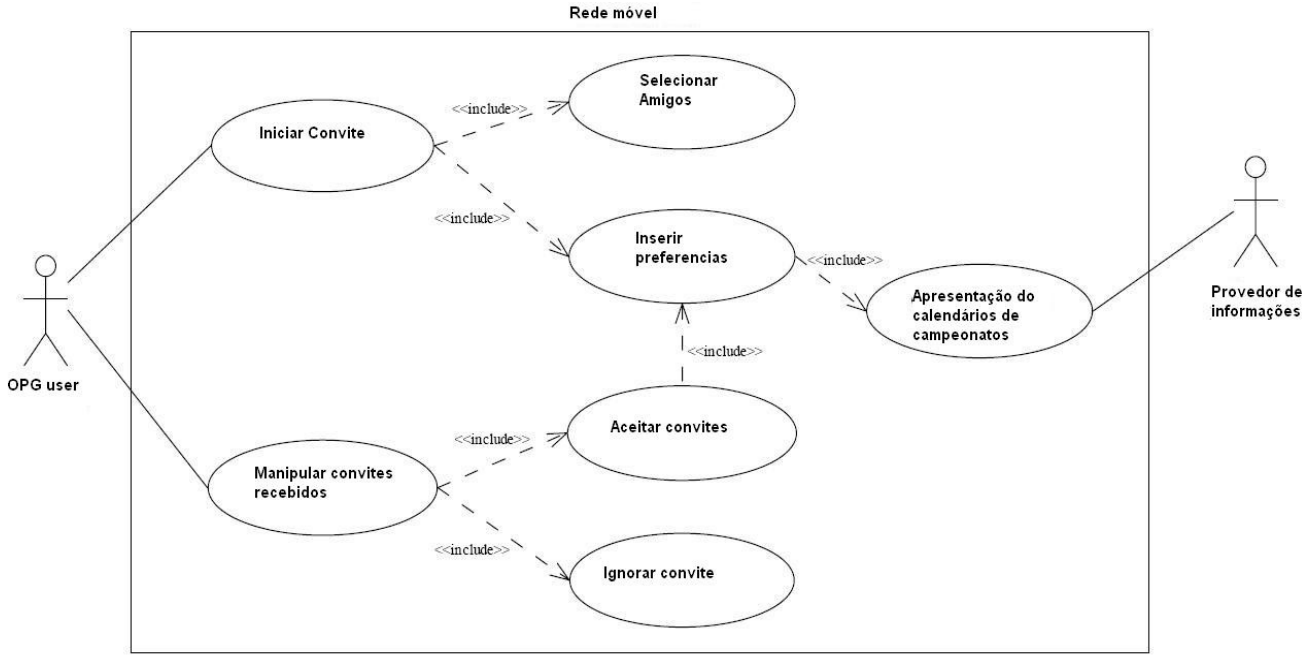

Figura 11 – OPG Caso de uso.

5.1.2 Identificação inicial dos tipos de agentes

Essa etapa envolve identificação dos principais tipos de agentes e subseqüente formação de um primeiro esboço do Diagrama de Agentes. As seguintes regras devem ser aplicadas nesse passo:

- Adicionar um tipo de agente por usuário/dispositivos.
- Adicionar um tipo de agente por recursos (que incluem software legado).

Aplicando as regras acima ao caso de uso do OPG, o seguinte diagrama é obtido (fig 12).

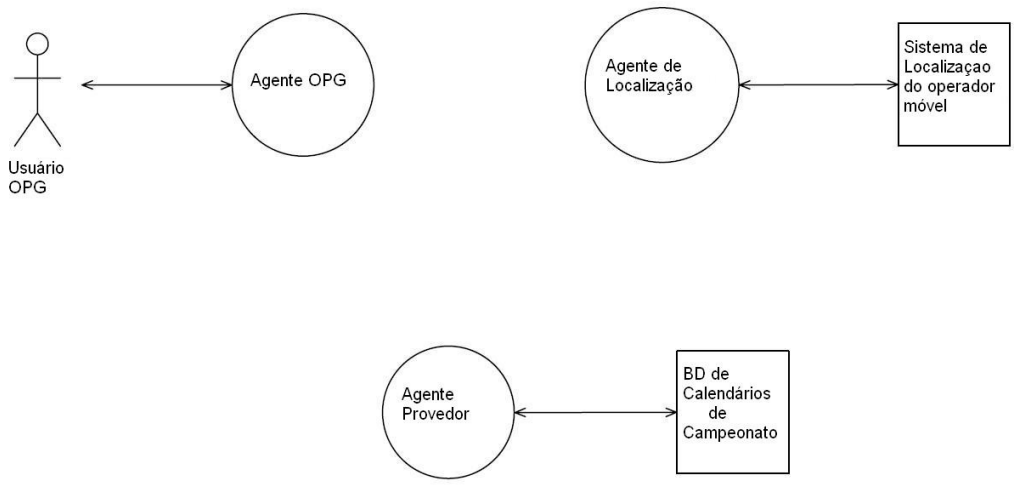

Figura 12 – diagrama de agente

O diagrama de agente é um dos principais artefatos produzidos na fase de análise e seu progressivo refinamento nas seguintes etapas. Com referência à figura acima, o diagrama de agentes inclui os seguintes 4 tipos de elemento:

1. Tipos de agentes: o atual tipo de agentes, representado por círculos.
2. Humanos: pessoas que devem interagir com o sistema, representados pelo símbolo de notação de ator em UML.
3. Recursos: sistemas externos que devem interagir com o sistema, representado por retângulos.
4. Conhecimentos: representado por flechas ligando instâncias dos elementos acima. Nesse momento, somente conhecimentos entre agentes e recursos/humanos é mostrado no diagrama.

Percebe-se que no diagrama de agente, diferente do diagrama de caso de uso de UML, é feita uma distinção entre humanos e sistemas externos.

Deve-se levar em conta a forma de como um sistema externo interage com o sistema legado. Existem três técnicas que ajudam nesse passo (Genesereth et al, 1994):

1. O uso de um agente tradutor.
2. A inserção de um wrapper.
3. Reescrever o código.

5.1.3 Identificação de responsabilidades

Nesse passo, para cada tipo de agente definido, uma lista inicial é feita de suas principais responsabilidades de uma forma intuitiva e informal. O artefato resultado desse é a tabela de responsabilidades.

As seguintes regras devem ser aplicadas nesse passo:

- Derivar o conjunto de responsabilidades inicial do caso de uso inicial.
- Considerar primeiro os agentes que têm suas responsabilidades mais claras e depois tentar identificar as responsabilidades de outros agentes.

Seguindo os passos acima, a seguinte tabela inicial de responsabilidades é produzida.

Tipos de Agente	**Responsabilidades**
OPG agente	Recebe requisições para iniciar convites dos usuários. Deixa os usuários selecionar amigos para convidar Apresenta calendários de competições Responde convites de outros OPG agentes

Tabela 1 – Tabela de responsabilidade

Muitas metodologias existentes tais como Gaia (Wooldridge et al, 2000) e MESSAGE (Caire et al, 2000) propõe uma abordagem diferente onde regras atômicas (equivalentes a responsabilidades definidas nesse passo) são identificadas inicialmente e depois ligado as tipos de agentes. Entretanto, essa abordagem é considerada menos intuitiva porque em alguns casos pode se tornar difícil determinar como que as regras devem ser agregadas dentro dos tipos de agentes, ex. quantos tipos de agentes devem existir e quais devem cobrir quais regras atômicas. A definição de responsabilidades como proposto remove essas ambigüidades.

5.1.4 Identificação conhecimentos

Essa etapa é focada em identificar quem precisa interagir com quem e, assim, o diagrama de agente é atualizado adicionando-se nele as relações de conhecimento, conectando os agentes que precisam ter uma ou mais interações. O termo "conhecimento" vem da metodologia Gaia, e é usado com algum sentido nessa metodologia proposta.

Uma obvia relação de conhecimento no OPG é requerido entre os diferentes agentes OPG: os que convidam e os convidados. E, desde que os agentes OPG devem apresentar os calendários das competições para seus usuários, e essas informações estão gravadas no banco de dados do provedor e são liberados pelos agentes provedores, haverá certamente uma relação de conhecimento entre os agentes OPG e os agentes provedores. Assim, voltando ao passo anterior, algumas novas responsabilidades podem ser adicionadas para o agente OPG e para o agente provedor. Para o agente OPG são essas:

- Apresenta (para os usuários) convites recebidos de outros agentes OPG.
- Deixa o usuário aceitar um convite recebido.
- Deixa o usuário rejeitar um convite recebido.
- Recupera o calendário de competições de um agente provedor relevante.

Para o agente provedor, as novas responsabilidades são:

- Responder a requisições dos agentes OPG.

Assim, o diagrama de agente é atualizado como na figura 13, e a tabela de responsabilidade também. Note que não há distinção entre conhecimentos e responsabilidades, assim todos os conhecimentos são colocados na tabela de responsabilidades e não separadamente.

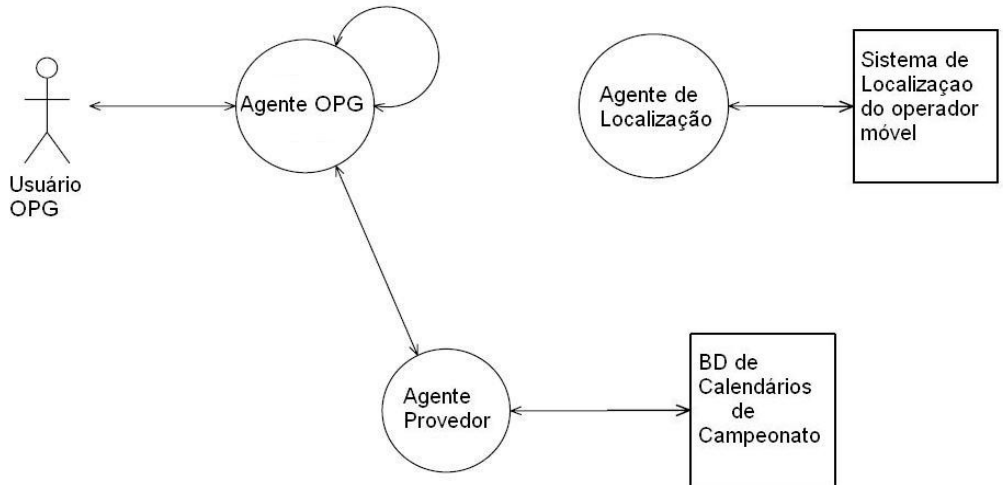

Figura 13 – Diagrama de agente atualizado.

Tipos de Agente	Responsabilidades
Agente OPG	Serve de requisições para iniciar convites dos usuários. Deixa os usuários selecionar amigos para convidar Apresenta calendários de competições Apresenta (para os usuários) convites recebidos de outros agentes OPG. Deixa o usuário aceitar um convite recebido. Deixa o usuário rejeitar um convite recebido.
Agente provedor	Responder a requisições dos agentes OPG.

Tabela 2 – Tabela de responsabilidades atualizada.

5.1.5 Refinamento de agentes

Nesse passo, o conjunto de tipos de agentes identificados inicialmente é refinado aplicando algumas considerações:

- **Suporte:** Quais informações de suporte o agente precisa para completar suas responsabilidades, e como, quando e onde a informação é gravada.

- **Descoberta:** Como os agentes ligados por conhecimento se descobrem.
  Em casos simples, a descoberta de agentes pode ser feita utilizando propriedades de convenção de nomes. Porém essa abordagem trás algumas limitações:

- Nomes de agentes devem ser únicos, globalmente.
- Agentes que estão envolvidos em uma interação devem se conhecer antes.
- Assumir convenção não é extensivo.
- Convenção de nomes não pode ser aplicada quando diferentes usuários começam seus próprios agentes escolhendo os nomes. Em tais casos, não existe garantia que os nomes serão únicos globalmente.

A melhor forma de resolver esse problema de descoberta entre agentes é adotando o mecanismo de páginas amarelas. Isso permite descobrir agentes pelas suas características bases, como os serviços que eles fornecem.

JADE fornece um mecanismo de *yellow page*s excelente.

- **Gerenciamento e monitoramento:** É o requerimento do sistema para manter o rastro de agentes existentes, ou iniciar e parar agentes em demanda.

Assim conseguimos a seguinte tabela de responsabilidades.

Tipo de Agente	**Responsabilidades**
Agente OPG	Serve de requisições para iniciar convites dos usuários. Deixa os usuários selecionar amigos para convidar Apresenta calendários de competições Apresenta (para os usuários) convites recebidos de outros agentes OPG. Deixa o usuário aceitar um convite recebido. Deixa o usuário rejeitar um convite recebido.

	Recuperar calendário de campeonatos de um agente provedor. Recuperar a cidade corrente do agente localização. Recuperar o agente provedor do agente de paginas amarelas.
Agente Provedor	Responder a requisições dos agentes OPG. Registrar com o agente de paginas amarelas.
Agente Localização	Responder a requisições dos agentes OPG Notificar a mudança de cidade para os agentes OPG

Tabela 3 – Tabela de responsabilidade para OPG atualizado.

5.1.6 Organização de informação dos agentes

Outro artifício que pode ser útil é o diagrama de instalação de agente, onde os dispositivos físicos de agentes serão organizados (referenciados como domino em algumas metodologias). O diagrama para o cenário do OPG é mostrado na figura 14 abaixo.

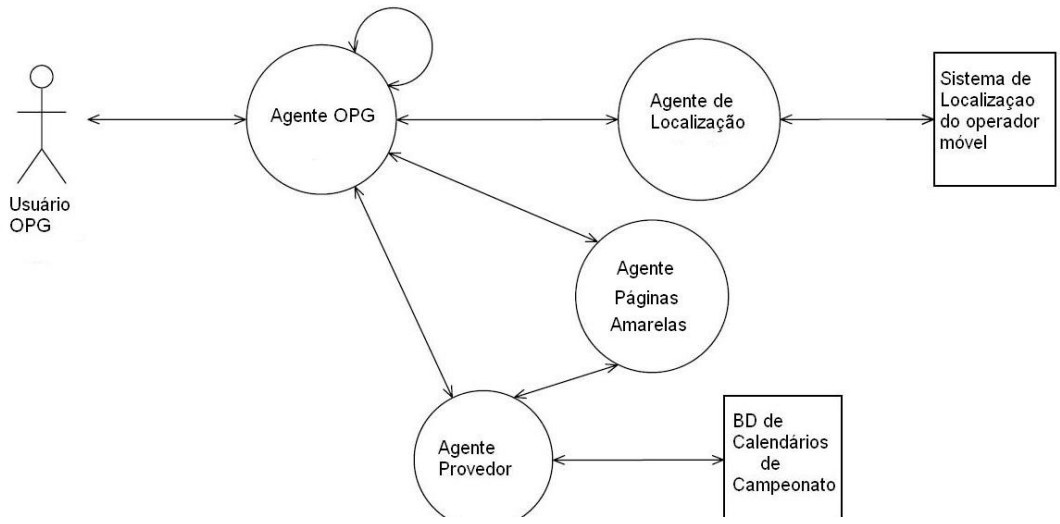

Figura 14 – Diagrama de agentes refinado.

Deve-se notar que o diagrama não tem a intenção de dar qualquer informação detalhada a respeito da organização, diferente do diagrama de instalação em UML, onde detalhes tais como o modo de comunicação entre nós são dados. A única proposta do diagrama é deixar explicito os requerimentos básicos de organização que são referenciados durante a parte de design e quando aplicado considerações tais como quebra ou fusão de agentes ou quando considerar comunicação eficiente.

Figura 15 – Diagrama de Instalação de Agentes.

Os artefatos produzidos após os passos da análise formam a base para a fase de design (projeto). Os artefatos produzidos em cada etapa e suas relações são sumarizados na figura 16.

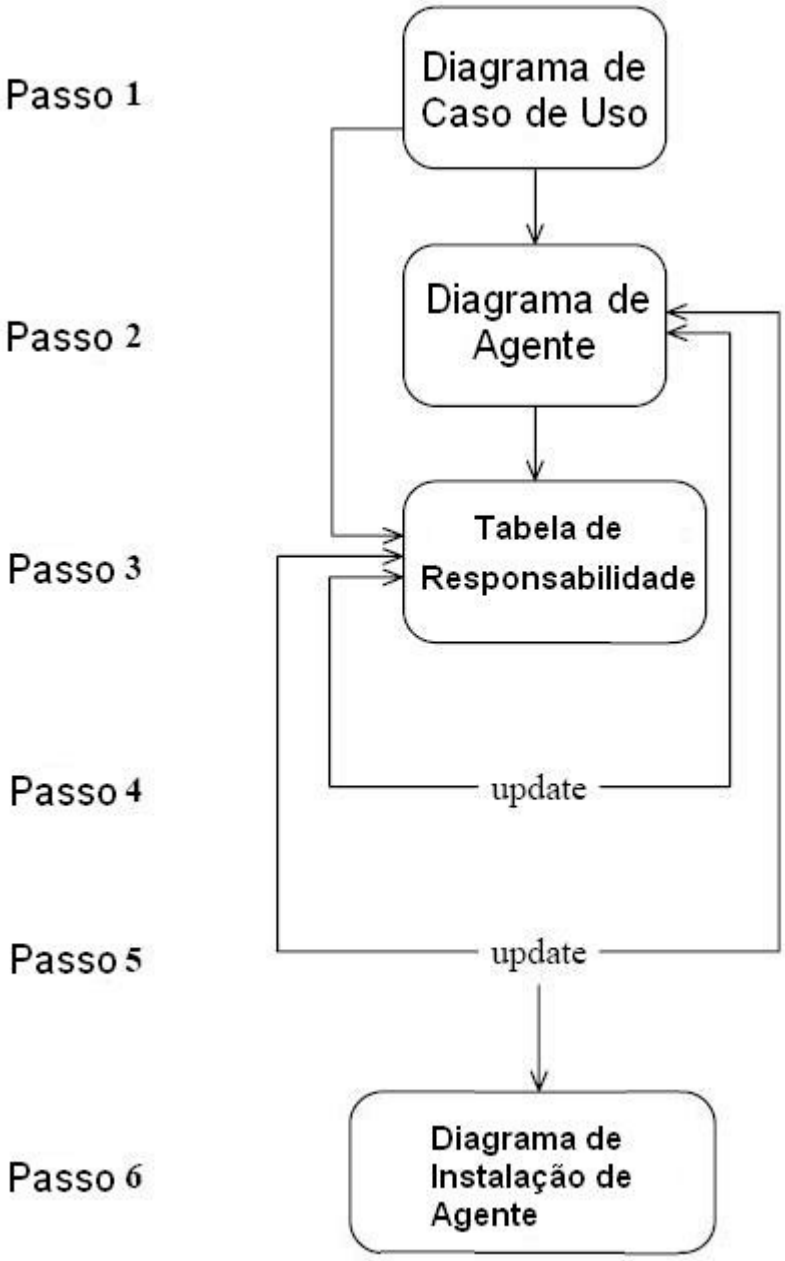

Figura 16 – Fases da Análise

5.2 Projeto

Uma vez que o problema tenha ficado claro e com um nível suficiente de detalhe, podemos passar da fase de análise para a fase de projeto, que visa especificar a solução com maior nível de detalhe. Não existem grandes barreiras entre essas duas fases, e a iteração entre elas é necessária. Sempre é possível e recomendado voltar para a fase de análise para revisar o que for preciso. Como essa metodologia se propõe a utilizar a plataforma JADE, dessa fase, de design, em diante a metodologia fica atrelada

a desenvolvimentos com a plataforma JADE. Qualquer tentativa de utilizar essa metodologia com outra plataforma teria que passar por adaptações na fase de design.

A fase de design nos permitirá chegarmos a um nível de detalhe que é bastante suficiente para uma relativa transição direta para a implementação de código, com uma significante possibilidade de gerar parte do código fonte automaticamente por ferramentas.

Similar a fase de análise, a fase de projeto possui um numero de passos lógicos a ser seguido, com algum grau de sobreposição. Alguns passos podem ser pulados, outros podem ser adicionados, dependendo do interesse do projetista. Para continuar a modelagem do OPG iremos utilizar os seguintes passos da fase de Design.

5.2.1 Passo 1: Dividindo/Unindo/renomeando Agentes

Esse passo envolve observar os artefatos produzidos e determinar quais dos agentes produzidos no diagrama de agentes devem ser fundido ou separados. Esse passo deve ser considerado importante, já que tem um efeito direto na eficiência e na complexidade do sistema. Baseado nisso as seguintes regras devem ser aplicadas nesse passo:

- **Duplicação de dados deve ser evitada:** Se existem dois ou mais agentes que compartilham a maioria das informações em comum requeridas para a conclusão de suas próprias tarefas, deve-se cogitar a possibilidade de fundir esses agentes em um único agente.
- **Duplicação de código para acessar recursos deve ser evitada:** Se existe um ou mais agentes que precisam acessar um mesmo recurso, deve-se considerar a possibilidade de fundi-los num só.
- **Evitar dividir agentes, a menos que se tenha uma boa razão para fazer isso:** Isso pode aumentar a complexidade do sistema em toda parte e diminuir a eficiência do sistema, já que desnecessária comunicação entre os agentes poderá acontecer.
- **Cada agente está situado em uma única máquina:** Se duas peças de funcionalidades devam ser fornecida em diferentes maquinas, então essas peças de funcionalidades devem ser fornecidas por diferentes agentes.
- **Evitar que agentes tornem-se muito complexos:** Isso os torna difícil de projetar e dar manutenção.

Tendo em vista essas cinco regras, pode parecer que exista um contraste entre elas. Entretanto, o que temos que levar em consideração é o balanço entre um excessivo número de agentes simples e um pequeno número de agentes complexos.

No caso do OPG temos um número consideravelmente pequeno de agentes, então não considera-se necessário quaisquer divisões ou absorção dos agentes existentes.

5.2.2 Passo 2: Especificação das interações.

Nessa etapa, para cada tipo de agente, todas as responsabilidades relacionadas com relações de conhecimento entre outros agentes (baseado na tabela de responsabilidade produzido na fase de análise) são levados em conta e uma tabela de interação é produzida para cada tipo de agente. Cada linha na tabela representará uma interação que será incluída:

- A descrição da interação.
- A responsabilidade (da tabela de responsabilidade) que originou a interação. Isso liga artefatos da fase de análise com a fase de design podendo ser usado mais tarde para checar inconsistências.
- Um protocolo de interação escolhido para implementar a interação. O padrão de protocolo de interação FIPA deve ser considerado primeiro. Se não for suficiente, um protocolo ad-hoc deve ser definido.
- O papel do agente no protocolo de interação (IP). Podendo ser I para inicial ou R de resposta. Outros papéis podem ser adicionais se necessário.
- O nome e o tipo do agente caso sejam relevantes.
- O disparador da consideração, ex. quando a interação inicia. Essa consideração deve ser descrita informalmente.

Interação	**Linha da tabela de responsabilidade**	**IP**	**Papel**	**Com**	**Quando**
Convidar outros usuários	1	Contract Net	I	OPG agente	O usuário inicia um convite
Responder a um	5	Contract Net	R	OPG agente	Um convite é

convite					recebido
Recurepar calendário de competições	8	Requisição FIPA	I	Agente provedor da cidade atual	-Na inicialização - É detectado mudança de cidade
Recuperar info. da cidade atual	9	Requisição FIPA	I	Agente de localização	Sempre
Recuperar o agente provedor de uma dada cidade	10	Requisição FIPA	I	Agente de paginas amarelas	-Na inicialização - É detectado mudança de cidade

Tabela 4 – Tabela de interação para o OPG após passo 2 de design.

Podemos começar mapeando nosso agente de paginas amarelas para o *directory facilitator agent* existente na plataforma JADE.

5.2.3 Passo 3: Modelos de mensagem.

Todos os papéis de protocolos de interação, identificados no passo anterior, são implementados como comportamentos dos agentes do framework JADE. Nessa etapa, objetos *MessageTemplate* mais apropriados são especificados para serem usados nesses comportamentos para receber mensagens, e a esses modelos são adicionados a linhas da tabela de interação. As seguintes regras devem ser aplicadas nesse passo:

- Use *MenssageTemplates* baseado no ID de conversação no iniciador de papeis de comportamento (assegurando que é gerado um único ID por agente).
- Analisar conflitos e modificar as *MessageTemplate*s usadas nos comportamentos de resposta.
- Se conflitos não podem ser resolvidos, deve-se considerar aplicar padrões de modelos dinâmicos.

Interação	**Resp.**	**IP**	**Papel**	**Com**	**Quando**	**Modelo**

Convidar outros usuários	1	Contract Net	I	OPG agente	O usuário inicia um convite	Conv-id
Responder a um convite	5	Contract Net	R	OPG agente	Um convite é recebido	Perf = CFP
.....						

Tabela 5 – Tabela de interação após passo 3 do design.

5.2.4 Passo 4: Descrições para ser registrado e encontrado (Páginas Amarelas).

Nesse passo, a conversão de nomes adotada e a descrição de serviços dos agentes no catálogo de páginas amarelas mantido pelo *directory facilitator* do Jade são formalizados.

Convenção de nomes é dependente de domínio. É sugerido o uso de linguagem natural para especificá-los.

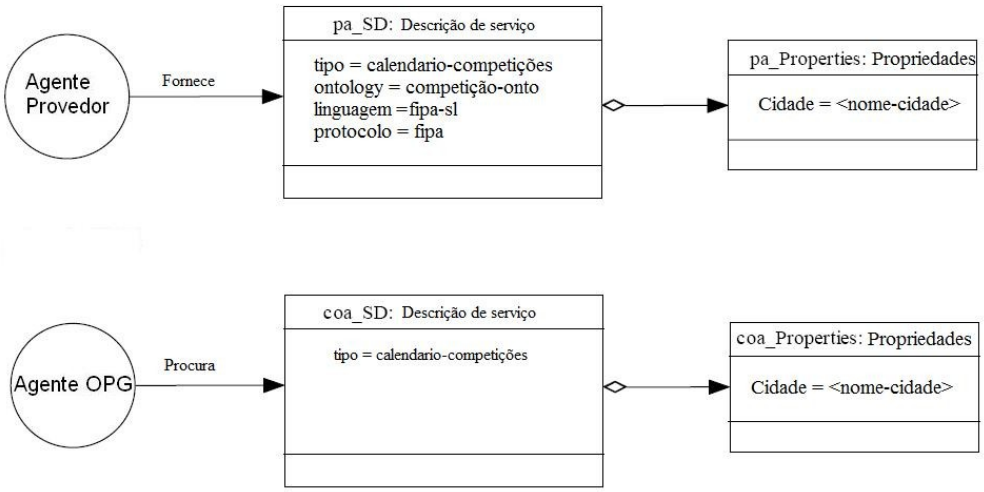

Figura 17 – Registro de serviço no OPG.

5.2.5 Passo 5: Interações Agente-Recurso

É freqüente o caso de um ou mais agentes no sistema que devem interagir com recursos externos tais como banco de dados, arquivos de informações ou software legado. Essa interações foram identificadas no passo 2 da análise e expressado no diagrama de Agentes em relações de conhecimento com um elemento. Tais recursos podem ser classificados em duas principais categorias:

- Recursos passivos: Recursos que mudam seus estados somente como conseqüência de estímulos do agente que controla o recurso.
- Recursos ativos: Recursos que podem modificar seus estados independentemente do agente que o controla.

5.2.5.1 Recursos Passivos

Banco de dados inteiramente controlado pela interação de agentes, um arquivo no sistema, ou uma biblioteca C que fornece funções computacionais são exemplos de recursos passivos.

Agentes JADE são pedaços de código Java, assim técnicas padrões da tecnologia Java, são usados para acessar esse tipo de recurso. Para acessar um banco de dados podemos usar o JDBC, no caso de arquivos usamos as classes do pacote Java.io, e no caso de bibliotecas C poderíamos usar o JNI.

5.2.5.2 Recursos Ativos

Um banco de dados que pode ser acessado manualmente por um usuário, ou que é acessado por outro programa, um arquivo de log que é preenchido continuamente por um programa externo são exemplos de recursos ativos.

Recursos ativos podem fornecem interfaces baseada em mudanças, assim o agente controlador pode detectar mudanças imediatamente. Em outros casos os recursos podem fornecer interfaces com métodos que bloqueiam até que uma mudança seja detectada. Porém, em certos casos, a única forma de detectar mudanças relevantes nos recursos ativos é consultá-los periodicamente.

No OPG o agente provedor deve acessar o banco de dados com as informações do calendário de competições e outras estatísticas. Devemos tomar mais atenção para esse caso de acesso a um banco de dados relacional. Fazer consultas a um banco de dados é simples utilizando a linguagem SQL e qualquer lugar do código, porém o

agente controlador deve possuir acesso ao banco. Isso evita que haja código SQL em toda parte do sistema. Se um agente necessita de informação da base de dados ele deve fazer uma requisição ao agente provedor que então faz o acesso ao banco e retorna ao agente cliente o dado requisitado.

5.2.6 Passo 6: Interação Usuário-Agente

Em alguns casos, um agente precisa interagir com o usuário. Em nosso OPG agentes que interagem com o usuário foram definidos na etapa 2 da fase de análise. Em nosso sistema, o usuário deve interagir a partir de seu celular virtual.

A implementação é feita utilizando o *framework* gráfico já utilizado nos jogos MMORPG.

5.2.7 Passo 7: Comportamento interno dos agentes

O trabalho que um agente tem q fazer está carregado dentro dos comportamentos dos agentes (behaviours) (*Bellifemine et al,* 2007*).* Nessa etapa devemos olhar a tabela de responsabilidade criada na etapa de analise e mapear as responsabilidades para os comportamentos dos agentes.

Em Jade utilizamos as seguintes classes que representam tipos diferentes de comportamentos:

- *OneShotBehaviour:* implementa tarefas que são executados uma única vez e terminada imediatamente.
- *CyclicBehaviour*: implementa tarefas que estão sempre ativas, e realiza sempre as mesmas operações toda vez que é executada.
- *TickerBehaviour:* implementa uma tarefa que executa periodicamente a mesma operação.
- *WakerBehaviour:* implementa uma tarefa que executa uma vez após certa quantidade de tempo, e então finaliza.

No momento encontramos comportamentos complexos, preferimos quebrá-los em vários outros comportamentos mais simples e então adotamos uma das classes de composição de comportamentos que o JADE disponibiliza:

- *SequentialBehavior:* implementa composições de tarefas que lista suas sub-tarefas seqüencialmente.
- *FSMBehaviour:* implementa composições de tarefas que lista suas tarefas de acordo com uma Maquina de Estados Finitos.

Para demonstrar um exemplo da implementação de responsabilidades complexas, pegue, por exemplo, a responsabilidade “deixar o usuário selecionar amigos para uma partida”. Essa responsabilidade foi modelada como uma Máquina de Estados Finitos, e é mostrado pelo diagrama de transição de estado na figura 17 abaixo.

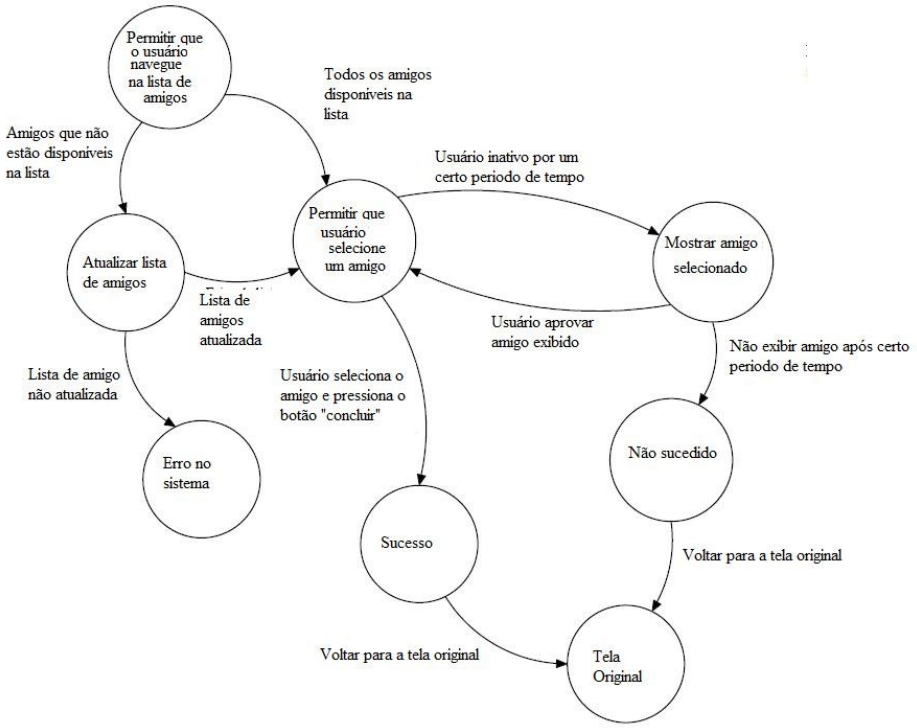

Figura 18 - Diagrama de transição de estado

5.3 Pós Design

Após a etapa de design é possível ir direto para a implementação de código. Agora, após as etapas de modelagem, tem-se uma idéia mais abrangente de como o sistema deve ser implementado usando a plataforma JADE.

CONCLUSÃO

Essa trabalho apresenta tópicos relacionado a modelagem de sistemas multi-agentes, comparação entre metodologias orientadas a objetos e as metodologias baseada em agentes, e a dificuldade de seguir uma metodologia que não tenha sido baseada em engenharia de software orientada a objetos.

Agente é uma forma de abstração diferente de Classe e Objeto, sendo assim, é necessário desacoplar a relação direta classe-agente. O trabalho apresenta as noções de agentes fraco e fortes dada por Wooldridge e Shoham, as arquiteturas existentes de agentes, sistemas multi agentes e a programação orientada a agentes.

O trabalho bruni algumas das metodologias existentes de modelagem de sistemas baseado em agentes e multi-agentes. A partir daí, é apresentado o modelo de aplicativo OPG que é utilizado como estudo de caso da metodologia proposta baseada nas existentes, evitando se basear em metodologias orientadas a objetos.

Um olhar mais delicado sobre a tecnologia JADE e sua arquitetura que traz ferramentas para deixar de lado o problema infra-estrutural do projeto e se focar na lógica de negócio do desenvolvimento por agente.

Após seguir todo o trajeto da Análise e do Projeto do sistema OPG resultando em um projeto com informações que facilita a implementação.

O estudo das metodologias traz mais experiência para ser capaz de criticar/louvar erros/acertos de projetos de software.

Idéias para futuros trabalhos incluem o desenvolvimento de ferramentas de automação de código, obtendo a partir da modelagem das fases de análise e projeto do sistema transformando em código fonte automaticamente. Esse tipo de ferramenta será muito comum num futuro próximo, com um potencial para ser abordado e pesquisado atualmente.

REFERÊNCIAS BIBLIOGRÁFICAS

Bauer, B. (2002). UML class diagrams revisited in the context of agent-based systems. Lecture Notes in Computer Science, 2222:101–110

Bauer, B., Müller, J. P., and Odell, J. (2001). Agent UML: A formalism for specifying multiagent interaction. In Agent-Oriented Software Engineering, pages 91–103. Springer-Verlag.

Bellifemine, Caire and Greenwood. Developing Multi Agent Systems with JADE. John Wiley & Sons. Apr.2007

Caire, Coulier, Garijo, Gomez, Pavon, Leal, Chainho,. Kearney, Stark, Evans, and Massonet, "Agent Oriented Analysis Using Message/UML," *Lecture Notes in Computer Science*, M. Wooldridge, G. Weiss, and P. Ciancarini (Eds.), Springer-Verlag, vol. 2222, 2002, pp. 119-135.

Coleman et al, D. (1994). Object Oriented Development: The Fusion Method. Prentice-Hall.

Collinot, Drogoul, and Benhamou, "Agent oriented design of a soccer robot team," in *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96)*, Kyoto, Japan, 1996, pp. 41–47.

Dennis and Wixom, *Systems Analysis and Design: An Applied Approach*, John Wiley and Sons, 2000.

Ferguson, I.A. Towards an Architecture for Adaptive, Rational, Mobile Agents. InWerner, E. and Demazeau, Y. (eds), *Decentralized AI 3 – Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent World*, pp. 249–262, Elsevier, Amsterdam, The Netherlands, 1991.

Free Software Foundation, Inc. GNU LESSER GENERAL PUBLIC LICENSE, Version 3, 29 June 2007 www.gnu.org/licenses/lgpl.html

Genesereth and S. P. and Ketchpel, "Software Agents," *Communication of the ACM*, vol. 37(7), 1994.

Hampton, Martin, Pitt, and Ottinger, A Critique of Use Cases, 9 July 1997, veja em: http://ootips.org/use-cases-critique.html.

Jade website, http://jade.cselt.it/ acessado em 2008

Kendall, E. A. (2001). Agent software engineering with role modelling. In First international workshop, AOSE 2000 on Agent-oriented software engineering, pages 163–169, Secaucus, NJ, USA. Springer-Verlag New York, Inc.

Luck, McBurney, and Preist. "Agent Technology: Enabling Next Generation Computing," AgentLink, 2003.

MESSAGE: Methodology for Engineering Systems of Software Agents – Initial Methodology. EURESCOM Participants in Project P907-GI (2000)

Muller, J.P., Pischel, M. and Thiel, M. Modelling Reactive Behaviour in Vertically Layered Agent Architectures.
In Wooldridge, M. and Jennings, N.R. (eds), *Intelligent Agents: Theories, Architectures, and Languages*
(LNAI 890), pp. 261–276, Springer-Verlag, Heidelberg, 1995.

Odell, "Objects and agents: how do they differ?," *Journal of Object-Oriented Programming*, October 2000.

Pollack and Ringuette Pollack,M.E. and Ringuette,M. Introducing the tile world: Experimentally evaluating agent architectures. In National Conference on Artificial Intelligence.

Shoham, "Agent Oriented Programming," *Artificial Intelligence*, vol. 60(1), pp. 51-92, 1993.

Wooldridge, Jennings, and Kinny, “The gaia methodology for agent-oriented analysis and design,” *Autonomous Agents and Multi-Agent Systems*, vol. 3(3), pp. 285-312, 2000.

Wooldridge and Jennings, “Intelligent agents: theory and practice,” *The Knowledge Engineering Review*, vol. 10(2), pp. 115-152, 1995.

Wood, M. F. and DeLoach, S. (2000). An overview of the multiagent systems engineering methodology. In Agent-Oriented Software Engineering – Proceedings of the First International Workshop on Agent-Oriented Software Engineering, 10th June 2000, Limerick, Ireland. P. Ciancarini, M. Wooldridge, (Eds.) Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January 2001. , p. 207–222.