Engenharia de Software baseada em Agentes

Rosario Girardi
Universidade Federal do Maranhão - DEINF - GESEC
Av. dos Portugueses, s/n, Campus do Bacanga
CEP 65085-580 São Luís - MA - Brasil
rgirardi@deinf.ufma.br

RESUMO

R. Girardi. 2004. Engenharia de Software baseada em Agentes. Congresso Brasileiro de Ciência da Computação, Itajaí, 2004, 913 – 937. Itajaí, SC – Brasil, ISSN 1677-2822

Apresenta-se um elenco de técnicas, metodologias e ferramentas para o desenvolvimento de software baseado em agentes. Descrevem-se as características deste paradigma de desenvolvimento e suas vantagens na construção de sistemas complexos. Discutem-se os modelos de desenvolvimento para a construção de aplicações multiagente e técnicas recentes para a análise de requisitos e projeto de aplicações baseadas em agentes. Apresentam-se as abordagens de desenvolvimento para e com reutilização neste paradigma. Discutem-se soluções de projeto comumente utilizadas na forma de um conjunto de padrões arquiteturais e de projeto detalhado. Apresenta-se uma visão geral dos ambientes de desenvolvimento disponíveis para a construção de sistemas multiagente.

PALAVRAS DE INDEXAÇÃO ADICIONAIS: *Sistemas Multiagente, Arquiteturas de Agentes, Métodos e Técnicas em Engenharia de Software, Padrões de software*

INTRODUÇÃO

A Engenharia de Software tradicional tem desenvolvido técnicas e ferramentas apropriadas para a construção de sistemas de computação com comportamento predeterminado, isto é, o software ira a executar exatamente aquilo que foi planejado, projetado e programado na construção do sistema. Qualquer situação não contemplada nessas instâncias pode provocar falhas no sistema e conseqüentemente perdas tanto no nível econômico como humano.

A Engenharia de software baseada em agentes vem fornecer soluções para abordar a crescente complexidade dos sistemas de computação que geralmente devem operar em ambientes não predizíeis, abertos e que mudam rapidamente. Estes sistemas devem ser capazes de decidir por si mesmos o que fazer em qualquer situação para alcançar seus objetivos. Os sistemas multiagente são considerados uma excelente metáfora para caracterizar sistemas complexos e o conceito de agente como abstração de software é de grande utilidade para a compreensão, engenharia e uso desse tipo de sistemas "(GIRARDI, 2001)".

A sistematização de metodologias para o desenvolvimento baseado em agentes é um tópico de ativa pesquisa.

Este artigo apresenta um elenco de técnicas, metodologias e ferramentas para o desenvolvimento de software baseado em agentes. Descrevem-se as características deste paradigma de desenvolvimento e suas vantagens na construção de sistemas complexos. Discutem-se os modelos de desenvolvimento para a construção de aplicações multiagente e técnicas recentes para a análise de requisitos e projeto de aplicações baseadas em agentes. Apresentam-se as abordagens de desenvolvimento para e com reutilização neste paradigma. Discutem-se soluções de projeto comumente utilizadas na forma de um conjunto de padrões arquiteturais e de projeto detalhado. Finalmente, apresenta-se uma visão geral dos ambientes de desenvolvimento disponíveis para a construção de sistemas multiagente.

O PARADIGMA DE DESENVOLVIMENTO BASEADO EM AGENTES

A caracterização dos agentes de software, dos sistemas multiagente e do desenvolvimento baseado em agentes tem sido amplamente abordada na literatura “(BRADSHAW, 1997)” “(GENESERETH, 1997)” “(SYCARA, 1998”) “(JENNINGS, SYCARA E WOOLDRIDGE, 1998)” “(NWANA, 1996)” “(WOOLDRIDGE E JENNINGS, 1994)” “(WOOLDRIDGE, 2002)”.

Esta seção apresenta os conceitos básicos e arquiteturas de agentes e sistemas multiagente e o potencial deste novo paradigma para a construção de sistemas complexos.

A abstração agente

Um agente é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo utilizando-se dos executores. Por exemplo, nos agentes humanos, os olhos e ouvidos, são sensores; as mãos e a boca são executores. A 0 lustra as interações de um agente genérico com seu ambiente "(RUSSEL E NORVIG, 1995)".

Na construção de programas baseados em agentes, deve-se decidir como construir o mapeamento de percepções a ações.

A estrutura básica de um agente tem uma forma bem simples (Figura 2). Possui uma memória interna que irá atualizar-se com a chegada de novas percepções. Essa memória é utilizada nos procedimentos de tomada de decisão, os quais irão gerar ações para serem executadas. De forma a manter um histórico do comportamento do agente, a memória do agente é também atualizada a partir da ação selecionada.

A autonomia é a característica principal dos agentes. Os agentes são capazes de atuar sem intervenção humana ou de outros sistemas através do controle que eles possuem do seu estado interno e do seu comportamento.

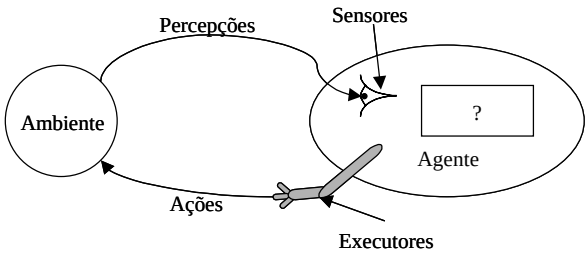

Figura 1 - As interações de um agente genérico com seu ambiente "(RUSSEL E NORVIG, 1995)"

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)"

Os conceitos previamente descritos constituem uma definição básica e de consenso do que é um agente. De acordo com esta definição, alguns exemplos de agentes são os sistemas de controle, desde um simples termostato até sistemas de controle de reatores nucleares. Um termostato possui um sensor para detectar a temperatura de uma sala. O sensor produz dois tipos de percepções: uma que indica que a temperatura está muito alta (ou muito baixa) e outra que indica que a temperatura está adequada. De acordo as percepções, o agente termostato tomara a decisão de executar ou não a ação de esquentar (esfriar).

Fora essa definição básica, existe uma enorme variedade de caracterizações da entidade agente, em particular, do conceito de *agente inteligente.*

"Para WOOLDRIDGE (1999)" um agente inteligente possui comportamento autônomo e *flexível* para alcançar seus objetivos, onde flexibilidade significa três coisas:

Reatividade – os agentes inteligentes percebem seu ambiente e respondem em tempo às mudanças que nele ocorrem para alcançar seus objetivos.

Proatividade - os agentes inteligentes são capazes de exibir comportamento orientado a objetivos tomando a iniciativa de forma a alcançar seus objetivos.

Habilidade social - os agentes inteligentes são capazes de interagir com outros agentes e, possivelmente, com humanos para alcançar seus objetivos.

As principais contribuições do paradigma de desenvolvimento baseado em agentes provêm da area da Inteligência Artificial. Esta disciplina propõe quarto abordagens principais para a construção de sistemas computacionais inteligentes "(RUSSEL E NORVIG, 1995)":

- a abordagem do teste de Turing (sistemas que atuam como humanos);
- a abordagem cognitiva (sistemas que pensam como humanos);
- a abordagem das leis do raciocínio (sistemas que pensam racionalmente) e,
- a abordagem dos agentes racionais (sistemas que atuam racionalmente)

Os sistemas que exibem comportamento racional caracterizam a abordagem de agentes para a construção de sistemas inteligentes. Atuar racionalmente significa atuar para alcançar metas considerando as próprias crenças.

As ações tomadas por um agente racional são supostamente as corretas, aquelas que causam que o agente seja o mais bem-sucedido. Uma medida de desempenho estabelece o critério que determina quanto bem sucedido um agente é.

"RUSSEL E NORVIG (1995)" caracterizam a racionalidade de um agente, em um momento dado, através dos seguintes aspectos:

- a medida de desempenho que define o grau de sucesso;
- o histórico completo das percepções do agente;
- o conhecimento que o agente possui do ambiente;
- as ações que o agente pode realizar

Um agente racional ideal deveria selecionar e executar aquela ação que é de esperar maximize sua medida de desempenho, considerando o histórico das percepções e o conhecimento que o agente possui.

A abordagem de agentes da Inteligência Artificial é mais geral que aquela baseada nas leis do raciocínio, onde a ênfase está na realização de inferências corretas. A realização de inferências às vezes caracteriza o comportamento de um agente porque uma das formas de se comportar racionalmente é raciocinar logicamente até a conclusão de que uma ação em particular permitirá atingir as metas do agente. Porém, realizar apenas inferências não é suficiente para exibir comportamento inteligente. Existem situações onde uma ação reflexiva simples pode ser mais efetiva que uma ação cuja decisão de execução é tomada mais lentamente através de algum mecanismo de inferência.

Agentes versus objetos

O paradigma de agentes surge como uma evolução natural da orientação a objetos. Como os objetos, a abstração agente possui uma memória e um comportamento. Porém, ela não é uma entidade passiva como os objetos tradicionais. A priori, um agente pode ser definido como um objeto ativo, autônomo, social e com capacidade de aprendizado "(GIRARDI, 2001)" "(HYACINTH, 1996)" "(WOOLDRIDGE, 1999)".

A autonomia, o controle do seu comportamento, é o principal aspecto que distingue objetos de agentes. Um objeto encapsula estado e comportamento, mais não pode pela sua iniciativa própria ativar o seu comportamento. Um objeto pode invocar métodos publicamente acessíveis de outros objetos. Uma vez que um método é invocado, as ações correspondentes são executadas. Neste sentido, os objetos não são autônomos porque são totalmente dependentes os uns dos outros para a execução de suas ações.

A autonomia de um agente é suportada tanto pelo conhecimento embutido no agente, no momento da sua criação, bem como pela experiência adquirida pelo agente nas suas interações com o ambiente no qual está inserido.

Os agentes estão continuamente ativos, na execução de um ciclo de observar seu ambiente, atualizar seu estado interno e selecionar e executar uma ação para executar. Em contraste, um objeto é passivo a maioria do tempo, só ficando ativo quando outro objeto requer seu serviço invocando um dos seus métodos.

Tipos de agentes

Na construção de programas que utilizam agentes, deve-se decidir como construir o mapeamento de percepções a ações. Esse mapeamento determina a arquitetura interna do agente.

Existem diferentes propostas de classificação dos tipos de agentes. A seguir são apresentadas as propostas de "RUSSEL E NORVIG (1995)" "WOOLDRIDGE (1999)".

Classificação de Russel e Norvig

"Segundo RUSSEL E NORVIG (1995)", existem quatro tipos diferentes de agentes, possuindo cada um aspectos diferenciados na busca da solução de problemas, são eles:

- Agentes reflexivos
- Agentes reflexivos que mantém registro do ambiente
- Agentes baseados em metas
- Agentes baseados na utilidade

Nos agentes reflexivos, cada percepção dispara alguma ação pré-estabelecida no programa, sendo esta relação uma simples regra do tipo <condição, ação> onde a condição é dada pela percepção do agente. Por exemplo: "se o carro da frente freou então começar a frear". Este agente faz uma simulação dos reflexos natos dos humanos. A Figura 3 mostra a estrutura de um agente reflexivo, onde se percebe o mapeamento da percepção para a ação.

função Agente-Reflexivo-Simples (*percepção*)
retorna *ação*
estática: *regras* {conjunto de regras do tipo condição-ação}

estado ← Interpretar-entrada (*percepção*)
regra ← Casamento-de-regra (*estado*,*regras*)
ação ← Obter-ação (*regra*)
retornar *acão*

Figura 3 - A estrutura de um agente reflexivo simples "(RUSSEL E NORVIG, 1995)"

Nos agentes reflexivos que mantém registro do ambiente, o estado interno do agente é atualizado, ou seja, ocorre um registro da seqüência perceptual e das ações executadas pelo agente.

A estrutura deste tipo de agente mostra como a percepção corrente combinada com o estado interno antigo gera a atualização do estado corrente. A parte mais interessante é a função Atualizar-estado, que é responsável pela criação do novo estado interno (Figura 4).

função Agente-reflexivo-com-estado (*percepção*)
retorna *ação*
estática: *estado* {descrição do estado atual do ambiente}
regras {conjunto de regras do tipo condição-ação}

estado ← Atualizar-estado (*estado*, *percepção*)
regra ← Casamento-de-regra (*estado*,*regras*)
ação ← Obter-ação (*regra*)
estado ← Atualizar-estado (*estado*, *ação*)
retornar *ação*

Figura 4 - A estrutura de um agente reflexivo com estado "(RUSSEL E NORVIG, 1995)"

Os agentes baseados em metas, além de manterem um registro do estado do ambiente, possuem uma meta que descreve um estado desejável a ser atingido. Então, o agente pode fazer uma combinação do desejável com os resultados de possíveis ações, para tomar decisões na busca da meta. Alguma das vezes isto pode ser feito de forma simples, a meta é alcançada simplesmente com uma ação, outras vezes pode ser mais complicado, pode ser necessário longas seqüências de ações para alcançar-se a meta (0).

O último tipo de agente é o baseado na utilidade (Figura 6). Apesar da importância das metas, elas sozinhas não representam o bastante para se ter um comportamento de qualidade "(RUSSEL E NORVIG, 1995)". No agente baseado em metas só há uma preocupação entre os estados "feliz" e o "infeliz", enquanto uma medida de utilidade mais geral deve permitir uma comparação entre vários diferentes estados, de acordo com o quão feliz se busca ser.

função Agente-baseado-em-meta (*percepção*) retorna *ação*
estática: *estado* {descrição do estado atual do ambiente}
meta {meta do agente, inicialmente não definida}
seq_ações {seqüência de ações, inicialmente não definida}

estado ← Atualizar-estado (*estado*, *percepção*)
se *meta* não está definida então
meta ← Definir_meta (*estado*)
seq_ações ← Definir_seqüência_ações (*meta*)
ação ← Obter-ação (*seq_ações*)
estado ← Atualizar-estado (*estado*, *ação*)
retornar *ação*

Figura 5 - A estrutura de um agente baseado em meta

função Agente-baseado-em-utilidade (*percepção*) retorna *ação*
estática: *estado* {descrição do estado atual do ambiente}
utilidade {medida de utilidade do agente}

estado ← Atualizar-Estado (*estado*, *percepção*)
ação ← Obter-ação (*estado*, *utilidade*)
estado ← Atualizar-Estado (*estado*, *ação*)
retornar *ação*

Figura 6 - A estrutura de um agente baseado na utilidade

Classificação de Wooldridge

"WOOLDRIDGE (1999)" classifica as arquiteturas dos agentes em quatro tipos:

- Arquiteturas baseadas em lógica
- Arquiteturas reativas
- Arquiteturas em camadas

Arquiteturas de crença-desejo-intenção (BDI)

Arquiteturas baseadas em lógica

Nas arquiteturas baseadas em lógica os agentes contêm um modelo simbólico do ambiente do agente, explicitamente representado em uma base de conhecimento e a decisão da ação a executar é tomada a através de raciocínio lógico. Esta arquitetura é utilizada nos agentes baseados em metas e agentes baseados na utilidade na classificação de "RUSSEL E NORVIG (1995)".

Arquiteturas reativas

As arquiteturas reativas são aquelas que não possuem conhecimento do ambiente do agente e não utilizam raciocínio lógico. Elas se baseiam na suposição de que um agente pode desenvolver inteligência a partir de percepções e interações com seu ambiente, não necessitando de um conhecimento do mesmo. A decisão da ação a executar é tomada através de um mapeamento direto da situação para a ação. Esta arquitetura é utilizada nos agentes reflexivos na classificação de "RUSSEL E NORVIG (1995)".

Arquiteturas em camadas

Nas arquiteturas em camadas a decisão da ação a executar é feita via várias camadas de software, onde cada camada raciocina

sobre o ambiente em diferentes níveis de abstração. Esta abordagem, também conhecida como arquitetura híbrida, estrutura um agente em dois subsistemas: um deliberativo, contendo um modelo simbólico do ambiente que desenvolve planos e toma decisões via raciocínio lógico e um reativo, capaz de reagir a eventos que ocorrem no ambiente sem nenhum tipo de raciocínio.

O objetivo da arquitetura de agentes híbridos é a combinação das virtudes, supostamente existentes, de ambas as abordagens anteriores: a abordagem simbólica que permite o desenvolvimento de planos dinâmicos e a tomada de decisões sobre estes e a abordagem reativa, baseada na rápida reação a eventos sem a necessidade de manipulação de estruturas simbólicas complexas, o que apresenta uma maior adaptabilidade, robustez e desempenho.

Pode-se identificar dois tipos de fluxo de controle nas arquiteturas de camadas (Figura 7):

- *Horizontal* – cada camada de software está diretamente conectada ao sensor de entrada e a ação de saída (Figura 7a). De fato, cada camada age como um agente, produzindo sugestões sobre que ação executar;
- *Vertical* – o sensor de entrada e a ação de saída estão localizados no máximo em uma camada (Figura 7b e Figura 7c).

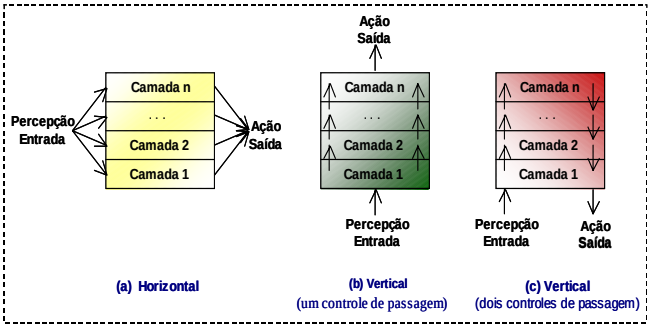

Figura 7 - Informação e fluxo de controle nos três tipos de arquitetura em camadas de agentes "(WOOLDRIDGE, 1999)"

A grande vantagem das arquiteturas em camadas horizontais é sua simplicidade conceitual. Um agente com *n* tipos diferentes de comportamento é estruturado em *n* camadas diferentes. Entretanto, por causa das camadas competirem umas com as outras para gerar sugestões de ações, existe o perigo de que o comportamento global do agente não seja coerente. Com o objetivo de assegurar que tais arquiteturas sejam coerentes, geralmente é incluída uma função de mediador que toma decisões sobre que camada tem controle do agente. A necessidade de tal controle é problemática: significa que o projetista tem que considerar potencialmente todas as possíveis interações entre as camadas. Se existem *n* camadas na arquitetura e cada camada é capaz de sugerir *m* possíveis ações, então existem m^n interações a serem consideradas. A introdução de um sistema de controle central também pode provocar um gargalo na tomada de decisões do agente.

Estes problemas são superados em parte na arquitetura em camadas verticais. Uma arquitetura em camadas verticais pode ser *de uma passagem* ou *de duas passagens*.

Nas arquiteturas verticais de uma passagem, o controle flui seqüencialmente através de cada camada, até que a camada final gera a ação de saída. Nas arquiteturas de duas passagens, a informação flui até chegar à última camada (primeira passagem) e o controle então flui de volta para baixo. Existem algumas similaridades interessantes entre a idéia de arquiteturas em camadas verticais de duas passagens e o modo como as organizações trabalham, com a informação fluindo para cima, para os níveis mais altos da organização, e com os comandos fluindo para baixo.

Em ambas as arquiteturas em camadas verticais, de uma e de duas passagens, a complexidade de interações entre as camadas é reduzida. Já que existem *n-1* interfaces entre *n* camadas, se cada camada é capaz de sugerir *m* ações, existem no máximo m^2 *(n-1)* interações entre as camadas. Isto é claramente mais simples do que no caso das camadas horizontais. Entretanto, esta simplicidade leva ao problema de pouca flexibilidade. Para que uma arquitetura em camadas verticais tome uma decisão, o controle deve passar entre cada uma das diferentes camadas.

Arquiteturas BDI

As arquiteturas BDI (Belief-Desire-Intention) "(RAO E GEORGE, 1995)" baseiam-se na teoria de que os seres humanos são regidos pôr três estados mentais fundamentais: crenças, desejos e intenções. O estado do agente é representado pôr três estruturas: suas crenças (beliefs), seus desejos (desires) e suas intenções (intentions).

As crenças de um agente são o conhecimento que o agente possui sobre o ambiente em que ele se encontra. Os desejos de um agente representam o estado motivacional do sistema. Um estado mental é motivador se é um mecanismo ou representação que tende a produzir, modificar ou selecionar ações à luz das crenças. As intenções de um agente são as ações que têm decidido realizar. Além destes componentes, algumas arquiteturas BDI usam o conceito de planos, que seriam o "passo-a-passo" a ser seguido, quando gerada uma intenção, para a realização de uma ação sobre o ambiente.

A Figura 8 ilustra os componentes principais de uma arquitetura BDI:

- Um conjunto de crenças, que representam as informações que o agente tem sobre seu ambiente atual;
- Uma função de revisão de crenças, que a partir da entrada percebida e as crenças atuais do agente determina um novo conjunto de crenças;
- Uma função geradora de opções, que determina as opções disponíveis para o agente (seus desejos), tomando como base suas crenças e suas intenções atuais;
- Um conjunto de desejos atuais, representando possíveis cursos de ações disponíveis para o agente;
- Uma função filtro, que representa o processo de deliberação do agente e que determina as intenções dos agentes, tendo como base suas crenças, desejos e intenções atuais;
- Um conjunto de intenções atuais, representando o foco atual do agente;
- Uma função de seleção de ação, que determina uma ação para executar tendo como base às intenções atuais.

O entendimento do modelo BDI pode ser alcançado através do seguinte exemplo de raciocínio lógico (adaptado de "WOOLDRIDGE (1999)"). Quando um aluno deixa a universidade com uma primeira graduação, ele se depara com uma decisão a tomar, sobre o que fazer com sua vida. O processo de decisão tipicamente inicia-se tentando se entender as *opções* que se tem disponíveis em função do conhecimento (*crenças*) que o aluno possui do seu ambiente. Por exemplo, se o aluno tem um bom histórico escolar, então uma opção é se tornar um acadêmico. Outra opção é se tornar um profissional. Após a geração do grupo de alternativas (*desejos*), deve-se optar por uma e se comprometer com ela. Esta escolha vem a ser sua *intenção*. As intenções alimentam o raciocínio lógico futuro do agente. Por exemplo, se o aluno decidir que quer se tornar um acadêmico, então ele deve comprometer-se com esse objetivo e despender tempo e esforço para alcançá-lo.

As intenções têm um papel crucial no processo de raciocínio. Talvez, a propriedade mais óbvia das intenções é que elas determinam as ações a serem realizadas. Se o aluno verdadeiramente tem a intenção de torna-se um acadêmico, então se espera de ele agir para a intenção, tentar alcançá-la. Por exemplo, espera-se que se inscreva em vários programas de mestrado (espera-se que ele tente raciocinar para alcançar a intenção), espera-se que ele escolha um curso que acredite ser o melhor para satisfazer sua intenção. Entretanto, se o curso da ação falhar, espera-se que ele tente novamente e não desista. Por exemplo, se a primeira tentativa em um programa de mestrado for rejeitada, espera-se que tente outras universidades alternativas.

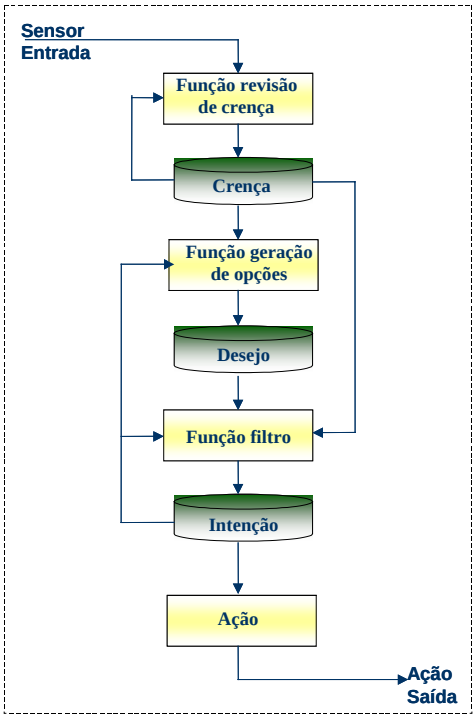

Figura 8 - Diagrama esquemático de uma arquitetura BDI genérica "(WOOLDRIDGE, 1999)"

Uma vez adotada uma intenção, é fato que esta intenção irá restringir o raciocínio futuro. Por exemplo, enquanto se tem uma intenção particular, não haverá consideração por opções que são inconsistentes com esta intenção. Procurar tornar-se um acadêmico, por exemplo, poderia excluir a opção de ir para festas todas as noites: as duas são mutuamente exclusivas.

As intenções persistem. Uma vez intencionado ser um acadêmico, se deve persistir com esta intenção e tentar alcançá-la. Se imediatamente se abandonar a intenção sem dedicar recursos para alcançá-la então nunca se alcançará nada. Entretanto, não se deve persistir por muito tempo, se ficar claro que nunca se tornara um acadêmico, é racional abandonar esta intenção. Similarmente, se a razão da intenção se acaba, é também racional abandoná-la. Por exemplo, se o aluno adotou a intenção de se tornar um acadêmico porque acreditava que seria fácil, mas descobriu que esperariam de que ensinasse, então a justificativa para a intenção não está mais presente e deve ser abandonada.

Finalmente, a intenção esta intimamente relacionada com crenças sobre o futuro. Por exemplo, se aluno pretende ser um acadêmico, então deve acreditar que realmente o será. Se realmente acredita que nunca será um acadêmico, não faz sentido ter a intenção de se tornar um. Entretanto, se pretende ser um acadêmico, deve pelo menos acreditar que existe uma boa chance de que o será.

Agentes deliberativos versus reativos

Como pode observar-se na análise dos tipos de arquiteturas de agentes anteriormente apresentada, a característica básica que as distingue é o mecanismo utilizado para a tomada de decisão da ação a ser executada a partir de uma percepção.

Nesse sentido, as arquiteturas podem ser classificadas em três tipos básicos: deliberativa, reativa e em camadas (Tabela 1).

Agentes deliberativos

Os agentes deliberativos possuem um modelo simbólico do ambiente. A tomada de decisão da ação a ser executada a partir de uma percepção é realizada via raciocínio lógico de acordo ao plano de ação elaborado para atingir a meta.

Um bom exemplo de um agente que trabalha segundo esta arquitetura seria o de um aspirador universal a vácuo "(RUSSEL E NORVIG, 1995)", um pequeno agente robô para limpar uma casa. O robô é equipado com um sensor, que detecta se existe sujeira na casa e um aspirador a vácuo usado para sugar a sujeira. Além disso, o robô tem uma orientação definida (norte, sul, leste ou oeste) e só pode dar um passo para frente ou girar 90º à direita para aspirar a sujeira. O agente move-se ao redor de uma sala que se encontra dividida em quadrados de tamanho igual. Deve-se levar em consideração que a única atividade do agente é limpar a sala, não sendo permitida sua saída desta. Como forma de simplificar o exemplo, a sala é uma matriz 3x3 e o agente sempre começa no quadrado (0,0) da grade, de frente para o norte (Figura 9).

Arquiteturas de agentes	Outros nomes e variações	Descrição
Arquitetura Deliberativa	Arquitetura baseada em lógica Arquitetura BDI Arquitetura intencional Arquitetura baseada em metas Arquitetura baseada na utilidade	Possui um modelo simbólico do ambiente. As decisões são tomadas via raciocínio lógico. Possui um conjunto de metas e intenções. Elaboram plano de ações para alcançar uma meta.
Arquiteturas Reativas	Arquiteturas Reflexivas	Sem modelo simbólico interno As decisões tomadas são implementadas em alguma forma de mapeamento direto da situação para a ação, usando regras de condição/ação.
Arquiteturas em camadas	Arquiteturas híbridas	Mistura componentes das arquiteturas deliberativa e reativa. As decisões tomadas via várias camadas de software.

Tabela 1 - Classificação das arquiteturas de agentes

De forma resumida, o agente pode executar qualquer uma das três ações possíveis: ir para frente, aspirar, ou girar. A meta é percorrer a sala continuamente, procurando por sujeira e removendo-a. Nota-se através deste exemplo que existe uma representação do domínio (no caso, a sala). O agente possui conhecimento sobre este domínio, além de seguir instruções previamente definidas para a realização da tarefa de limpar a sala, o que evidencia suas características de agente deliberativo.

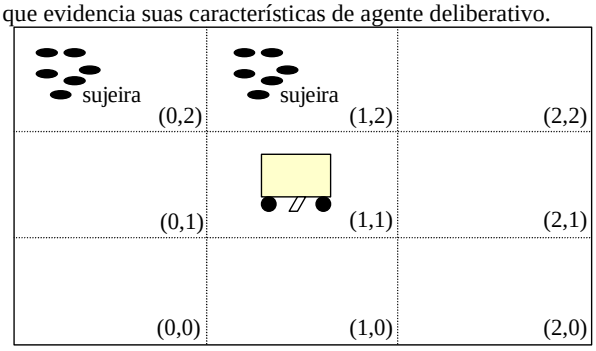

Figura 9 - Aspirador Universal: um exemplo de agente deliberativo "(RUSSEL E NORVIG, 1995)"

Agentes reativos

Os agentes reativos não possuem um modelo do ambiente. Eles agem como uma resposta a um estímulo de ambiente.

"BROOKS (1991)" considera que o comportamento inteligente pode ser gerado sem representações simbólicas e raciocínio lógico mais como uma propriedade emergente de certos sistemas complexos. Para ele, a inteligência é resultado da interação do agente com o ambiente e, portanto, os agentes reativos podem exhibir também comportamento inteligente.

Na Figura 10 podemos observar este tipo de arquitetura. Cada módulo é responsável por uma tarefa definida, não necessariamente complexa e deve possuir todos os recursos necessários para processa-la. Esse tipo de agente pode não resolver qualquer tarefa, pois pode não existir um módulo competente para ela. Uma vez que os módulos trabalham em paralelo deve existir comunicação entre os módulos através de um mecanismo simples.

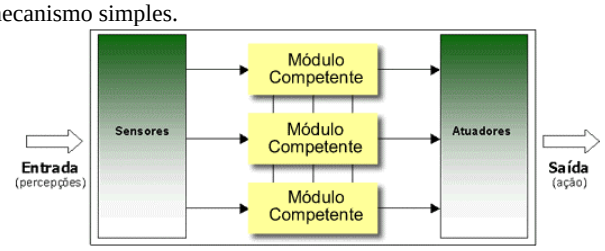

Figura 10 - Arquitetura de um agente reativo genérico "(WOOLDRIDGE, 1995)"

Esta arquitetura possui uma estrutura em módulos descentralizada, o que aumenta a tolerância a erros e a robustez do agente. Em contraste com os agentes deliberativos, a falha de qualquer módulo não implica na falha do sistema. Os módulos competentes têm seus objetivos implícitos que são definidos por suas funcionalidades e não podem ser modificados. Diferentemente dos agentes deliberativos, eles não podem usar seu conhecimento interno para gerar dinamicamente seus novos objetivos. Apesar desta restrição, muitos pesquisadores são da opinião de que os agentes reativos são capazes de ações orientadas a objetivos. De forma similar à inteligência, a orientação a objetivos é resultado da interação com o ambiente e não de uma avaliação ou planejamento centralizado.

O modelo de referência desta classe de sistemas, a arquitetura de subjugação ("subsumption architecture"), deve-se a "BROOKS (1986)" e consiste de uma hierarquia de comportamentos concebida para realização de tarefas específicas. Cada comportamento compete com outros comportamentos influenciando assim as ações finais do agente. Os níveis inferiores da hierarquia representam estilos de comportamentos elementares enquanto que os níveis superiores representam comportamentos mais abstratos e elaborados.

Para ilustrar a arquitetura de subjugação em mais detalhe, apresenta-se a seguir o exemplo de "STEELS (1990)" que mostra como os agentes da arquitetura de subjugação foram utilizados no seguinte cenário:

O objetivo é explorar um planeta distante, mais concretamente, para coletar amostras de um tipo particular de rocha preciosa. A localização das amostras da rocha não é sabida com antecedência, mas elas estão tipicamente agrupadas em pontos certos. Um número de veículos autônomos está disponível, podendo mover-se ao redor do planeta coletando amostras para depois entrar em uma nave espacial que os levará de volta para a Terra. Não existe um mapa detalhado do planeta disponível, embora seja sabido que o terreno é cheio de obstáculos – montes, vales, etc. – que impedem os veículos de trocarem alguma comunicação. Para a solução deste problema, dois mecanismos são introduzidos: o primeiro é um campo gradiente. Com o objetivo de que o agente saiba em que direção a nave está, esta gera um sinal. Um segundo mecanismo habilita os agentes a comunicarem-se uns com os outros. Como o terreno possui alguns obstáculos, a comunicação direta entre os agentes pode não ser possível. Desta forma, um método de comunicação indireta entre os agentes pode ser adotado. Este simples mecanismo possibilita cooperação entre os agentes.

Sistemas multiagente

Um sistema multiagente (SMA) pode ser caracterizado como um grupo de agentes que atuam em conjunto no sentido de resolver problemas que estão além das suas habilidades individuais. Os agentes realizam interações entre eles de modo cooperativo para atingir uma meta.

Sistemas multiagente e sistemas complexos

A construção de software de alta qualidade não é tarefa simples, principalmente devido à complexidade natural do software, caracterizada pelas interações necessárias entre seus componentes. Na procura de técnicas apropriadas para abordar a complexidade do software, os paradigmas de desenvolvimento tem evoluído do paradigma estruturado à orientação a objetos ao paradigma de desenvolvimento de software baseado em agentes (Figura 11).

Mecanismos para abordar a complexidade do software

O paradigma de desenvolvimento baseado em agentes fornece mecanismos para abordar a complexidade do software, como decomposição, abstração e interações flexíveis "(GIRARDI, 2001)" "(HUHNS E STEPHENS, 1999)" "(JENNINGS, 2000)".

Decomposição de software baseada em agentes

Através da decomposição, um problema complexo é dividido em subproblemas mais simples que podem ser abordados de maneira independente. Concentrando-se em um determinado

momento só em um problema simples, os projetistas podem controlar a complexidade de um problema.

Na decomposição de software baseada em agentes, um problema complexo é dividido em subproblemas mais simples: uma comunidade de agentes autônomos que interagem de maneira flexível.

Este tipo de decomposição reduz o nível de acoplamento entre os componentes do sistema. Como os agentes são ativos e autônomos, eles sabem quando eles devem atuar e quando eles devem atualizar seu estado, diferentemente de um objeto passivo, que precisa ser invocado por outro objeto ou entidade externa para fazer isto. Isso reduz a complexidade de controle que não é mais centralizado mas localizado em cada agente.

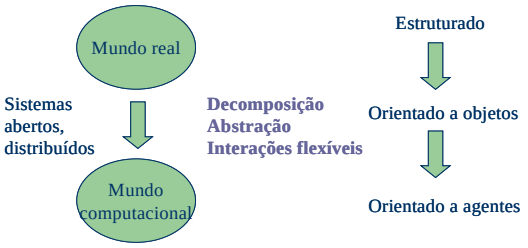

Figura 11 – A evolução dos paradigmas de desenvolvimento para abordar a complexidade do software

Mcanismo de abstração

Um mecanismo de abstração permite definir um modelo mais simples de um problema, enfatizando os aspectos mais pertinentes e ocultando detalhes que são irrelevantes naquele nível de abstração ou situação particular.

A Engenharia de software geralmente confronta a complexidade do mundo real procurando mecanismos de abstração apropriados para mapear o mundo real ao mundo computacional, tentando reduzir a distância cognitiva entre estes dois mundos.

Um agente é uma abstração de software efetiva que fornece uma boa metáfora para a modelagem de sistemas complexos devido, principalmente, à forte correspondência entre a noção de subsistema e de sociedade multiagente. Ambos envolvem um conjunto de componentes agindo e interagindo de acordo com seus papéis.

Interações flexíveis

A interação entre dois componentes de software provavelmente é a característica mais importante de software complexo. Por exemplo, em sistemas abertos é impossível saber exatamente, na fase de projeto, quais os componentes de um sistema e quais as interações entre esses componentes. Nestes sistemas são necessários componentes com comportamento autônomo e flexível.

O paradigma de agentes simplifica o desenvolvimento de sistemas complexos porque os agentes podem decidir sobre o tipo e escopo das suas interações em tempo de execução. Os agentes são capazes de iniciar uma interação e responder a outros agentes de maneira autônoma e flexível.

A autonomia permite o projeto de agentes capazes de atender requisitos não preestabelecidos. Portanto, a abordagem dos SMA é adequada para a modelagem de sistemas abertos, porque os agentes são capazes de reagir a eventos não previstos, explorando as oportunidades que surgem e realizando dinamicamente acordos com outros agentes cuja presença pôde não ter sido prevista no projeto do SMA.

Arquiteturas de sistemas multiagentes

A arquitetura de um SMA mostra a maneira como o sistema está implementado em termos de propriedades e estrutura e como os agentes que o compõem podem interagir a fim de garantir a funcionalidade do sistema.

De acordo com a sua complexidade "(KNAPIK E JOHNSON, 1998)", a arquitetura de um sistema multiagente pode ser classificada em três grupos:

- **Arquitetura simples:** quando é composta por um único e simples agente.
- **Arquitetura moderada:** quando é composta por agentes que realizam as mesmas tarefas, mas possuem diferentes usuários e podem residir em máquinas diferentes;
- **Arquitetura complexa:** quando é composta por diferentes tipos de agentes, cada um com certa autonomia, podendo cooperar e estar em diferentes plataformas.

As arquiteturas dos sistemas multiagente podem ser também classificadas segundo os mecanismos de cooperação e coordenação utilizados na sociedade.

A cooperação entre agentes

Nos sistemas multiagente, uma política de cooperação se faz necessária, uma vez que é através deste mecanismo que os agentes expressam suas necessidades a outros agentes a fim de realizar uma determinada tarefa. O mecanismo de cooperação entre agentes visa determinar a maneira como os agentes expõem suas necessidades a outros agentes para atingirem determinados objetivos.

O processo de cooperação pode acontecer de duas maneiras: Partilha de tarefas e Partilha de resultados. A primeira caracteriza-se pela necessidade de agentes auxiliares durante a execução de uma tarefa por um determinado agente, enquanto que na segunda, os agentes disponibilizam informações para a sociedade, prevendo que algum outro agente possa necessitar dela em determinado momento.

Sendo assim, na grande maioria das vezes, durante a realização de tarefas, os agentes necessitam da ajuda dos outros agentes da sociedade, ou de forma contrária, auxiliam outros agentes para que o objetivo geral do sistema possa ser alcançado. Conseqüentemente, a localização de um determinado agente dentro da sociedade se faz necessária para que a cooperação possa ser efetuada.

As principais abordagens de arquiteturas de SMA, onde a cooperação está em destaque são: a arquitetura quadro-negro, a arquitetura de troca de mensagens e a arquitetura federativa, apresentadas a seguir.

Arquitetura quadro-negro

Em uma sociedade baseada na arquitetura *quadro-negro* os agentes não se comunicam entre si de maneira direta, mas sempre através de um quadro-negro.

O quadro-negro é uma estrutura de dados persistente onde existe uma divisão em regiões ou níveis, visando facilitar a busca de informações. Ele é um meio de interação entre os agentes (uma espécie de repositório), onde estes escrevem e lêem mensagens que serão usadas para atingir o objetivo do sistema. Pode-se assim dizer que um quadro-negro é uma memória de compartilhamento global onde existe uma quantidade de informações e conhecimento usados para leitura e escrita pelos agentes.

Em sistemas multiagente, os quadros-negros são utilizados como um repositório de perguntas e respostas. Os agentes que necessitam de alguma informação escrevem seu pedido no quadro à espera que outros agentes respondam à medida que acessem o mesmo.

O processo de gravação e recuperação das mensagens no quadro-negro pelos agentes pode se tornar demorado demais para

um sistema de tempo real. Essa particularidade faz o uso da arquitetura quadro-negro inviável para sistemas dessa natureza em que um tempo de resposta adequado é fundamental.

Arquitetura de troca de mensagens

Nesta arquitetura, os agentes se comunicam diretamente, uns com os outros, através de mensagens assíncronas. Para isso, é necessário que cada agente saiba os nomes e endereços de todos os agentes que formam o sistema para que as mensagens possam ser trocadas.

Para que as trocas de mensagens ocorram de maneira adequada entre os agentes é necessário estabelecer um protocolo de conversação. O protocolo é quem dita as regras e impõe o formalismo necessário para que as mensagens sejam encaminhadas e compreendidas pelos agentes.

Arquitetura federativa

Considerando uma arquitetura de troca de mensagens, onde o número de agentes é muito grande, a emissão de uma mensagem em *broadcasting* levará um tempo que pode inviabilizar todo processo de comunicação do sistema.

Diante desse problema, surgiu a arquitetura federativa onde os agentes da sociedade são divididos em grupos ou federações segundo um critério de agrupamento escolhido. Junto a cada grupo de agentes encontram-se os agentes chamados de *facilitadores*, responsáveis por receber a mensagem que chega em cada grupo e encaminhá-la para o agente destinatário presente naquele grupo.

A arquitetura federativa permite a diminuição do fluxo de mensagens desnecessárias entre os agentes que formam a sociedade, pois os facilitadores têm a capacidade de remetê-las ao respectivo destinatário sem a necessidade de enviá-las a todos os agentes.

A coordenação entre agentes

A coordenação entre agentes refere-se à maneira em que os agentes estão organizados a fim de cooperar para alcançar um objetivo comum do sistema. Existem dois mecanismos básicos para coordenar os agentes: o mecanismo mestre-escravo e o mecanismo de mercado "(PARAISO, 1997)".

Nas arquiteturas do tipo “mestre-escravo” existem duas classes de agentes: os gerentes (mestres) e os trabalhadores (escravos). Os agentes trabalhadores são coordenados por um gerente que distribui as tarefas entre estes e espera o resultado. O agente facilitador pode existir se os agentes estiverem divididos em grupos.

Nas arquiteturas baseadas no mecanismo de mercado, todos os agentes estão em um mesmo nível e sabem as tarefas que cada agente é capaz de desempenhar. Esta arquitetura visa diminuir a quantidade de mensagens trocadas, tendo em vista que cada agente já conhece todos os outros. Baseado nesse mecanismo surgiu a Programação Orientada ao Mercado (MOP), onde os agentes podem ser classificados como produtores e consumidores e são utilizados para maximizar lucros através de um processo de negociação "(PARAISO, 1997)".

A comunicação entre agentes

A comunicação é uma importante característica que os agentes inteligentes possuem para atingirem seus objetivos.

A Linguagem de Comunicação entre Agentes (ACL - Agent Communication Language) foi definida para a comunicação entre agentes e pode ser divida em "(FININ, LABROU E MAYFIELD, 1993)":

-Vocabulário: dicionário de conceitos do domínio de aplicação do sistema multiagente e seus relacionamentos. Também conhecido como ontologia.

-Linguagem interna: KIF (Knowledge Interchange Format), linguagem de programação em lógica de primeira ordem com capacidade de codificar dados simples, restrições, regras e expressões.

-Linguagem externa: KQML (Knowledge Query and Manipulation Language) que é uma camada lingüística capaz de encapsular estruturas KIF, fornecendo uma comunicação mais eficiente.

Uma mensagem ACL nada mais é do que uma expressão KQML onde os argumentos são termos ou sentenças KIF formadas por termos da ontologia.

Aplicações

Muitos sistemas baseados em agentes já foram desenvolvidos em uma variedade de áreas de aplicação: industrial (Controle de tráfico aereo, Manufatura, Controle de processos, Sistemas de transporte), comercial (Gerência de informação, Comércio eletrônico, Gerência de processos de negócio) e médicas (Monitoramento de pacientes) "(JENNINGS, 2000)".

Alguns exemplos de essas aplicações são:

- Comércio eletrônico, onde agentes “compradores” e “vendedores” compram e vendem bens e serviços em representação de seus usuários;
- Sistemas de transporte, onde os agentes representam veículos de transporte, bens ou usuários transportados;
- Acesso a informação, em ambientes dinâmicos, como a Internet, onde os agentes são responsáveis pela recuperação e filtragem de informação;
- Controle de tráfico, onde os agentes são responsáveis por interpretar dados vindos de diferentes sensores de forma a melhorar o tráfico urbano ou aéreo;
- Jogos interativos, onde agentes animados jogam com humanos ou outros agentes;
- Simulação de sociedades (inclusive humanas), onde os agentes desempenham os papéis dos membros da sociedade.

METODOLOGIAS DE DESENVOLVIMENTO

A sistematização de técnicas e metodologias para a Engenharia de software baseada em agentes é um tópico de ativa pesquisa.

De uma forma similar ao que aconteceu com o desenvolvimento orientado a objetos, uma grande quantidade de técnicas e metodologias tem sido propostas para a Engenharia de software baseada em agentes e ainda não existe um padrão reconhecido que reúna as vantagens de cada uma delas.

A maioria das propostas aborda as fases de análise de requisitos e projeto de sistemas multiagente e se inspira ou estende conceitos das técnicas para o desenvolvimento orientado a objetos. Outras , como CoMoMAS "(GLASSER, 1997)" e MAS-CommonKADS "(IGLESIAS, GARIJO E GONZALEZ, 1998)" utilizam conceitos de modelagem da Engenharia do conhecimento.

Técnicas para a análise de requisitos

No processo de desenvolvimento de software, a fase de análise de requisitos visa definir *o que* o sistema deve fazer e, de acordo com o paradigma de desenvolvimento utilizado, especificar um modelo com a representação dos requisitos do software.

Os métodos para a análise de requisitos de sistemas multiagente geralmente focalizam a modelagem de objetivos, papéis, atividades e interações de indivíduos de uma organização. Apesar de ainda não existir uma definição comum desses conceitos, os significados seguintes são geralmente atribuídos aos mesmos.

Uma organização está composta de indivíduos com objetivos gerais e específicos que estabelecem o que a organização pretende alcançar. O alcance de todos os objetivos específicos permite

alcançar o objetivo geral da organização. Por exemplo, um sistema de informação pode ter o objetivo geral de “satisfazer as necessidades de informação de uma organização” e os objetivos específicos de “satisfazer as necessidades de informação dinâmica” e “satisfazer as necessidades de informação em longo prazo”.

Os objetivos específicos são alcançados através do exercício de responsabilidades que os indivíduos têm. Os indivíduos desempenham papéis com um certo grau de autonomia e exercitam suas responsabilidades através da execução de atividades. Para isso, eles dispõem de recursos. Por exemplo, um indivíduo pode desempenhar o papel de “recuperador de informação” com a responsabilidade de executar atividades para satisfazer as necessidades de informação dinâmicas de uma organização. Outro indivíduo pode desempenhar o papel de “filtrador de informação” com a responsabilidade de executar atividades para satisfazer as necessidades de informação a longo prazo da organização. Recursos podem ser, por exemplo, as regras da organização para acessar e estruturar suas fontes de informação.

Às vezes, os indivíduos têm que comunicar-se com outros indivíduos internos ou externos para cooperar na execução de uma atividade. Por exemplo, os indivíduos que fazem os papéis de “recuperador de informação” e “filtrador de informação” podem precisar interagir com o indivíduo “gerenciador de fontes de informação” que tem a responsabilidade de armazenar e atualizar as fontes de informação da organização.

De acordo com estas definições, a maioria das técnicas para a análise de requisitos da Engenharia de software multiagente estabelecem os procedimentos a serem seguidos nas seguintes tarefas de modelagem:

- *Modelagem de objetivos*. Considerando o problema que o sistema pretende resolver, o objetivo geral do sistema é identificado. Através de um refinamento do objetivo geral, são obtidos os objetivos específicos. Um modelo de objetivos é o produto desta tarefa de modelagem.

- *Modelagem de papéis*. Para cada objetivo específico, são identificadas as responsabilidades que serão exercitadas pelos papéis internos e externos da organização. Então, são definidas as atividades que deverão ser realizadas no exercício de cada responsabilidade. Durante este processo de refinamento, pode ser identificado que a mesma atividade ou um conjunto de atividades relacionadas são executadas por vários papéis. Neste caso, deveria ser criado um papel independente com a responsabilidade de executar estas atividades em nome dos outros papéis. Os recursos que um papel precisará para executar suas atividades são também identificados. Um *modelo de papéis* é o produto desta tarefa de modelagem.

- *Modelagem de interações*. Através de uma análise das suas atividades respectivas, são identificadas as interações entre papéis internos e entre papéis e entidades externas. Um *modelo de interações* é o produto desta tarefa de modelagem.

Além dessas tarefas de modelagem, as versões mais recentes de algumas metodologias "(WOOD E DELOACH, 2001)" "(WOOD, DELOACH E SPARKMAN, 1998)" "(COSSENTINO *et al*., 2002)" "(CAIRE *et al*., 2002)" propõem também a modelagem dos conceitos do domínio da aplicação de software e os relacionamentos entre esses conceitos como base para a construção da(s) ontologia(s) do sistema multiagente. Um *modelo de conceitos* é o produto desta tarefa de modelagem.

A Tabela 2 apresenta as tarefas de modelagem de algumas técnicas para a análise de requisitos de sistemas multiagente: GAIA “(WOOLDRIDGE, JENNINGS E KINNY, 2000)”, MaSE "(WOOD E DELOACH, 2001)" "(WOOD, DELOACH E SPARKMAN, 1998)", MADS “(GIRARDI E SODRÉ, 2002)” “(SODRÉ, 2002)”, TROPOS "(CASTRO, KOLP E MYLOPOULOS, 2001)" "(MYLOPOULOS E CASTRO, 2000)", SODA “(OMICINI, 2001)”, PASSI "(COSSENTINO *et al*., 2002)" , MESSAGE “(CAIRE ET ALII., 2001)” e COMPOR-M “(OLIVEIRA DE ALMEIDA, 2004)”.

Às vezes, as tarefas de modelagem bem como os produtos construídos são nomeados de forma diferente à descrição geral anteriormente apresentada. Também, a técnica de construção e representação de um determinado produto pode variar de uma a outra metodologia. A seguir são descritas tarefas e produtos de modelagem particulares de cada metodologia, não incluídos na descrição geral de tarefas e produtos anteriormente realizada.

Na metodologia MaSE, os objetivos são mapeados em papéis mas esse mapeamento é auxiliado pela identificação de casos de uso e a construção de diagramas de seqüência. A representação dos casos de uso e do diagrama de seqüência é baseada na UML “(FOWLER E SCOTT, 2000)”, linguagem para modelagem de sistemas orientados a objetos. Os casos de uso definem cenários básicos de interação com o usuário e são extraídos da especificação de requisitos. Ùm diagrama de seqüência é construído para cada caso de uso, descrevendo a seqüência de interações entre os papéis do sistema. Além dessas tarefas, MaSE também propõe a modelagem do comportamento de cada papel, utilizando a notação dos diagramas de estado da UML.

Na metodologia TROPOS "(CASTRO, KOLP E MYLOPOULOS, 2001)" "(MYLOPOULOS E CASTRO, 2000)" o conceito de ator substitui o conceito de papel das outras metodologias. Os atores têm objetivos a atingir através da execução de atividades e com o auxílio de recursos. O produto da fase de análise é chamado de *modelo de dependência estratégica*, um grafo onde cada nó representa um ator e os arcos as dependências entre atores.

Tarefas de modelagem	Técnicas							
	GAIA	MaSE	MADS	TROPOS	SODA	PASSI	MESSAGE	COMPOR-M
Modelagem de objetivos		•	•	•	•	•	•	
Modelagem de papéis	•	•	•	•	•	•	•	•
Modelagem de interações	•	•	•	•	•	•	•	•
Modelagem de conceitos		•				•	•	•

Tabela 2 – Tarefas de modelagem de algumas técnicas para a análise de requisitos de sistemas multiagente

A metodologia PASSI "(COSSENTINO *et al*., 2002)" também utiliza conceitos de modelagem da UML. Para a identificação e modelagem de papéis são previamente identificados os casos de uso do sistema, onde os cenários são representados em diagramas de seqüência. Os casos de uso ou pacotes de casos de uso dão origem a agentes. As entidades externas que interagem com os agentes são representadas como atores. Os agentes atuam para alcançar seus objetivos e para isso precisam interagir com outros agentes ou atores. Cada agente pode desempenhar um ou mais papéis, identificados através da construção de diagramas de seqüência que representam as responsabilidades de cada agente em cenários específicos. Um agente pode participar de vários cenários tendo diferentes papéis em cada um e o mesmo agente pode aparecer mais de uma vez em um diagrama de seqüência específico.

Na modelagem de conceitos da metodologia PASSI são construídos diagramas de classe que descrevem o conhecimento dos agentes, dos atores e de suas interações. A construção do diagrama de classes é baseada na atividade de identificação dos agentes, onde cada agente é introduzido como uma classe no diagrama e, cada mensagem em um diagrama de seqüência é representada como uma associação no diagrama de classes.

A metodologia MESSAGE "(CAIRE *et al*., 2002)" também utiliza conceitos de modelagem da UML. MESSAGE é baseada nos conceitos de agente, organização, papel, recurso, tarefa, interação e objetivo. Um agente é uma entidade autônoma. Uma organização é um grupo de agentes trabalhando em virtude de um objetivo comum. Um papel descreve as características externas de um agente em um contexto particular e um agente pode desempenhar vários papéis. O recurso é usado para representar entidades não autônomas, por exemplo, bases de dados. Uma tarefa é uma atividade realizada por um agente. O produto da análise de requisitos é um modelo de visões composto de: *visão da organização*, representando organizações, agentes, papéis e recursos e seus relacionamentos; *visão das tarefas e objetivos*, representando os objetivos, as tarefas e as dependências entre os mesmos; *visão dos papéis e agentes*, representando os papéis que os agentes desempenham e seus atributos, tais como, objetivo, recursos, tarefas e interações; *visão das interações* e a *visão do domínio*, representando os conceitos e relacionamentos específicos do domínio.

Técnicas para o projeto arquitetural e detalhado

No processo de desenvolvimento de software a fase de projeto visa definir uma solução ao problema especificado no modelo de requisitos, de acordo com o paradigma de desenvolvimento utilizado. O produto desta fase é uma arquitetura de software e uma especificação detalhada dos componentes da arquitetura. A fase é geralmente dividida em duas subfases:

- O *projeto arquitetural*, onde são definidos os diferentes componentes da arquitetura e a forma em que eles cooperam para a alcançar o objetivo geral do sistema.
- O *projeto detalhado*, onde é definido o comportamento e os atributos de dados de cada um dos componentes da arquitetura.

A maioria das técnicas para o projeto de software da Engenharia de software multiagente estabelecem os procedimentos a serem seguidos nas seguintes tarefas de modelagem:

- *Projeto arquitetural*
  - *Modelagem de agentes.* O objetivo desta tarefa é identificar os agentes que irão compor a arquitetura do sistema multiagente e representa-los em um *modelo de agentes.* Para isso é analisado o modelo de papéis especificado na fase de análise de requisitos onde, inicialmente, é realizado o mapeamento um "papel" para um "agente", sendo cada papel associado a um agente. Porém, podem existir situações onde um agente pode englobar mais de um papel, ou ainda, da necessidade de atribuir um papel a mais de um agente, de forma a atender requisitos de coesão, desempenho e reusabilidade. Cada agente irá realizar um conjunto de atividades necessárias para o cumprimento de suas responsabilidades. Estas atividades são derivadas diretamente das atividades dos papéis desempenhados pelos agentes. È importante ressaltar que, nesta tarefa, é realizada apenas a identificação dos agentes que irão constituir a arquitetura do sistema multiagente, sem a preocupação com a arquitetura interna do agente. O produto desta tarefa é um *modelo de agentes* que especifica os papéis atribuídos a cada agente e um *modelo de atividades* que especifica as atividades a serem executadas por cada agente.
  - *Modelagem de interações.* O objetivo desta tarefa é identificar as interações entre os agentes que compõem a arquitetura do sistema multiagente (origem, destino e seqüência) e representa-los em um *modelo de agentes.* Para isso é realizado um mapeamento das interações entre papéis às interações entre agentes considerando os papéis que cada agente desempenha. O produto desta tarefa é um *modelo de interações* que especifica as interações entre os agentes do sistema.
  - *Modelagem da arquitetura do sistema multiagente.* O objetivo desta tarefa é, a partir dos agentes e interações identificados nas tarefas anteriores, especificar uma solução computacional ao problema especificado na fase de análise de requisitos. São especificados os mecanismos de cooperação e coordenação entre agentes a fim de que o sistema resolva eficientemente o problema. O produto desta tarefa é um *modelo arquitetural* do sistema multiagente. Consultando coletâneas ou sistemas de padrões disponíveis, soluções de projeto arquitetural podem ser reutilizadas.
- *Projeto detalhado*
  - *Modelagem detalhado dos agentes.* O objetivo desta tarefa é especificar a arquitetura de cada agente da sociedade. A partir de um detalhamento das atividades a serem executadas por cada agente, é feito uma análise do seu comportamento e do mecanismo mais apropriado para a tomada de decisões da ação a executar a partir de uma percepção. Assim, os agentes são estruturados de acordo a uma estrutura deliberativa ou reativa. O produto desta tarefa é um *modelo de projeto detalhado*, que especifica a arquitetura de cada agente do sistema. Consultando coletâneas ou sistemas de padrões disponíveis, soluções de projeto detalhado podem ser reutilizadas.

A Tabela 3 apresenta as tarefas de modelagem de algumas técnicas para o projeto de sistemas multiagente: GAIA "(WOOLDRIDGE, JENNINGS E KINNY, 2000)", MaSE "(WOOD E DELOACH, 2001)" "(WOOD, DELOACH E SPARKMAN, 1998)", MADS "(GIRARDI E SODRÉ, 2002)" "(SODRÉ, 2002)", TROPOS "(CASTRO, KOLP E MYLOPOULOS, 2001)" "(MYLOPOULOS E CASTRO, 2000)", SODA "(OMICINI, 2001)", PASSI "(COSSENTINO *et al*., 2002)" e COMPOR-M "(OLIVEIRA DE ALMEIDA, 2004)".

Subfases	Tarefas de modelagem	Técnicas						
		GAIA	MaSE	MADS	TROPOS	SODA	PASSI	COMPOR-M
Projeto arquitetural	Modelagem de agentes	•	•	•	•	•	•	•
	Modelagem de interações	•	•	•	•	•	•	•
	Modelagem da arquitetura do sistema multiagente	•	•	•	•	•	•	
Projeto detalhado	Modelagem de atividades e dados dos agentes				•	•		•

Tabela 3 – Tarefas de modelagem de algumas técnicas para o projeto de sistemas multiagente

Às vezes, as tarefas de modelagem bem como os produtos construídos são nomeados de forma diferente à descrição geral anteriormente apresentada. Também, a técnica de construção e representação de um determinado produto pode variar de uma a outra metodologia. Por exemplo, em algumas metodologias, as interações entre agentes são representadas em diagramas de seqüência, em outras, em diagramas de estado. A seguir são descritas tarefas e produtos de modelagem particulares de cada metodologia, não incluídos na descrição geral de tarefas e produtos anteriormente realizada.

Na metodologia MaSE, além das tarefas especificadas na Tabela 3 é construído um *diagrama de desenvolvimento* que mostra a quantidade e local de cada tipo de agente no sistema.

Na metodologia TROPOS, no projeto detalhado dos agentes do sistema, são considerados apenas agentes deliberativos. Portanto, nesta subfase são modelados os eventos internos e externos que disparam planos e crenças envolvidas no raciocínio do agente. As atividades do agente são representadas em um *diagrama de habilidades* utilizando a notação do diagrama de atividades da AUML "(ODELL, PARUNAK E BAUER, 2000)".

A Figura 13 mostra o diagrama de habilidades referente a busca dos resultados de uma consulta efetuada pelo agente *User Interface*. Os nós representam planos e os arcos representam eventos internos e externos.

No *diagrama de planos*, cada nó (plano) do *diagrama de habilidades* é especificado em um diagrama de ação da AUML. Na Figura 14 temos um exemplo de *diagrama de planos* referente à habilidade *Exchange Information*.

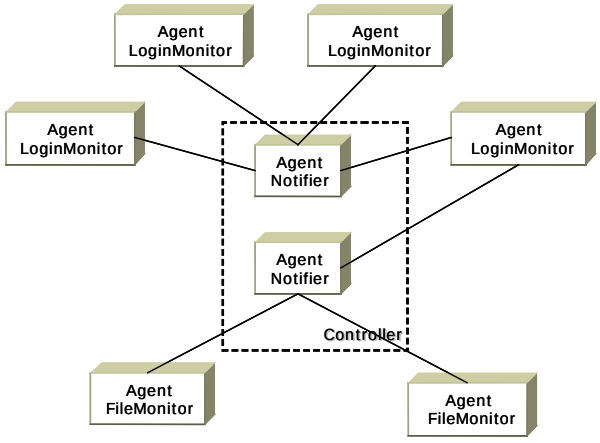

Figura 12 - Exemplo de diagrama de desenvolvimento da MaSE

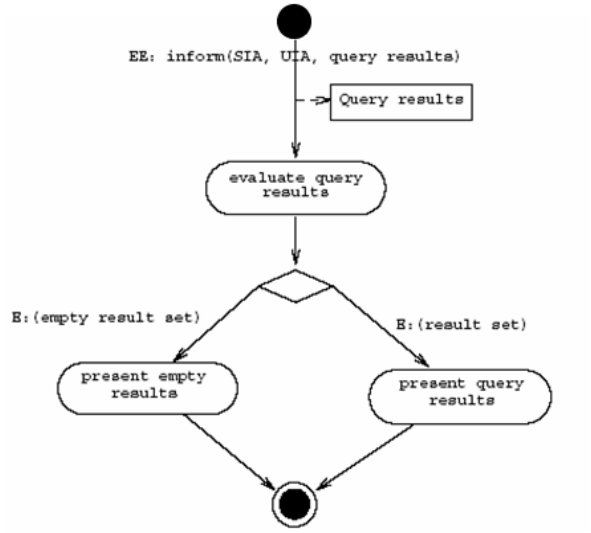

Figura 13 - Diagrama de habilidades usando o diagrama de atividades da AUML na metodologia TROPOS

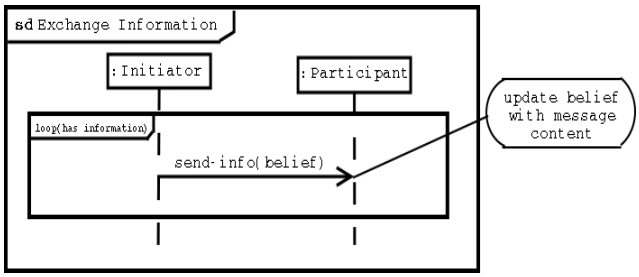

Figura 14 - Exemplo de diagrama de planos na metodologia TROPOS

Além da fase de projeto, a metodologia PASSI "(COSSENTINO *et al.*, 2002)" define técnicas para a fase de implementação do sistema multiagente. Nesta fase são construídos os *modelos de implementação de agentes, de código e de desenvolvimento.*

O *modelo de implementação de agentes* é uma solução em termos de classes e métodos, representada em um diagrama de classes (Figura 15) e em diagramas de atividades (0).

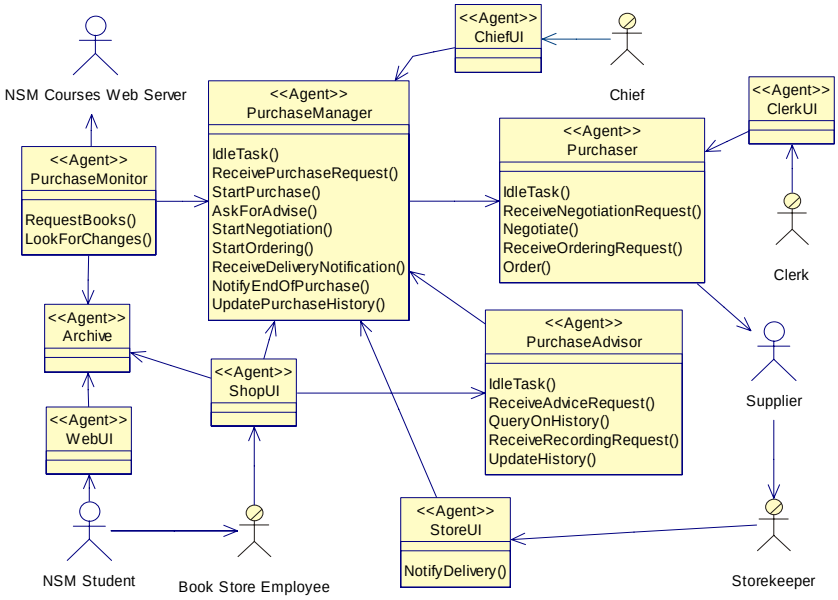

Figura 15 - Exemplo de diagrama de classes na PASSI

Já *o modelo de código* provê a solução em nível de código. Para tanto, é construido o código fonte final do sistema possívelmente reutilizando classes de uma biblioteca de classes disponível.

Finalmente, é construido o *modelo de desenvolvimento*, com a distribuição das partes do sistema nas unidades de hardware. Para isso, utiliza-se um *diagrama de distribuição*, que descreve a alocação dos agentes às unidades de processamentos disponíveis.

Na metodologia COMPOR-M, o comportamento (habilidades) requeridas para cada agente é mapeado em serviços providos por componentes funcionais. Caso não existam, serão implementados e estarão disponíveis para reutilização em futuros sistemas. Além da fase de projeto, a metodologia COMPOR-M também fornece suporte às fases de implementação e teste.

AUML – Uma Linguagem de Especificação para o Desenvolvimento de SMA

A AUML (Linguagem de Modelagem Unificada para Agentes) surgiu tendo em vista que a UML (Linguagem de Modelagem Unificada) possui limitações para modelagem de agentes e sistemas baseados em agentes. Dado que a UML tem tido uma larga aceitação na modelagem de software orientado a objetos, nada mais coerente que tentar expandir a UML na utilização de agentes, tendo em vista que um agente pode ser considerado como a extensão de um objeto. Logo, essa extensão levou em consideração as exigências e as distinções da tecnologia de agentes, surgindo então a AUML "(ODELL, PARUNAK E BAUER, 2000)". "ODELL, PARUNAK E BAUER (2000)" abordam um subconjunto da AUML, onde são especificados Protocolos de Interações de Agentes (AIP) e outras noções baseadas em agentes.

De forma geral, as técnicas para representação dos AIP's foram divididas em três níveis de apresentação "(ODELL, PARUNAK E BAUER, 2000)":

Nível 1: representação dos protocolos de interações globais do sistema (projeto arquitetural)

Nível 2: representação das interações entre os agentes (projeto arquitetural)

Nível 3: representação do processamento interno do agente (projeto detalhado)

No nível 1, os protocolos de interação de agentes (AIP's) oferecem soluções reutilizáveis que podem ser aplicadas em vários tipos de seqüências de mensagens que são encontradas entre agentes. Na AUML há uma técnica que expressa soluções de protocolos para o reuso: os pacotes.

Os pacotes são mecanismos de propósito geral utilizados para organizar elementos do modelo em grupos, podendo um pacote estar inserido dentro de um outro pacote. Os pacotes são utilizados para tratar uma grande funcionalidade do sistema.

A Figura 16 mostra dois pacotes. O pacote comprador expressa um protocolo simples entre um corretor e um varejista. O varejista recebe uma proposta de compra do corretor que responde com uma contraproposta. Para certos produtos, o varejista pode solicitar previamente a um atacadista as disponibilidades e os custos dos produtos solicitados pelo comprador. Baseado nas informações recebidas do atacadista, o varejista pode fornecer uma proposta mais precisa. Toda essa operação poderia ser colocada dentro do pacote comprador, entretanto, há situações em que não há necessidade de protocolos adicionais envolvendo o atacadista. Portanto, dois pacotes podem ser definidos: o pacote comprador e o pacote fornecedor. Quando um cenário particular requer um protocolo com o atacadista, o pacote fornecedor pode ficar aninhado ao pacote comprador, caso contrário ele pode ficar como um pacote distinto.

O nível 2 utiliza diagramas de seqüência, colaboração, atividade e estado que mostram como os agentes interagem entre si através da troca de mensagens. A representação gráfica dos diagramas de seqüência (0) e colaboração (Figura 18) mostra a seqüência cronológica das comunicações entre os agentes.

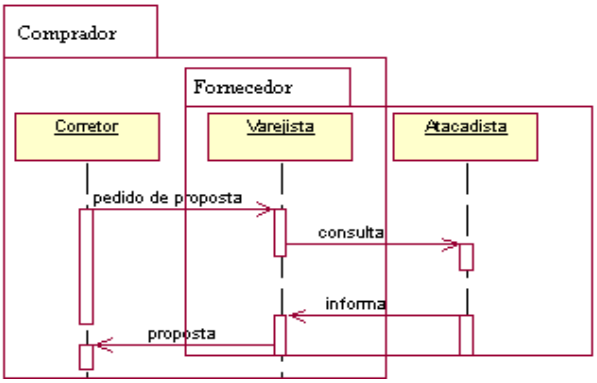

Figura 16 - Utilização de pacotes para expressar protocolos aninhados

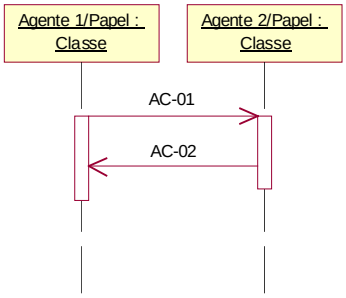

Figura 17 - Exemplo de um diagrama de seqüência entre agentes "(ODELL, PARUNAK E BAUER, 2000)"

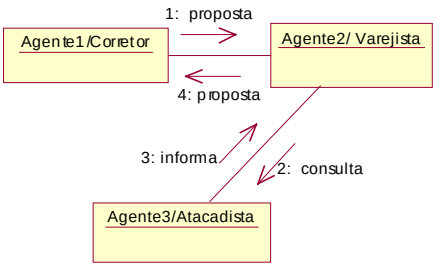

Figura 18 - Exemplo de um diagrama de colaboração entre os agentes

O diagrama de atividades e o diagrama de estados mostram o fluxo do processamento da informação em um sistema multiagente. Os diagramas de atividades (0) fornecem uma visão baseada no processamento das informações, onde os agentes são representados pelas divisões verticais, as operações são representadas por retângulos de bordas arredondadas e os eventos por setas.

Um diagrama de estado (Figura 20) é um gráfico que representa uma máquina de estados, onde estados são simbolizados por retângulos arredondados e as transições por setas que interconectam os estados.

No nível 3 é abordado o comportamento interno dos agentes. Para especificar esse comportamento utiliza-se os diagramas de estado e de atividades do nível 2.

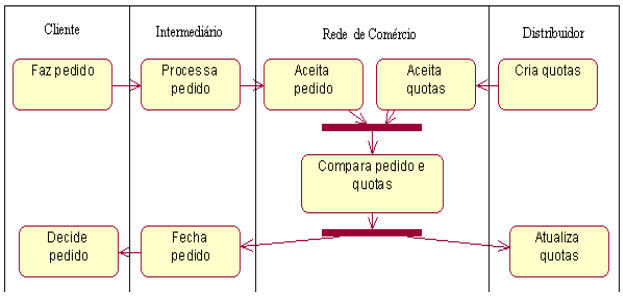

Figura 19 - Exemplo de um diagrama de atividades

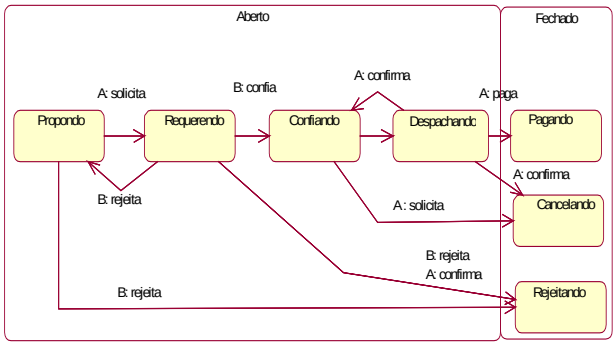

Figura 20 - Exemplo de um diagrama de estados

Padrões de projeto para o desenvolvimento de sistemas multiagente

O conhecimento aquirido através do desenvolvimento de SMAs está permitido a extração e especificações de boas experiências de projeto, na forma de padrões.

Um padrão de projeto é uma solução comum bem sucedida para um problema também comum, descrito por vários atributos que podem variar de acordo com os critérios adotados. Porém, existem atributos essenciais que devem ser claramente identificáveis ao se ler um padrão:

Nome: todo padrão deve ter um nome significativo.

Contexto: estabelece pré-condições dentro das quais o problema e sua solução costumam ocorrer e para as quais a solução é desejável, o que reflete a aplicabilidade do padrão.

Problema: estabelece o problema a ser resolvido pelo padrão, descreve a intenção e objetivos do padrão perante o contexto e forças específicas.

Força: descreve os impactos, influências, restrições relevantes ao problema e como eles interagem ou são conflitantes entre si bem como quando aplicar o padrão no contexto.

Solução: são instruções que descrevem como o problema é resolvido, podendo para isso utilizar texto, diagramas e figuras.

Usos conhecidos: são exemplos bem sucedidos do uso dos padrões em sistemas reais.

No contexto do projeto “Engenharia de Domínio Multiagente” "(GIRARDI, 2003)" uma coletânea de padrões arquiteturais “(SILVA, 2003)” para o desenvolvimento de SMA e um sistema de padrões para o desenvolvimento de SMA adaptativos/adaptáveis baseados na modelagem de usuários "(OLIVEIRA, 2004)" "(GIRARDI, 2003)" foram desenvolvidos. Um sistema de padrões é um conjunto coeso de padrões co-relacionados.

Padrões arquiteturais para o desenvolvimento de SMAs

A seguir é apresentada parte da coletânea de padrões arquiteturais para o desenvolvimento de SMA "(SILVA, 2003)" desenvolvida no contexto do projeto de pesquisa "Engenharia de Domínio Multiagente" "(GIRARDI, 2003)": os padrões *quadro-negro, federativo, mestre-escravo* e *camadas.*

A solução proposta pelos padrões segue a notação gráfica da AUML "(ODELL, PARUNAK E BAUER, 2000)". As interações entre agentes são descritas utilizando a linguagem KQML "(FININ, LABROU E MAYFIELD, 1993)".

Padrão Quadro-negro

Contexto: Os agentes não podem comunicar-se entre si de forma direta. Necessita-se de uma memória de compartilhamento global. Necessita-se construir um repositório de perguntas e respostas que possa ser acessado pelos agentes em busca de conhecimento ou informação.

Problema: Como podem os agentes ter acesso a informações e conhecimento de maneira facilitada sem se comunicarem diretamente?

Força: Os agentes precisam colaborar para executar tarefas complexas que se estendem além das suas capacidades individuais.

Solução: A solução para o problema está na criação de um ambiente de coordenação passivo chamado de quadro-negro.

Esse ambiente é acessado por dois tipos de agentes: o especialista e o supervisor. O primeiro pode acrescentar, atualizar ou apagar dados do quadro-negro. Vários agentes no papel de especialista tentam ao mesmo tempo inserir informações no quadro-negro usando uma mensagem do tipo *insert(X)*. O especialista também monitora continuamente o quadro-negro na procura de mudanças. O supervisor é responsável por decidir qual especialista deve responder a uma mudança, quando múltiplos especialistas quiserem responder ao mesmo tempo. O supervisor é uma espécie de controlador das ações dos especialistas. O quadro-negro usa uma mensagem do tipo pipes(X) para redirecionar as mensagens ao supervisor que responde com uma mensagem do tipo reply(X) e indica quem deve acessar o quadro-negro.

A solução é composta por dois pacotes (*Participantes* e *Controle*). Os participantes são todos os agentes que fazem parte do padrão. O pacote de controle é formado pelo agente supervisor e o quadro negro que lista as intenções dos especialistas para o supervisor.

A Figura 21 ilustra a interação entre o pacote *Participantes* e o pacote *Controle*. O pacote *Participantes* é composto pelos agentes especialistas e o agente supervisor, o pacote *Controle* pelo quadro-negro e pelo agente supervisor.

Padrão Federativo

Contexto: Os agentes da sociedade precisam ser divididos em grupos ou federações com o objetivo de auxiliar na diminuição do fluxo de mensagens entre os agentes da sociedade.

Problema: Quando uma mensagem em broadcasting pode inviabilizar todo o processo de comunicação de um sistema, como pode reverter-se esse quadro?

Força: Um agente facilitador gerencia as mensagens enviadas à federação e remete-as ao agente destinatário apropriado cujo nome e endereço não precisam ser conhecidos pelo agente que envia a mensagem.

Solução: Observando o contexto podemos dividir o sistema em grupos menores seguindo um critério de agrupamento. Em cada um desses grupos existe um agente facilitador que é responsável por receber e encaminhar a mensagem para o agente destinatário (Figura 22).

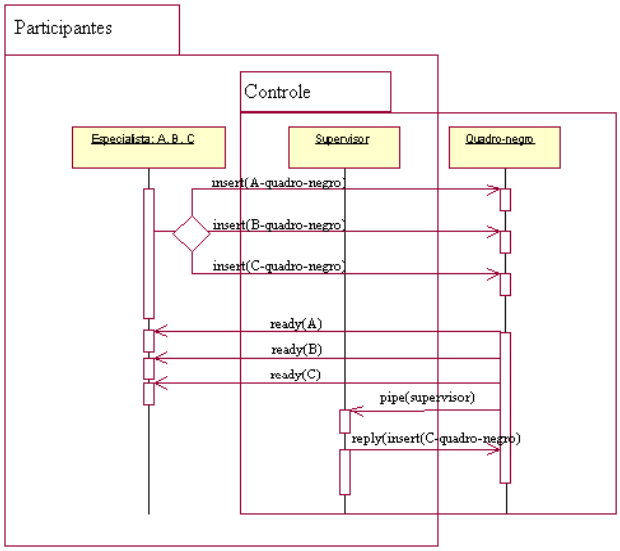

Figura 21 - Diagrama de pacotes com as interações entre os agentes no padrão Quadro-negro

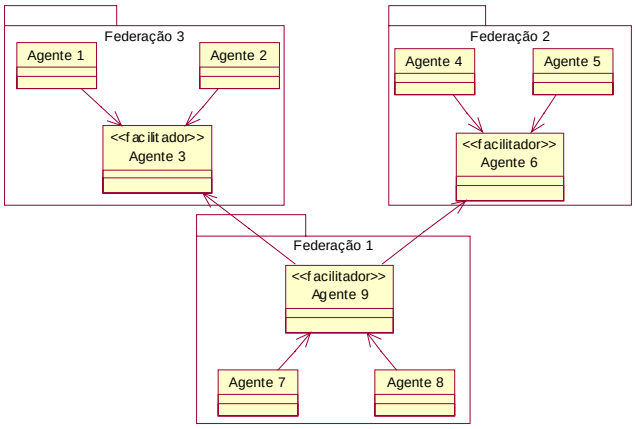

Figura 22 - Estrutura do padrão Federativo

A Figura 23 mostra o diagrama de pacotes onde o agente 8 faz uma pergunta ao agente facilitador 9, usando *askif(X)*. Como ele não sabe responder e nem conhece ninguém da sua federação capaz de responder, essa pergunta é encaminhada para os demais agentes facilitadores, através de uma mensagem do tipo *broadcast(ask-if(X))*. Os agentes facilitadores observam se alguém de sua federação pode responder. No exemplo da Figura 23, o agente facilitador 6 recebe a pergunta e a encaminha para o agente 5, usando *recommend-one(ask(X))*. O agente 5 responde retornando a solução para o agente facilitador 6, usando *tell(X)* que tem a função de encaminhar a resposta até o solicitante inicial agente 8, usando *forward(tell(X))*.

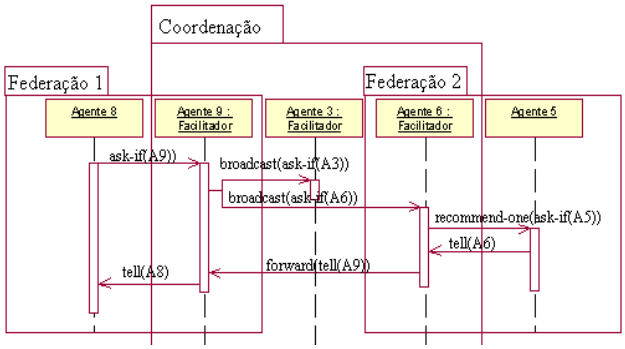

Figura 23 - Diagrama de pacotes e a interação dos agentes na federação

Padrão Camada

Contexto: Um sistema muito grande necessita ser decomposto em subsistemas, para facilitar o entendimento, a manutenção e o reuso dos subsistemas.

Problema: Como estruturar e organizar as dependências entre subsistemas que estão em diferentes níveis de abstração.

Força: As camadas são organizadas de maneira hierárquica. Cada camada tem o objetivo de prover serviços bem definidos para a camada seguinte, a qual funciona como um cliente para a camada anterior.

Solução: A solução envolve a definição de um critério de abstração para a divisão das responsabilidades entre as camadas. Em seguida, determina-se o número de níveis de abstração, dá-se um nome as camadas e associa-se o serviço de cada uma delas. Observa-se na Figura 24, que a camada j, fornece serviços a camada j+1 e delega tarefas à camada j-1.

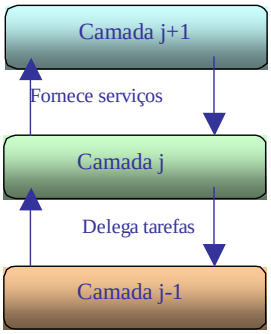

Figura 24 - Estrutura do padrão camada

Padrão Mestre-Escravo

Contexto: Para abordar a complexidade de uma tarefa, o projetista precisa dividir uma tarefa em sub-tarefas menores a serem executadas por diferentes agentes.

Problema: Como coordenar a realização de uma tarefa complexa?

Força: Um agente pode dividir uma tarefa e delegar sub-tarefas a outros agentes para melhorar a confiança, desempenho, ou precisão da tarefa que é executada.

Solução: A solução envolve um mestre que divide a tarefa em sub-tarefas, em seguida delega essas sub-tarefas a escravos e depois os resultados parciais são enviados dos escravos para o mestre, que tem a responsabilidade de computar o resultado final. O mestre indica um escravo para cada sub-tarefa, neste caso ele utiliza recruit-all(X) para delegar tarefas para os agentes escravos. Enquanto o escravo computa o resultado parcial da tarefa que foi nomeada pelo mestre, o mestre pode continuar seu trabalho normalmente. Quando um escravo tiver terminado seu trabalho envia uma resposta, usando reply(X), para o mestre que compila o resultado final e retorna para o cliente. A interação dos agentes do padrão é mostrada na Figura 25.

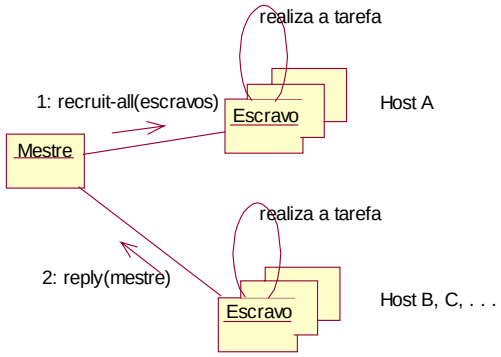

Figura 25 - Diagrama de colaboração entre os agentes do padrão Mestre-Escravo.

Sistema de padrões para o projeto de sistemas multiagente adaptativos baseados na modelagem de usuários

O APSUMAS (*Agent-based Pattern System for User Modeling and Adaptation of Systems*) é um sistema de padrões para o projeto de sistemas multiagente adaptativos e adaptáveis baseados na modelagem de usuários.

Os principais problemas no projeto da modelagem de usuários, de sistemas adaptativos e de desenvolvimento de agentes de software são analisados e descritos nos padrões. Esses padrões captam conhecimento, técnicas e problemas comuns na modelagem de usuários e adaptação de sistemas e organizam a arquitetura dos agentes em reativos ou deliberativos, segundo as características e funcionalidades específicas de cada agente.

A metodologia para a extração dos padrões que compõem o APSUMAS, é baseada na análise dos conceitos e técnicas para o desenvolvimento de sistemas adaptativos e adaptáveis; das técnicas de modelagem de usuário e dos conceitos e técnicas para o projeto de sistemas multiagente e agentes de software.

A Figura 26 mostra, em uma rede semântica, o relacionamento entre os padrões que compõem o APSUMAS. O padrão conceitual *Sociedade multiagente para sistemas adaptativos/adaptáveis* descreve os principais conceitos sobre modelagem de usuários e sistemas adaptativos/adaptáveis e justifica a escolha da abordagem de agentes para o desenvolvimento de tais sistemas. A organização dos agentes no sistema multiagente é descrita pelo padrão arquitetural *Camadas multiagente para sistemas adaptativos/adaptáveis* que distribui os agentes em camadas conforme as tarefas que serão executadas pelos mesmos.

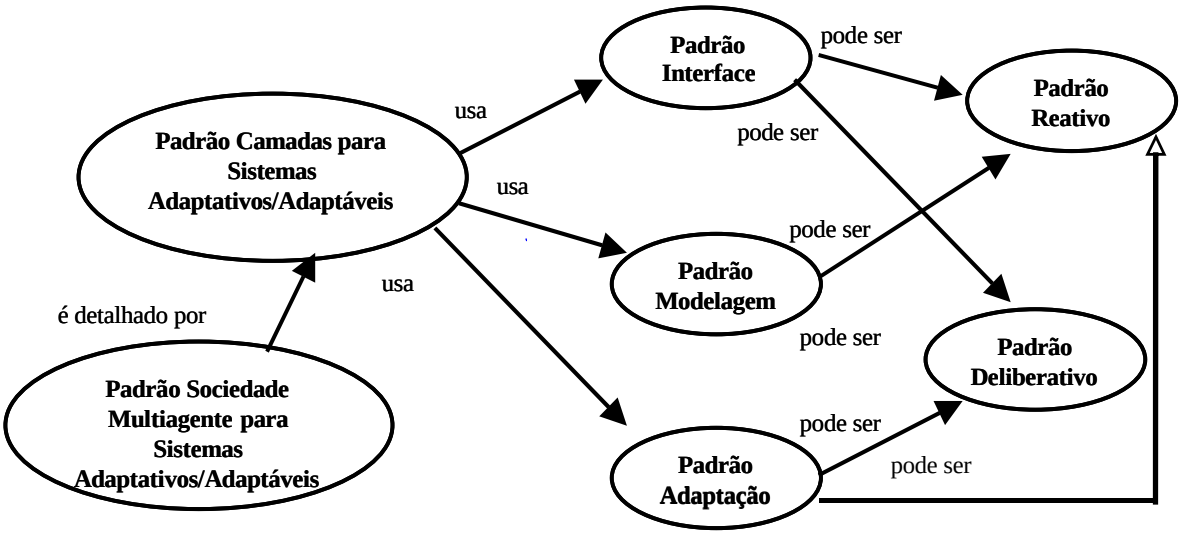

Figura 26 - Rede semântica do sistema de padrões APSUMAS

O detalhamento dos agentes que compõem as camadas é descrito pelos padrões de projeto detalhado *Interface, Modelagem e Adaptação*. Os padrões de projeto *Interface*, *Modelagem* e *Adaptação*, por sua vez, podem ser derivados do padrão *Deliberativo* ou do padrão *Reativo.*

A Tabela 4 e a Tabela 5 apresentam, de forma sucinta, alguns atributos utilizados na descrição dos padrões do APSUMAS, classificando-os segundo o tipo de padrão (conceitual, arquitetural ou de projeto detalhado) e segundo a dependência ou independência do domínio da aplicação.

Padrões baseados em Agentes					
Classificação		**Contexto**	**Problema**	**Solução**	**Padrão**
Padrões Independentes de Domínio	**Padrões de projeto detalhado**	Em sistemas multiagente com tarefas complexas, é preciso projetar agentes que estejam aptos a executar essas tarefas, exibindo comportamento direcionado a metas, tomando iniciativas e decisões por meio de mecanismos de raciocínio e/ou adaptando-se às mudanças no ambiente.	Como projetar um agente para que possa raciocinar sobre um problema de forma que o possibilite atingir metas pró-ativamente? .	O agente deve ser do tipo deliberativo: ele deve possuir um modelo do seu ambiente e de si mesmo e deve raciocinar sobre esse modelo de forma que o possibilite a criar um plano de ações para atingir suas metas.	Deliberativo
		Em sistemas multiagente com tarefas que demandam respostas rápidas durante a sua execução, é necessário projetar agentes que estejam aptos a reagir a estímulos do seu ambiente, sem conhecimento do seu ambiente nem a utilização de raciocínio lógico.	Como projetar um agente para apenas reagir a estímulos do seu ambiente?	O agente deve ser do tipo reativo: ele não possui um modelo do seu ambiente e age usando um comportamento do tipo estímulo/resposta.	Reativo

Tabela 4 - Padrões baseados em agentes para o desenvolvimento de sistemas adaptativos/adaptáveis

Padrões baseados em Agentes					
Classificação		**Contexto**	**Problema**	**Solução**	**Padrão**
Padrões Dependentes de Domínio	**Padrão Conceitual**	Esse padrão é aplicável quando há necessidade de projetar sistemas adaptativos e/ou adaptáveis que ofereçam serviços personalizados aos seus usuários, levando em consideração as diferentes características e necessidades desses usuários	Como construir e organizar sistemas que modelam características e preferências de usuários de forma a adaptar-se às suas necessidades?	Projetar e implementar um sistema multiagente onde os agentes da sociedade modelam as características e preferências de usuários e produzem os efeitos de adaptação.	Sociedade multiagente para Sistemas Adaptativos/adaptáveis
	Padrão Arquitetural	Nos sistemas adaptativos/adaptáveis, baseados na tecnologia de agentes, é preciso projetar soluções para organizar os agentes de maneira que possam satisfazer as necessidades e preferências de seus usuários.	Como estruturar um sistema multiagente para construir modelos de usuários e se adaptar a diferentes tipos de usuários ou grupos de usuários?	A arquitetura do sistema multiagente agrupa os agentes em três camadas, de acordo com as funcionalidades básicas dos sistemas adaptativos/adaptáveis: camada de interface; camada de modelagem e camada de adaptação,	Camadas Multiagente para Sistemas Adaptativos/Adaptáveis
	Padrões de projeto detalhado	Nos sistemas adaptativos/adaptáveis, baseados na tecnologia de agentes, é preciso projetar soluções para a interação do sistema com os usuários.	Como gerenciar a interação entre um sistema multiagente e seus usuários de forma personalizada?	A solução envolve a criação de um agente que servirá como interface entre o usuário e o sistema.	Interface
		Nos sistemas adaptativos/adaptáveis, baseados na tecnologia de agentes, é preciso projetar soluções para representar as características, preferências e interesses dos usuários.	Como criar e manter modelos de usuários em sistemas adaptativos?	A solução envolve a criação de um agente responsável por construir e manter um modelo de usuários.	Modelagem
		Nos sistemas adaptativos/adaptáveis, baseados na tecnologia de agentes, é preciso projetar soluções para a construção de modelos para a adaptação dos sistemas de acordo com um modelo de usuários.	Como construir modelos de adaptação para que a aplicação possa adaptar-se às necessidades dos seus usuários?	A solução envolve a criação de um agente cujo principal papel é suprir modelos de adaptação de acordo com os modelos de usuários.	Adaptação

Tabela 5 - Padrões baseados em agentes para o desenvolvimento de sistemas adaptativos/adaptáveis

A ENGENHARIA DE DOMÍNIO BASEADA EM AGENTES

A Engenharia de Domínio

A Engenharia de Domínio é uma abordagem sistemática para a construção de abstrações de software reutilizáveis "(ARANGO, 1988)" "(HARSU, 2002)".

Uma abstração de software é uma especificação de alto nível, natural e sucinta que se corresponde com uma ou várias realizações num nível de representação mais detalhado. A especificação descreve o que o artefato de software faz, eliminando os detalhes irrelevantes e enfatizando a informação que, nesse nível, é importante para o usuário da abstração "(GIRARDI, 1996)".

O processo da Engenharia de Domínio é composto das fases de análise, projeto e implementação do domínio de uma família de aplicações de software.

As atividades da Análise de Domínio identificam oportunidades de reutilização e requisitos comuns de uma família de aplicações no domínio. O produto da fase é um Modelo de Domínio. As atividades de Projeto de Domínio buscam a especificação de uma solução ao problema especificado no Modelo de Domínio. O produto desta fase é um ou mais frameworks – arquiteturas de software reutilizáveis – e uma coleção de padrões de projeto que documentam soluções bem sucedidas no domínio. Os componentes reutilizáveis do(s) frameworks elaborados na fase anterior são implementados na fase de Implementação do Domínio. Esta é a abordagem composicional da Engenharia de Domínio. Na sua abordagem gerativa, a Engenharia de Domínio produz Linguagens Específicas de Domínio (LEDs) que podem ser usadas como geradores de aplicações para construir uma família de aplicações no domínio.

Ontologias e abstrações de software

Uma ontologia "(CHANDRASEKARAN E JOSEHSON, 1999)" é a representação do vocabulário de um domínio. Mais precisamente, não é simplesmente o vocabulário como tal o que qualifica a ontologia mas os conceitos que os termos do vocabulário pretendem capturar.

As ontologias de domínio são descrições formais de classes de conceitos e relacionamentos entre esses conceitos que descrevem um domínio de aplicação. Estruturas de representação de conhecimento baseadas em frames são geralmente utilizadas para representar ontologias de domínio, onde os conceitos são representados por frames e os relacionamentos entre conceitos como slots dos frames.

As ontologias são particularmente úteis na representação de abstrações de software reutilizáveis de alto nível como modelos de domínio e frameworks. Elas fornecem uma terminologia não ambigua que pode ser compartilhada por todos os atores envolvidos em um processo de desenvolvimento. Por outro lado, uma ontologia pode ser generalizada, tanto quanto necessário, facilitando assim sua reutilização e adaptação.

Uma razão fundamental para a utilização de ontologias no processo de desenvolvimento de sistemas multiagente "(COSSENTINO *et al*., 2002)" "(DILEO, JACOBS E DELOACH, 2002)" é que elas são necessárias para viabilizar a comunicação entre agentes. Os agentes se comunicam através de mensagens que contém expressões formuladas em termos de uma ontologia. Para que os agentes possam entender o significado dessas expressões, eles precisam acessar uma ontologia comum.

A Engenharia de Domínio Multiagente

Além da complexidade das aplicações, os problemas de produtividade no desenvolvimento bem como de qualidade e manutenção dos produtos de software continuam sendo um desafio para a Engenharia de software. A reutilização de software tem apresentado boas soluções para abordar esses problemas "(GIRARDI E IBRAHIM, 1993)". Por isso, a Engenharia de software baseada em agentes deveria aproveitar os avanços bem como as lições aprendidas na teoria e prática da reutilização de software, em particular, a necessidade de dispor de abstrações de software efetivas para a reutilização "(GIRARDI, 2003)".

A abordagem multiagente tem-se mostrado particularmente apropriada para representar problemas do mundo real. Porém, a efetividade do desenvolvimento *com* reutilização dependerá de uma adequada integração das abstrações geradas na Engenharia de Domínio, bem como de mecanismos apropriados para mapear especificações em realizações, seja através de mecanismos de composição ou geração automática.

O grupo de pesquisa em Engenharia de Domínio Multiagente "(GIRARDI, 2003)" tem desenvolvido um modelo inicial para a Engenharia de Domínio Multiagente e trabalha na construção de um ambiente de desenvolvimento de software composto por um conjunto de ferramentas e bibliotecas de abstrações de software de alto nível para o desenvolvimento de SMAs. Experiências estão sendo conduzidas nos domínios jurídico e turístico para a resolução de problemas de acesso à informação “(CERVEIRA, 2003)” “(LINDOSO, SERRA E GIRARDI, 2003)”.

Desenvolvimento de aplicações multiagentes específicas

Uma aplicação multiagente específica é obtida através da composição de um conjunto de frameworks multiagente disponíveis na biblioteca do ambiente de desenvolvimento (Figura 27). Os requisitos de uma aplicação multiagente específica são usados para mapear o nível de especificação de modelos de domínio e usuários em uma realização que satisfaz essas necessidades. Essa realização está associada a um conjunto de frameworks multiagentes que representam soluções baseadas em agentes a esses requisitos.

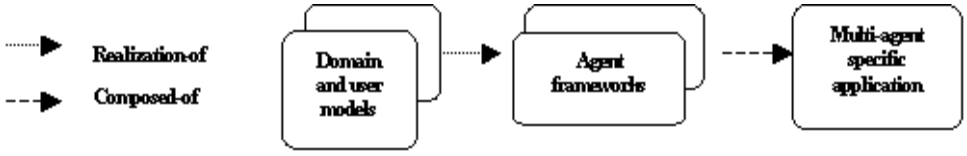

Figura 27 - O Modelo da Engenharia de Aplicações Multiagente

Desenvolvimento de abstrações de software de alto nível

A Figura 28 mostra o modelo proposto para a Engenharia de Domínio Multiagente, suas fases (Análise de Domínio, Projeto de Domínio e Implementação de Domínio) e os produtos gerados em cada fase. O conhecimento das técnicas utilizadas em cada fase bem como os produtos gerados em cada uma delas são representados em ontologias.

Na Figura 28, os produtos gerados em cada fase são ilustrados considerando seu nível de abstração (do mais abstrato ao mais concreto) e seu nível de dependência de um domínio de aplicação (do mais dependente ao mais independente): Modelos de Domínio, Modelos de Usuário, Padrões arquiteturais e de projeto detalhado, Frameworks multiagente e Agentes genéricos.

A modelagem de domínio e usuários produz a especificação de requisitos de uma família de aplicações em um domínio atendendo os perfis dos potenciais usuários do sistema. Os modelos de domínio e usuários são os produtos reutilizáveis gerados nesta fase.

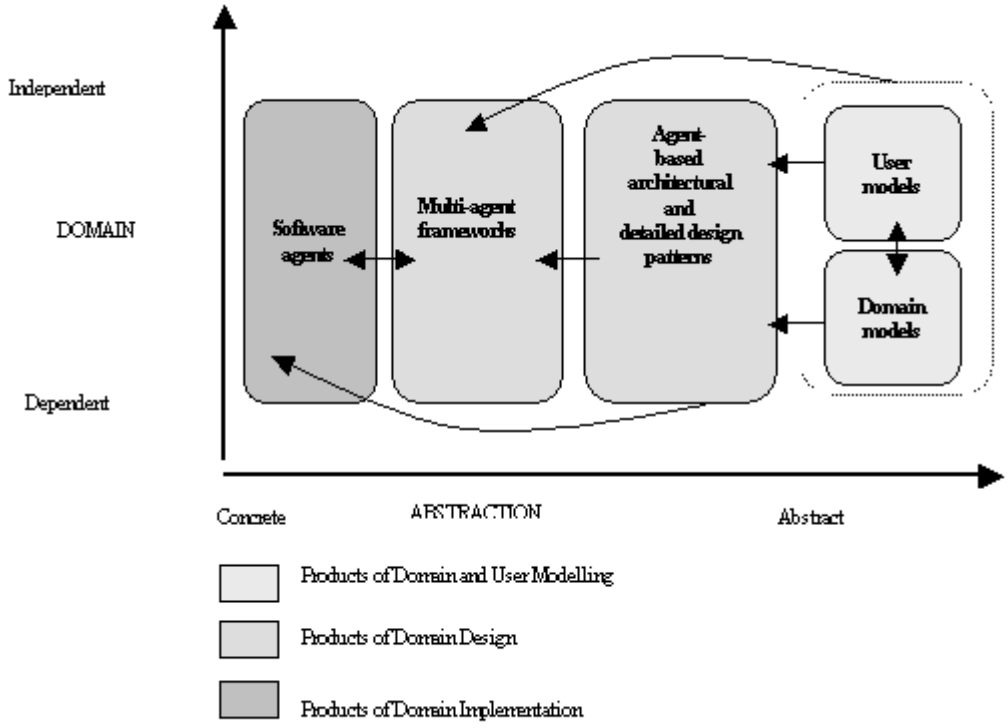

Figura 28 - Fases e produtos da Engenharia de Domínio Multiagente

O modelo de domínio - independente do domínio da aplicação e especificado em um alto nível de abstração - representa a formulação de um problema, conhecimento e atividades do mundo real. A formulação é o suficientemente genérica para representar uma família de sistemas. O modelo de usuário especifica as características, necessidades, preferências e objetivos dos usuários do sistema.

O Projeto de domínio na Engenharia de Domínio Multiagente produz uma solução computacional multiagente, reutilizável no desenvolvimento de uma família de aplicações similares em um determinado domínio. O produto desta fase consiste de um conjunto de padrões arquiteturais, padrões detalhados e frameworks multiagente.

A fase de implementação de domínio implementa os agentes componentes dos frameworks multiagente definidos no projeto de domínio.

Os produtos baseados em ontologias da Engenharia de Domínio Multiagente

No modelo proposto para a Engenharia de Domínio Multiagente, são usadas ontologias para representar o conhecimento das técnicas para a construção de modelos de domínio, modelos de usuários e frameworks multiagente.

Na Análise de Domínio e Usuários, uma ontologia genérica, ONTODUM "(FARIA, 2004)" "(FARIA, OLIVEIRA E GIRARDI, 2003)" "(GIRARDI E FARIA, 2003)", guia a construção de modelos de domínio e usuários, criados atráves da instanciação da hierarquia de meta-classes da ONTODUM. O modelo de domínio é representado em uma ontologia baseada em frames onde os conceitos, atividades e relacionamentos entre conceitos são representados em frames de acordo com o critério de representação da ONTODUM (Figura 29).

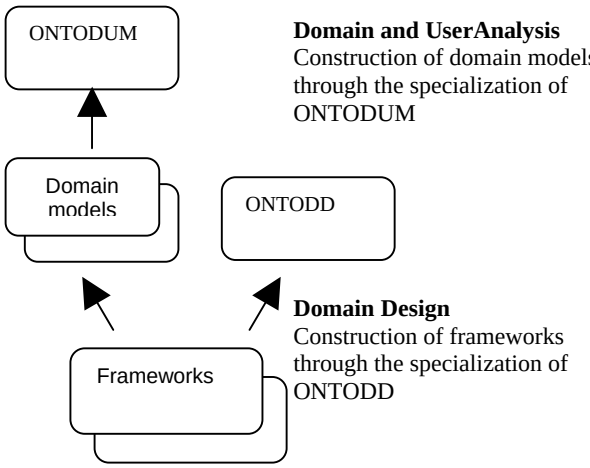

Figura 29 - Os produtos baseados em ontologias da Engenharia de Domínio Multiagente

No projeto de domínio, uma outra ontologia genérica, ONTODD "(FERREIRA, 2004)" "(FERREIRA E GIRARDI, 2003)", guia a construção de frameworks, criados através da instanciação da hierarquia de meta-classes de ONTODD, segundo os requisitos especificados em um ou mais modelos de domínio e reutilizando um conjuntos de padrões arquiteturais e de projeto detalhado. Portanto, um framework multiagente também é representado em uma ontologia baseada em frames onde a solução genérica aos requisitos especificados no (s) modelo(s) de domínio

é representada em uma estrutura de frames, de acordo ao critério de representação da ONTODD (Figura 29).

A Análise de Domínio

A técnica GRAMO "(FARIA, 2004)" define as atividades a serem realizadas na construção de modelos de domínio e usuários. A GRAMO guia a captura e especificação dos requisitos genéricos de uma família de aplicações segundo o paradigma computacional de agentes. Ela utiliza a ontologia genérica, ONTODUM, que representa o conhecimento da técnica GRAMO para a construção de modelos de domínio e usuários na Engenharia de Domínio Multiagente. A construção de modelos de domínio e usuários é feita através da instanciação da ONTODUM com o conhecimento do domínio de problema e dos usuários finais do sistema.

A técnica GRAMO consiste de duas fases: modelagem de domínio e modelagem de usuários ().

	Fases	Atividades		Produtos
TÉCNICA GRAMO	Modelagem de Domínio	Modelagem de Conceitos	Modelagem de Objetivos	Modelo de Domínio (Modelo de Conceitos, Modelo de Objetivos, Modelo de Papéis e Modelo de Interações)
			Modelagem de Papéis	
			Modelagem de Interações	
		Modelagem de Variabilidades		
	Modelagem de Usuários	Aquisição		Modelo de Usuários
		Representação		
		Manutenção		

Tabela 6 - Fases, atividades e produtos da técnica GRAMO

A fase de modelagem de domínio consiste das seguintes atividades: modelagem de conceitos, modelagem de objetivos, modelagem de papéis, modelagem de interações e modelagem de variabilidades, que geram os produtos modelo de conceitos, modelo de objetivos, modelo de papéis e modelo de interações. O produto desta fase é o modelo de domínio composto do modelo de conceitos, de objetivos, de papéis e de interações.

A modelagem de conceitos é feita paralelamente as modelagens de objetivos, papéis e interações. Na modelagem de conceitos são identificados conceitos e relacionamentos do domínio, que são representados no modelo de conceitos. Na modelagem de objetivos são identificados o objetivo geral, os objetivos específicos e as responsabilidades, que são representados no modelo de objetivos. Na modelagem de papéis são identificados os papéis e seus respectivos atributos, que são representados no modelo de papéis. Na modelagem de interações são identificadas as interações que ocorrem entre papéis ou entre papéis e entidades externas, que são representados no modelo de interações. A modelagem de variabilidades consiste em classificar as instâncias dos conceitos de objetivo geral, objetivo específico, papel, responsabilidade, atividade e recurso em conceitos fixos e variáveis.

A fase de modelagem de usuários consiste das seguintes atividades: aquisição, representação e manutenção das informações dos usuários. O produto desta fase é o modelo de usuários.

Na aquisição das informações dos usuários são identificadas as informações dos usuários. Na representação das informações dos usuários é feita a construção do modelo de usuários a partir das informações coletadas na atividade de aquisição das informações dos usuários. Na manutenção das informações dos usuários é feita a aquisição de novas informações dos usuários e atualização do modelo de usuários.

O Projeto de Domínio

A DDEMAS ("Domain design for multi-agent systems") "(FERREIRA, 2004)" "(FERREIRA E GIRARDI, 2003)" é uma técnica que busca a construção de soluções computacionais para uma família de sistemas multiagente. Através da utilização de informações contidas em um modelo de domínio em conjunto com uma coletânea de padrões é feita a construção de um framework multiagente (Figura 30).

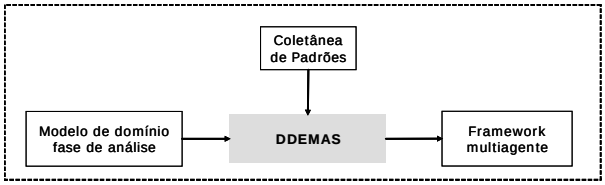

Figura 30 - Técnica DDEMAS

O modelo de domínio utilizado na técnica é criado na fase de análise através da utilização da técnica GRAMO "(FARIA, 2004)" "(FARIA E GIRARDI, 2003)". São também utilizados na construção do framework padrões contidos nas coletâneas de padrões arquiteturais e de projeto detalhado descritos na seção anterior.

A técnica DDEMAS é composta de três fases, através das quais é especificado o projeto de uma família de sistemas multiagente em ordem crescente de detalhamento: *modelagem de agentes, interações e atividades*, *projeto global* e *projeto detalhado*.

A Tabela 7 apresenta as fases, tarefas e produtos da DDEMAS.

Fases	Tarefas		Produtos
Modelagem de agentes, interações e atividades	Modelagem de agentes	Modelagem de interações e atividades	Modelo de agentes
			Modelo de interações
			Modelo de atividades
Projeto global	Construção do esboço do framework		Esboço do modelo arquitetural
	Seleção de padrão arquitetural		Modelo arquitetural
	Refinamento do framework		
Projeto detalhado	Detalhamento dos agentes		Modelo de atividades detalhado
	Seleção de padrão detalhado		Modelo de projeto detalhado
	Refinamento dos agentes		

Tabela 7 - As fases, tarefas e produtos da técnica DDEMAS

A primeira fase visa a construção dos *modelos de agentes*, *interações* e *atividades*, através dos quais são especificados os agentes e suas interações. Na segunda fase é feito o projeto global

da família de sistemas, é criado o *modelo arquitetural* representado o framework multiagente, onde os agentes são organizados segundo mecanismos de coordenação e cooperação apropriados ao contexto do problema. Na ultima fase, são construídos o *modelo de atividade detalhado* e o *modelo de projeto detalhado*. Estes modelos são referentes ao detalhamento do processamento e dados internos dos agentes que constituem o framework.

AMBIENTES DE DESENVOLVIMENTO E EXEMPLOS DE APLICAÇÕES

Várias ferramentas estão disponíveis para o desenvolvimento de SMAs, tais como: AgentBuilder "(AGENTBUILDER, 2004)", Zeus "(ZEUS, 2004)" e JADE "(JADE, 2004)".

A ferramenta AgentBuilder

AgentBuilder "(AGENTBUILDER, 2004)" fornece ferramentas gráficas para suportar todas as fases do processo da construção de um SMA. Essas ferramentas dão suporte para: organizar e controlar o projeto do desenvolvimento; analisar o domínio do problema; especificar protocolos de interação; definir a arquitetura do sistema multiagente; examinar o funcionamento da sociedade; especificar o comportamento do agente; criar agentes executáveis; e depurar erros.

O agente genérico da ferramenta *AgentBuilder*, basei-se nas arquiteturas BDI descritas previamente. Em *AgentBuilder*, um agente é caracterizado pelos seguintes conceitos que conformam o chamado *modelo mental* do agente "(AGENTBUILDER, 2004)":

- **Estado -** representa o estado atual do ambiente do agente. Esse estado é atualizado conforme as percepções do agente.
- **Habilidades -** descreve as ações que o agente pode executar quando condições pré-definidas são satisfeitas.
- **Negociação -** é um acordo feito com outro agente para que uma ação seja executada em um determinado instante.
- **Regras de comportamento -** podem ser vistas como sentenças do tipo *quando – se - então*. O *quando* da regra se aplica a novos eventos ocorrendo no ambiente de um agente, como novas mensagens recebidas de outros agentes. O *se* compara o estado atual com condições que são requeridas para uma regra ser aplicável. O *então* define as ações do agente executadas em resposta a um evento.
- **Intenções -** são similares a negociações, em que um agente executa ações em nome de outro agente. No entanto, uma negociação é um acordo para executar uma ação simples, já uma intenção é um acordo para executar quaisquer ações que são necessárias para atingir um estado desejado do ambiente.

O conjunto dessas características forma a Arquitetura do *Agente Inteligente Reticular*, a arquitetura do agente genérico da ferramenta *AgentBuilder*, ilustrada na Figura 31. Um interpretador monitora continuamente as mensagens que chegam, atualiza o modelo mental do agente e executa ações. No começo do processo, o agente é inicializado com crenças, intenções, habilidades e regras de comportamento iniciais. Um agente não trivial requer pelo menos uma regra de comportamento, os outros elementos são opcionais. O modelo mental contém as crenças, intenções, habilidades e regras atuais do agente "(AGENTBUILDER, 2004)".

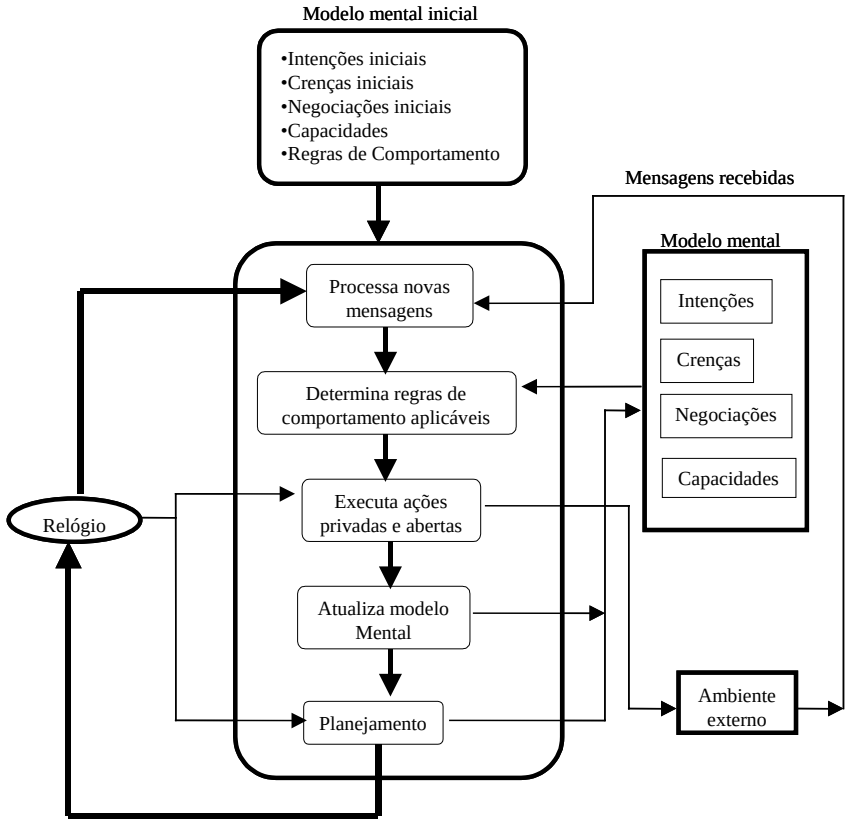

Figura 31 - Arquitetura do agente genérico da ferramenta AgentBuilder "(AGENTBUILDER, 2004)"

A Ferramenta Zeus

A ferramenta ZEUS consiste de um conjunto de componentes, escritos na linguagem de programação Java, que podem ser categorizados em três grupos funcionais (ou bibliotecas): uma *biblioteca de componentes agentes*, *uma ferramenta para a construção de agentes* e *um grupo de agentes de utilidade* que incluem os agentes *nameserver, facilitator* e *visualiser* "(ZEUS, 2004)".

O agente genérico Zeus inclui os seguintes componentes (Figura 32):

- Uma **caixa de correio** que manipula as comunicações entre o agente genérico e outros agentes.
- Um **manipulador de mensagens** que processa as mensagens que chegam na caixa de correio, despachando-as para os componentes relevantes do agente genérico.
- Um **mecanismo de coordenação** que toma decisões a respeito das tarefas dos agentes e também é responsável pela coordenação das interações com outros agentes.
- Um **banco de dados de conhecimento** que descreve as relações do agente genérico com outros agentes na sociedade e suas crenças sobre as habilidades desses agentes.
- Um **planejador e escalonador** que planeja as tarefas dos agentes, baseadas nas decisões tomadas pelo mecanismo de coordenação.
- Um **banco de dados de recursos** que mantém uma lista de recursos que pertencem e estão disponíveis ao agente genérico.
- Um **banco de dados de ontologias** que armazena a definição lógica de cada tipo de atributo.
- Um **banco de dados de tarefas/planos** que fornece uma descrição lógica das tarefas disponíveis nos agentes.
- Um **monitor de execução** que é responsável pelo relógio interno do agente genérico. Ele inicia, suspende e monitora as atividades programadas pelo *Planejador/Escalonador*. Também informa ao *Planejador* sobre as condições de término satisfatórias e excepcionais das tarefas que estão sendo monitoradas.

A Ferramenta JADE

JADE (Java Agent Development framework) "(JADE, 2004)" "(FIPA, 2004)" é um ambiente para desenvolvimento de aplicações baseadas em agentes conforme as especificações da FIPA (Foundation for Intelligent Physical Agents) para interoperabilidade entre sistemas multiagentes. JADE é implementado em Java.

O modelo computacional do agente é multitarefa, várias tarefas podem ser executadas concorrentemente. Cada funcionalidade provida por um agente, pode ser implementada com a integração de mais de um comportamento. A definição da ordem de execução das tarefas é feita pelo Gerenciador de Tarefas (Figura 33**)**.

O agente genérico do ambiente JADE é uma arquitetura reativa segue a estrutura mostrada na Figura 33. A arquitetura interna de um agente genérico em JADE é composta dos seguintes módulos "(JADE, 2004)":

- **Comportamentos** – representam as ações, intenções que cada agente possui. Cada funcionalidade ou serviço fornecido por um agente na realidade é um ou mais comportamentos que ele tem. Entretanto, só é permitido ao agente executar seus comportamentos se ele estiver no seu estado ativo.

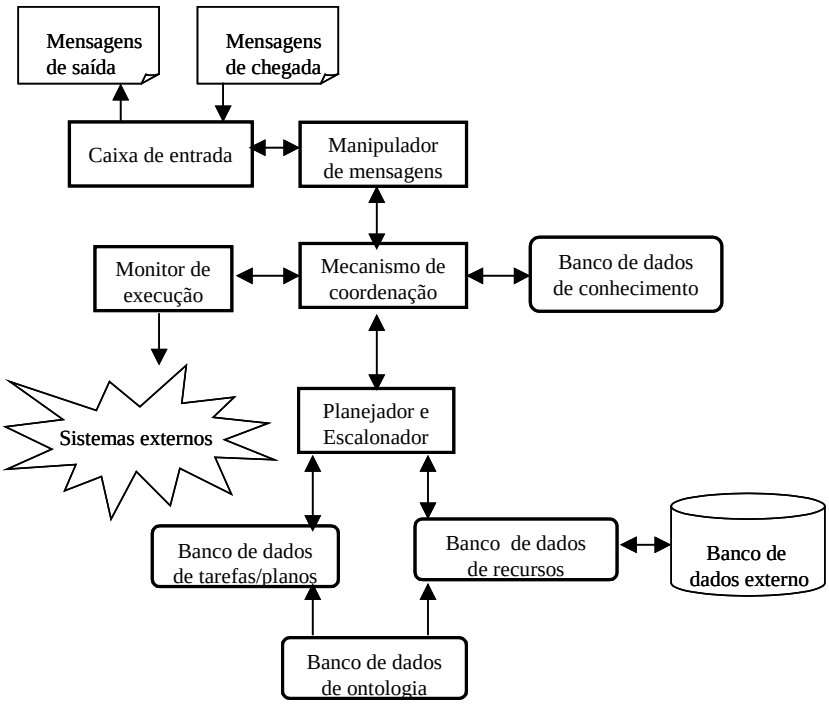

Figura 32 - Arquitetura do agente genérico Zeus "(ZEUS, 2004)"

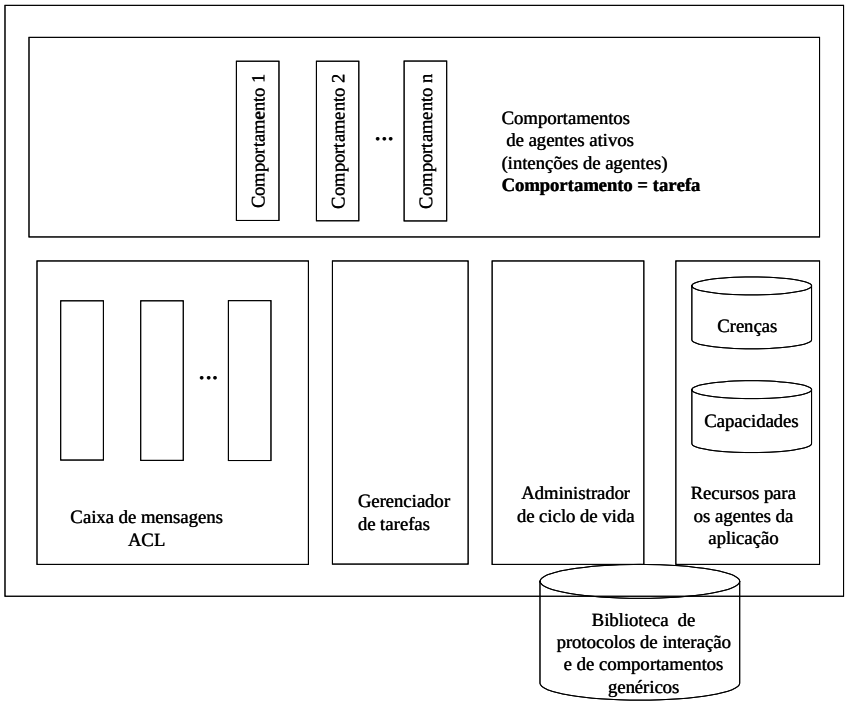

Figura 33 - Arquitetura do agente genérico JADE

- **Caixa de mensagens** – é uma estrutura de dados onde ficam armazenadas todas as mensagens recebidas pelo agente.
- **Escalonador de comportamentos** – responsável por decidir em que ordem os comportamentos deverão ser executados.
- **Gerenciador de ciclo de vida** – é o controlador do estado atual do agente, ou seja, é ele que determina o estado atual do agente (inicial, ativo, suspenso, espera, transitório, destruído, copia, movido).
- **Recursos da Aplicação** – representa o conhecimento adquirido pelo agente durante a execução da aplicação.

O JADE não habilita agentes com habilidades específicas além daquelas necessárias para comunicação e interação. No entanto, o modelo de agentes permite a integração de programas externos com os agentes de tarefas. A biblioteca existente em JADE inclui uma classe, chamada *JessBehaviour*, que permite o uso da linguagem *JESS* como mecanismo de raciocínio do agente. O *JESS* é bastante útil no controle da ativação e desativação dos comportamentos em JADE, o que permite a implementação de uma arquitetura formada por agentes reativos e deliberativos (na qual o *JESS* executa os papéis deliberativos e JADE, os reativos). No futuro, os desenvolvedores de JADE prometem o acréscimo de uma ferramenta visual para compor os comportamentos de agentes e oferecer uma arquitetura de mais alto nível.

AGRADECIMENTOS

Este trabalho é apoiado pelo CNPq.

REFERÊNCIAS

AGENTBUILDER USER'S GUIDE, Reticular Systems, external documentation, Disponível em: http://www.agentbuilder.com/, Acessado em: março de 2004.

ARANGO, G. **Domain Engineering for Software Reuse**. Ph.D. Thesis, Department of Information and Computer Science, University of California, Irvine, 1988.

BELLIFEMINE, F., RIMASSA, G., and POGGI, A. **Jade – a fipa-compliant agent framework**. In Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, UK, pp. 97–108, 1999.

BRADSHAW, M. J. **Software Agents**, The AAAI Press, 1997.

BROOKS, R. A. **Intelligence without Representation**, *Artificial Intelligence*, v. 47, pp. 139-159, 1991.

BROOKS, R. A. **A robust layered control system for a mobile robot,** *IEEE Journal of Robotics and Automation*, v. 2 (1), pp. 14-23, 1986.

CAIRE, G., LEAL, F., CHAINHO, P., EVANS, R., GARIJO, F., GOMEZ, J., PAVON, J., KEARNEY, P., STARK, J., COULIER, W., and MASSONET, P. **Agent-Oriented Analysis using MESSAGE/UML**, In Proceedings of the 2ª International Workshop on Agent-Oriented Software Engineering (AOSE 2001), Montreal, Canada, Springer-Verlag, pp. 119-135, 2001.

CASTRO, J., KOLP, M., and MYLOPOULOS, J. **A Requirement-Driven Software Development Methodology**, In Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAISE 2001), Interlaken, Switzerland, Springer-Verlag, pp. 108-123, 2001.

CERVEIRA, N. **Desenvolvimento de um Modelo de Domínio baseado em Ontologias para a Área Turística**, Monografia de Especialização, CEAPS/DEINF/UFMA, 2003.

CHANDRASEKARAN, B. and JOSEHSON, J. **What Are Ontologies, and Why Do We Need Them?**, *IEEE Intelligent Systems*, v. 14 (1), pp. 20 – 26, 1999.

CHAUHAN, D. **JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation,** PhD thesis, ECECS Department Thesis, University of Cincinnati, Cincinnati, OH, 1997.

COSSENTINO, M., BURRAFATO, P., LOMBARDO, S., and SABATUCCI, L., **Introducing Pattern Reuse in the Design of Multi-agent Systems**, In Proceedings of the Agent Technologies, Infrastructures, Tools, and Applications for E-Services (NODe 2002), Agent-Related Workshops, Efurt, Germany, Springer-Verlag, pp. 107-120, 2002.

DILEO, J., JACOBS, T., and DELOACH, S. **Integrating Ontologies into Multi-Agent Systems Engineering**, In Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002), Bologna, Italy, 2002.

FARIA, C. **Uma Técnica para a Aquisição e Construção de Modelos de Domínio e Usuários baseados em Ontologias para a Engenharia de Domínio Multiagente**, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2004.

FARIA, C., OLIVEIRA, I., e GIRARDI, R. **Especificação de uma Ontologia Genérica para a Construção de Modelos de Usuários**, Anais da 3ª Joranada Ibero-Americana de Engenharia de Software e Engenharia do Conhecimento (JIISIC 2003), Valdivia, Chile, Ed. LOM Ediciones Ltda., pp. 93-103, 2003.

FERREIRA, S. **Uma Ferramenta e Técnica para o Projeto Global e Detalhado de Sistemas Multiagente**, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2004.

FERREIRA, S., e GIRARDI, R. **Especificação de uma Ontologia Genérica para o Projeto de Domínio de Aplicaçoes Multiagente**, Anais do III Workshop de Ingeníeria de Software (WIS 2003) nas Jornadas Chilenas de Computação, Chilán, Chile, 2003.

FIPA – Foundation for Intelligent Physical Agents, Disponível em: http://www.fipa.org, Acessado em: março de 2004.

JADE - Java Agent Development Framework, Disponível em: http://jade.cselt.it, Acessado em: março de 2004.

FININ, T., LABROU, Y., and MAYFIELD, J. **KQML as an agent communication language,** Computer Science Department. University of Maryland Baltimore County. Baltimore, USA, 1993.

FOWLER, M. and SCOTT, K. UML Essencial: Uma breve guia para a linguagem – padrão de modelagem de objetos, Porto Alegre, Brasil, Bookman, 2000.

GENESERETH, M. R., and KETCHPEL, S. P. **Software Agents,** *Communications of the ACM*, v. 37(7), 1997.

GIRARDI, R. **Engenharia de Domínio Multiagente**, Anais do 3 Ibero-Americano Symposium on Software Engineering and Knowledge Engineering (JIISIC 2003), Scientific Cooperation, Universidad Austral de Chile, 2003.

GIRARDI, R, and FARIA, C. **A Generic Ontology for the Specification of Domain Models**, In Proceedings of the 1st International Workshop on Component Engineering Methodology (WCEM 2003) at Second International Conference on Generative Programming and Component Engineering, Efurt, Germany, Ed. Sven Overhage and Klaus Turowski, pp. 41-50, 2003.

GIRARDI, R., OLIVEIRA, I., and SILVA, G.B. J. **Towards a System of Patterns for the Design of Agent-based Systems,** In Proceedings of The Second Nordic Conference on Pattern Languages of Programs (VikingPLoP 2003), Bergen, Norway, 2003.

GIRARDI, R. **Reuse in Agent-based Application Development**, In: Proceedings of the 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS´2002) at International Conference on Software Engineering (ICSE´2002), Orlando, Florida, 2002.

GIRARDI, R. **Agent-Based Application Engineering**, In Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001), Setubal, Portugal, 2001.

GIRARDI, R. **An Analysis of the Contributions of the Agent Paradigm for the Development of Complex Systems**, In Joint Meeting of The 5th World Multiconference On Systemics, Cybernetics and Informatics (SCI 2001), Orlando, Florida, 2001.

GIRARDI, R, and SODRÉ, A. C. S. **A Methodology for Multi-Agent Application Development,** In 6th International Conference on Intelligent Tutoring Systems, Proceedings of the ITS´2002 Workshops - Architectures and Methodologies for Building Agent-Based Learning Environments, Biarritz, v. 1, pp. 58-66, 2002.

GIRARDI, R., and FERREIRA, S. **Arquiteturas de software baseadas em agentes**, Anais da IV Escola de Informática da Região Norte **(**EIN 2002**)**, Belem, Macapa, Palmas, 2002.

GIRARDI, R. **Towards Effective Software Abstractions for Application Engineering,** In: NASA Focus on Reuse Workshop, Fairfax, Virginia, 1996.

GIRARDI, R., and IBRAHIM, B. **New Approaches for Reuse Systems**, In: Position Paper Collection of the 2nd. International Workshop on Software Reuse (IWSR-2), 1993.

GLASSER, N. **The CoMoMAS Approach: From Conceptual Models to Executable Code,** In Proceedings of the 8th European Workshop on Modelling of Autonomous Agents in a Multi-Agent World (MAAMAW 1997), Ronneby, Suécia, 1997.

HARSU M. **A Survey on Domain Engineering**, Tech. Rep. 31, Institute of Software Systems, Tampere University of Technology, 2002.

HUHNS, N., and STEPHENS, L. M. **Multi-Agent Systems and Societies of Agents**, *In: Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, G. Weiss (ed.), The MIT Press, 1999.

HYACINTH, S. N. **Software Agents: An Overview**, *Knowledge Engineering Review*, v. 11(3), pp. 205-244, 1996.

IGLESIAS, C. A., GARIJO, M., and GONZALEZ, J. C. **Analysis and Design of Multiagent Systems using MAS-CommonKADS,** Intelligent Agents IV (ATAL 1997), Lecture Notes in Artificial Inteligence, Berlin, Alemanha, Springer-Verlag, pp. 314-327, 1998.

IGLESIAS, C. A., GARIJO, M., and GONZALEZ, J. C. **A Survey of Agent-Oriented Methodologies,** In Proceedings of the 5th International Workshop on Intelligent Agents V: Agent Theories, Architectures and Languages (ATAL 1998), Heidelberg, Alemanha, Springer-Verlag, pp. 317-330, 1999.

IGLESIAS, C. A., GARIJO, M., GONZALEZ, J. C., and VELASCO, J. R. **A Methodological Proposal for Multiagent Systems Development Extending CommonKADS,** In Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW 1996), Banff, Canadá, pp. 25/1-17, 1996.

Jat Lite. Disponível em: http://java.stanford.edu/, Acessado em: março de 2004.

JENNINGS, N. R. **On Agent-based Software Engineering**, *Artificial Intelligence*, v. 117, pp. 277-296, 2000.

JENNINGS, N. R**.,** SYCARA, K., and WOOLDRIDGE, M. **A Roadmap of Agent Research and Development**, *Autonomous Agents and Multi-Agent Systems*, v.1(1), pp.7-38**,** 1998**.**

KNAPIK, M., and JOHNSON, J. **Developing Inteligent Agents for Distributed Systems**, Computing McGraw-Hill, NY:McGraw-Hill, 1998.

LEMOS, A. J. D., GIRARDI, R., e FERREIRA, S. **ABARFI: uma arquitetura reutilizável para a recuperação e filtragem de informação,** Anais da 2da. Jornada Ibero-americana da Engenharia de Software e Engenharia de Conhecimento. Salvador, Bahia, Brasil, UNIFACS, 2002.

LEMOS, A. J. D. **Uma Arquitetura baseada em Agentes para a Recuperação e Filtragem de Informação**, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2001.

LINDOSO, A., SERRA, I., e GIRARDI, R. **ONTOINFOJUS: Um Modelo de Domínio baseado em Ontologias para o Acesso à Informação na Área Jurídica**, Anais do V Encontro de Estudantes de Informática do Tocantins (ENCOINFO 2003), Palmas, Tocantins, Brasil, Ed. ULBRA, pp. 251-260, 2003.

MYLOPOULOS, J., and CASTRO, J. **Tropos: A Framework for Requirements-Driven Software Development**, Brinkkemper, J. and Solvberg, A. (eds.), Information Systems Engineering: State of the Art and Research Themes, Berlin, Alemanha, Springer-Verlag, pp. 261-273, 2000.

NWANA, H. S. **Software Agents: An Overview**, *Knowledge Engineering Review*, v. 11(2), pp. 205-244, 1996.

ODELL, J., PARUNAK, H. D., and BAUER, B. **Extending UML for agents,** In Proceedings of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2000), Austin, USA, pp. 3–17, 2000.

OLIVEIRA DE ALMEIDA, H. COMPOR: Desenvolvimento de Software para Sistemas Multiagente, Dissertação (Mestrado em Ciência da Computação), Departamento de Ciência da Computação, Universidade Federal de Campina Grande – UFCG, 2004.

OLIVEIRA, I. **Um Sistema de Padrões baseados em Agentes para a Modelagem de Usuários e Adaptação de Sistemas**, Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2004.

OLIVEIRA, I., e GIRARDI, R. **Padrões Arquiteturais e de Projeto para a Modelagem de Usuários baseada em Agentes,** Anais da Terceira Conferência Latinoamericana em Linguagens de Padrões para Programação (SugarLoafPLOP 2003). Porto de Galinhas, Pernambuco, Brasil, 2003.

OMICINI, A. **"SODA Societies and Infrastructures in the Analysis and Design of Agent-based Systems"**, Proceedings of First International Workshop, AOSE 2000 on Agent-Oriented Software Engineering, Limerick, Ireland, pp. 185-193, January, 2001.

PARAISO, E. C. **Concepção e Implementação de um Sistema Multiagente para Monitoração e Controle de Processos Industriais,** Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial), CEFET - PR, 1997.

RAO, A. S., and GEORGE, M. P. **BDI Agents: From Theory to Practice,** In Proccedings of the First International Conference on Multi-Agent Systems (ICMAS 1995), São Francisco, MIT Press, pp. 312-319, 1995.

RUSSEL, S. and NORVIG, P. **Artificial Intelligence: A Modern Approach**, Prentice-Hall, 1995.

SILVA, F. J. H. A., e GIRARDI, R. **SIMCAP: um sistema multiagente para a captura de publicações científicas na web**, *Revista Eletrônica de Iniciação Científica da SBC*, Março de 2002.

SILVA, G. B. J. **Padrões Arquiteturais para o Desenvolvimento de Aplicações Multiagente,** Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2003.

SODRÉ, A. **Metodologia baseada em agentes para o desenvolvimento de software – MADS,** Dissertação (Mestrado em Engenharia de Eletricidade) – Área de Ciência da Computação, Departamento de Engenharia Elétrica, Universidade Federal do Maranhão – UFMA, 2002.

STEELS, L. **Cooperation between distributed agents through self organization**, In Y. Demazeau and J. P. Müller, editors, "Descentralized AI – Proceeding of the First European Wrokshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-1989)", Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, pp. 175-196, 1990.

SYCARA, K. **Multiagent Systems**, *AI Magazine*, v. 19 (2), pp. 79-92, 1998.

WOOD, M., and DELOACH, S. **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 Cicarini, M. Woodridge, editors. Lecture Notes in Computer Science, v. 1957, Springer Verlag, Berlin, 2001.

WOOD, M., DELOACH, S., and SPARKMAN, C. H. **Multi-Agent System Engineering,** *The International Journal of Software Engineering and Knowledge Engineering*. v. 11(3), June 2001.

WOOLDRIDGE, M. **An Introduction to Multi-Agent Systems,** John Wiley & Sons Ltd. Chichester, Inglaterra, 2002.

WOODRIDGE, M. **Reasoning about Rational Agents,** The MIT Press, Cambridge, Masachusetts.

WOOLDRIDGE, M., and CICARINI P. **Agent-Oriented Software Engineering: The State of the Art,** In Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering, Berlin, Alemanha, Springer-Verlag, pp. 1-28, 2000.

WOOLDRIDGE, M., JENNINGS, N., and KINNY, D. **The Gaia Methodology for Agent-Oriented Analysis and Design,** *International Journal of Autonomous Agents and Multi-Agent Systems*, v. 3, 2000.

WOOLDRIDGE, M. **Intelligent Agents,** In: Multi-Agent Systems - A Modern Approach to Distributed Artificial Intelligence, G. Weiss (ed.), The MIT Press, 1999.

WOOLDRIDGE, M., and JENNINGS, N. **Intelligent Agents: Theory and Practice.** *The Knowledge Engineering Review*, v. 10 (2), pp. 115-152, 1995.

WOOLDRIDGE, M. **Agent-Based Software Engineering,** In Proceedings IEE Software Engineering, v. 144 (1), pp. 26-37, 1997.

WOOLDRIDGE, M., and JENNINGS, N. **Agent Theories, Architectures and Languages: A Survey,** In Proceedings of Workshop on Agent Theories Architectures, and Languages (ECAI 1994), Lecture Notes in Artificial Intelligence, Amsterdam, Springer-Verlag, pp. 01-32, 1994.

ZEUS DOCUMENTATION, http://more.btexact.com/projects/agents.htm, março de 2004.