Capítulo 5

Engenharia de Software Orientada a Agentes

Jaelson Castro, Fernanda Alencar e Carla Silva

Abstract

Software agents or multi-agent systems have recently attracted considerable attention of the software industry. The objective of this chapter is to introduce this new software development paradigm, which is suitable for developing complex software systems. We discuss why the agent-oriented approach is a genuine improvement to the state of the art. An overview of the main concepts, methods, and existing tools is presented. We describe in detail the Tropos framework. The principal challenges of the agent-oriented paradigm for software development are discussed.

Resumo

Agentes de software ou sistemas multi-agentes tem recentemente atraído uma considerável atenção da indústria de software. O objetivo deste capítulo é introduzir este novo paradigma de desenvolvimento que é adequado para o desenvolvimento de complexos sistemas de software. Debateremos porque que a abordagem orientada a agente é um genuíno avanço em relação ao estado da arte. Uma visão geral dos principais conceitos, métodos e ferramentas existentes é apresentada. Descreveremos em detalhe o framework Tropos. Os principais desafios do paradigma de agentes para o desenvolvimento de software são discutidos.

5.1. Introdução

Atualmente, as aplicações industriais de software têm crescido no tamanho e na complexidade, fazendo com que seu projeto e construção se torne uma tarefa bastante difícil. Esta situação tem motivado os engenheiros de software a obterem um melhor entendimento das características de um software complexo, reconhecendo a interação como um de seus aspectos mais importantes. De fato, muitos pesquisadores buscam novos paradigmas para aumentar nossa habilidade de modelar, projetar e construir complexos sistemas de software de natureza distribuída [Wooldridge 2002].

Um dos mais promissores paradigmas do momento é a chamada orientação a agentes. Um agente é um sistema computacional capaz de ações flexíveis e autônomas em um ambiente dinâmico, aberto e imprevisível [Jennings and Bussmann 2003]. É a naturalidade e a facilidade com que uma variedade de aplicações pode ser caracterizada em termos de agentes que levam pesquisadores e desenvolvedores a ficarem tão entusiasmados sobre o potencial deste paradigma.

A tecnologia de agentes tem um grande potencial para se tornar uma solução de engenharia de software popular. No entanto, para o sucesso de sua adoção é necessário que tanto os indivíduos quanto as organizações se tornem mais familiarizados e confiantes com a noção de agentes de software autônomos [Jennings and Wooldridge 2001]. Este capítulo é um passo nesta direção.

Este capítulo está organizado da seguinte forma: na Seção 2 apresentamos a engenharia de software orientada a agentes. Já na Seção 3 descrevemos algumas abordagens de desenvolvimento de sistemas multi-agentes. Uma das mais promissoras abordagens, o *framework* Tropos é detalhado na Seção 4. Concluímos o capítulo discutindo o estado atual da engenharia de software orientada a agentes e indicando os desafios futuros da utilização deste novo paradigma.

5.2. Engenharia de Software Orientada a Agentes

Esta seção aborda a engenharia de software orientada a agentes. Inicialmente mostraremos as motivações para o uso desse novo paradigma, seguido dos aspectos e propriedades que caracterizam um agente de software. Também abordaremos as motivações e a importância dos sistemas baseados em agentes para a engenharia de software. Além disso, apresentaremos as principais áreas de aplicação dos softwares orientados a agentes. Finalmente, apontamos os desafios enfrentados atualmente no desenvolvimento de sistemas orientados a agentes.

5.2.1. Motivação do Paradigma de Agentes

Projetar e implementar software industrial de alta qualidade é difícil. De fato, assume-se que a tarefa de desenvolver tais projetos está entre as mais complexas tarefas de construção realizadas por humanos. Diante disso, vários paradigmas de implementação foram propostos na literatura (e.g., programação estruturada, programação declarativa, programação orientada a objetos e baseada em componentes). Esta evolução incremental dos paradigmas visa tornar a engenharia de software mais gerenciável, como também permite aumentar a complexidade das aplicações que podem ser construídas.

Na visão do usuário, software de qualidade [ISO 9126 1996] é aquele que além de satisfazer as suas necessidades, é feito no custo e prazo combinados e apresenta um padrão mínimo de qualidade. Já na visão do engenheiro de software, um bom software é aquele que apresenta as seguintes características [Sauvé 1999]:

- *Maior flexibilidade* - Possibilita satisfazer novos requisitos de negócio (funcionalidade) de forma fácil e rápida.
- *Melhor adaptabilidade* - Possibilita personalizar uma aplicação para vários usuários, usando várias alternativas para oferecer os serviços da aplicação com o mínimo de impacto no seu núcleo.
- *Melhor manutenibilidade* - Possibilita alterar partes de uma aplicação, de modo que as outras partes sofram um impacto mínimo.
- *Melhor reusabilidade* - Possibilita montar aplicações únicas e dinâmicas rapidamente.
- *Melhor aproveitamento do legado* - Possibilita reusar a funcionalidade de sistemas legados em novas aplicações.
- *Melhor interoperabilidade* - Possibilita integrar duas aplicações que executam em plataformas diferentes.
- *Melhor escalabilidade* - Possibilita distribuir e configurar a execução da aplicação de modo a satisfazer a vários volumes de transação.
- *Melhor robustez* - Possibilita implementar soluções de software com menos defeitos.

Além disso, os engenheiros de software consideram um bom processo de desenvolvimento de software aquele que apresenta as seguintes características [Sauvé 1999]:

- *Menor tempo de desenvolvimento* – Possibilita construir novos sistemas de forma mais rápida e com baixo orçamento.
- *Menor risco* - Possibilita todas as características de um software de qualidade, sem ter o risco de ter projetos fracassados.

A orientação a objetos é um paradigma bastante popular no desenvolvimento de software, porém foram realizadas muitas extensões deste paradigma para conseguir realizar, com menos esforço, algumas das propriedades de um software de qualidade. Por exemplo, extensões da orientação a objetos para suportar sistemas distribuídos [SUN 2005], para lidar melhor com características transversais [Kiczales et al. 1997], para desenvolver sistemas em tempo-real [Bihari et al. 1989] e assim por diante.

Neste contexto, nos últimos anos os pesquisadores têm considerado novas abordagens, tal como o paradigma de agentes, a fim de melhorar significantemente o processo de desenvolvimento de software complexo. Técnicas orientadas a agentes representam um novo meio de analisar, projetar e construir sistemas de software complexos. O uso da abordagem orientada a agentes é baseado nos seguintes argumentos [Jennings and Wooldridge 2001]: (i) o aparato conceitual de sistemas orientados a agentes é adequado para construir soluções de software para sistemas complexos e (ii) as abordagens orientadas a agentes representam um verdadeiro avanço sobre o estado da arte na engenharia de sistemas complexos.

De fato, técnicas orientadas a agentes são adequadas para desenvolver sistemas complexos de software porque [Jennings and Wooldridge 2001]:

- As decomposições orientadas a agentes são uma maneira efetiva de repartir o espaço do problema de um sistema complexo;
- As abstrações chave presentes no modo de pensar orientado a agentes são um meio natural de modelar sistemas complexos;
- A filosofia orientada a agentes para identificar e gerenciar relacionamentos organizacionais é apropriada para lidar com as dependências e interações que existem em um sistema complexo.

Na próxima seção, apresentaremos os principais conceitos que caracterizam um agente.

5.2.2. Propriedades de Agentes de Software

Embora muitas perspectivas diferentes sobre agentes têm sido descritas e discutidas, não há uma definição universalmente aceita sobre o que exatamente caracteriza um agente. Entretanto, há muitas definições para o termo de agente na literatura, incluindo:

> *"Internamente, um agente é descrito por uma função f, que captura percepções e recebe mensagens como entrada, gerando saídas na forma de execução de ações e envio de mensagens. O mapeamento f em si não é diretamente controlado por uma autoridade externa" [Müller 1996]*
> *"Agentes são unidades de software que podem lidar com mudanças ambientais e com os vários requisitos de redes abertas através de características como autonomia, mobilidade, inteligência, cooperação e reatividade" [Tahara et al. 1999]*
> *"Internamente, um agente é um componente computacional com qualidades mentais atribuídas, que incluem crença, desejo, objetivo, intenção e compromisso. O agente de computação toma decisões de resolução, planejamento e execução baseadas na manipulação simbólica destas qualidades mentais, a fim de se comprometer com suas obrigações para realizar uma tarefa final" [Li et al. 2000]*

Neste trabalho, nós usamos a seguinte definição de agente [Wooldridge 2002]:

> *"Um agente é um sistema computacional encapsulado que está situado em algum ambiente e é capaz de ação flexível autônoma neste ambiente, a fim de alcançar seus objetivos de projeto"*

Baseado em nossa pesquisa bibliográfica (cf. [Weiss 2000, Sturm and Shehory 2003, Green et al. 1997, Müller 1998, Wooldridge and Jennings 1995]), estabelecemos as seguintes propriedades de agente como sendo aquelas que precisam ser consideradas a fim de construir um sistema de software orientado a agentes [Silva et al. 2004]:

- *Autonomia:* habilidade de agir de forma independente sem intervenção direta de humanos ou de outros agentes. Entidades autônomas ativas não estão necessariamente de acordo com desejos ou solicitações externas [Yu 2002].
- *Deliberatividade:* um agente deliberativo toma decisões considerando tanto as informações vindas do ambiente onde ele se situa, como as informações sobre experiências prévias. Tais agentes estão aptos a gerar objetivos e agir de forma racional para atingi-los.
- *Reatividade:* agentes percebem seu ambiente e respondem a tempo as mudanças que ocorrem. Nota-se que agentes reativos só agem em resposta a estímulos externos.
- *Organização:* Uma organização consiste de um grupo de agentes cujo comportamento e relacionamentos são regulados por um conjunto de normas sociais que pretendem criar uma unidade capaz de atingir um objetivo comum [Dignum and Dignum 2001].
- *Habilidade de socializar-se*: habilidade de participar em relacionamentos múltiplos, interagindo com vários outros agentes, simultaneamente ou em momentos diferentes [Yu 2002].
- *Interação:* habilidade de se comunicar com o ambiente e com outros agentes: A interação de agentes pode ser classificada como:
  - *Comunicação*: habilidade de trocar mensagens com outros agentes.
  - *Cooperação*: habilidade de interagir com outros agentes para atingir um objetivo comum; acontece entre agentes não antagonistas que têm sucesso ou falham juntos.
  - *Competição*: habilidade de interagir com outros agentes onde o sucesso de um agente implica na falha de outros (o oposto de cooperação).
  - *Coordenação*: habilidade de executar alguma atividade em um ambiente compartilhado com outros agentes, determinando objetivos que eles compartilham e tarefas em comum, evitando conflitos desnecessários e gerenciando a utilização dos recursos [Weiss 2000].
  - *Negociação:* habilidade de interagir com outros agentes a fim de alcançar um acordo sobre algum problema [Green et al. 1997].

Na próxima seção destacaremos a importância e a motivação dos sistemas multi-agentes.

5.2.3. Importância dos Sistemas Multi-Agentes

Uma questão óbvia é por que sistemas multi-agentes são vistos como

uma nova direção importante na engenharia de software. Agentes são importantes/úteis porque [Odell 2000]:

- provêm uma maneira de pensar sobre o fluxo de controle em um sistema altamente distribuído,
- oferecem um mecanismo que permitem um comportamento emergente ao invés de uma arquitetura estática,
- codificam melhores práticas de como organizar entidades colaborativas concorrentes.

Existem várias razões que justificam o aumento de interesse na pesquisa de sistemas multi-agentes [Wooldridge and Ciancarini 2001]:

- Distribuição de dado e controle - Para muitos sistemas de software o controle global é distribuído em vários nós de computação freqüentemente distribuídos geograficamente. A fim de fazer com que estes sistemas trabalharem efetivamente, estes nós devem ser capazes de interagir autonomamente uns com os outros, ou seja, eles devem ser agentes.
- Sistemas legados - Uma maneira natural de incorporar sistemas legados em modernos sistemas de informação distribuídos é "agenticiando-os", isto é, envolvendo-os em uma camada de agente, que permitirá que eles interajam com outros agentes.
- Sistemas abertos - Muitos sistemas são abertos no sentido de que não é possível saber, em tempo de projeto, quais componentes compreenderão exatamente o sistema, nem como estes componentes interagem entre si. Para operarem efetivamente, estes sistemas devem ser capazes de tomar decisão autônoma flexível, ou seja, devem ser compostos por agentes.

Existem também razões que justificam o uso de sistemas multi-agentes ao invés de sistemas compostos por um único agente. Estas razões visam obter sistemas de software robustos e eficientes [Odell 2000]:

- Um agente poderia ser construído para fazer tudo (onipotente), mas agentes complexos representam um gargalo para velocidade, confiabilidade, manutenibilidade, etc. Dividir funcionalidade entre muitos agentes permite que a aplicação seja modular, flexível, modificável e extensível.
- Conhecimento especializado não é freqüentemente disponível em um único agente (onisciência). O conhecimento que é distribuído em várias fontes (agentes) pode ser integrado para uma maior completude quando for necessário.

Neste contexto, sistemas orientados a agentes estão sendo cada vez mais usados na indústria como, por exemplo, em aplicações voltadas para telecomunicações e comércio eletrônico [Luck et al. 2003]. Na próxima seção abordaremos as principais áreas onde os sistemas orientados a agentes estão presentes atualmente.

5.2.4. Áreas de Aplicação de Sistemas Multi-Agentes

A tecnologia de agentes está deixando de ser usada exclusivamente em universidades e laboratórios de pesquisa e está começando a ser usada também para resolver problemas do mundo-real em uma escala de aplicações industriais e comerciais. Aplicações em uso existem hoje e novos sistemas estão sendo desenvolvidos cada vez mais [Luck et al. 2003]. Nesta seção nós identificamos as principais áreas onde as abordagens baseadas em agente estão sendo usadas e provemos indicadores para alguns exemplos de sistemas nestas áreas [Wooldridge 2002].

Aplicações industriais – Aplicações industriais que usam o paradigma de agentes estão entre as primeiras que foram desenvolvidas e, portanto, a tecnologia de agentes está sendo aplicada em uma larga escala de sistemas industriais:

> Fabricação: sistemas nesta área incluem configuração de produtos de fabricação, controle de operações de fabricação, controle de robô de fabricação e determinação de seqüências de produção para um fábrica.
> Controle de processo: podemos citar sistemas para monitoração e diagnóstico de faltas em plantas de força nuclear, controle de espaço-nave, controle climático, controle de processamento de bobina de aço, gerencia de transporte elétrico e controle de acelerador de partícula.
> Telecomunicações: estes sistemas pretendem prover serviços melhores, mais rápidos e confiáveis e incluem controle de rede, transmissão e chaveamento, gerência de serviço e gerência de rede.
> Controle de Tráfico Aéreo: agentes são usados para representar tanto os equipamentos de aviação quanto os vários sistemas de controle de tráfego aéreo em operação.
> Sistemas de transporte: o domínio de gerencia de tráfego e transporte é adequado para uma abordagem baseada em agente por causa da sua natureza geograficamente distribuída.

Aplicações Comerciais – As aplicações comerciais, especialmente aquelas preocupadas com gerência da informação, tendem a serem orientadas mais para o mercado em geral.

> Gerência de informação: Mecanismos de busca na *web* são realizados por agentes, que agem autonomamente em benefício de algum usuário. Esta é provavelmente uma das áreas mais ativas para aplicações

baseadas em agentes. Outras áreas incluem um assistente pessoal que aprende sobre os interesses do usuário e com base neles compila um jornal pessoal, um agente assistente para automatizar várias tarefas em um *desktop* de computador, um assistente de navegação na *web* e um agente especialista de localização.
Comércio eletrônico: Um aumento da quantidade de comércio eletrônico está sendo empreendido por agentes, pois algumas tomadas de decisão podem ser realizadas por eles [AMEC 2006]. Como um exemplo, temos um simples "mercado eletrônico" chamado Kasbah [Chavez and Maes 1996]. Este sistema cria agentes de venda e de compra para cada item sendo vendido ou comprado, respectivamente.
Gerencia de processo de negócio: Organizações têm procurado desenvolver vários sistemas de Tecnologia da Informação para dar assistência a vários aspectos da gerência de processos de negócio. Aplicações nesta área incluem um sistema de gerência de cadeias fornecedoras, um sistema para gerência de fluxos de trabalho heterogêneos e um sistema baseado em agentes móveis para gerência de fluxo de trabalho inter-organizacional.

Aplicações de entretenimento –Agentes têm um papel óbvio nos jogos de computador, cinema interativo e aplicações de realidade virtual. Tais sistemas tendem a serem cheios de caracteres animados semi-autônomos, que podem ser naturalmente implementados como agentes.

Aplicações médicas – Estas aplicações incluem a área de monitoração de pacientes e plano de saúde. [Jennings and Wooldridge 2001].

Diante do uso crescente desta nova tecnologia alguns obstáculos e desafios foram encontrados no desenvolvimento de software orientado a agentes, os quais serão apresentados a seguir.

5.2.5. Desafios do Desenvolvimento Orientado a Agentes

Depois de destacar os benefícios potenciais da engenharia de software orientada a agentes, apresentamos nesta seção algumas das desvantagens inerentes à construção de software usando esta nova tecnologia. O conjunto de problemas exposto a seguir é diretamente atribuído às características do software orientado a agente e são intrínsecos à abordagem. Entretanto, projetistas tem encontrado meios de contornar estes problemas, já que sistemas multi-agentes robustos e confiáveis têm sido construídos.

O fato de que agentes têm de agir perseguindo seus objetivos enquanto mantém uma interação contínua com seu ambiente torna difícil projetar um software capaz de manter equilíbrio entre os comportamentos pró-ativo e reativo. Para atingir este equilíbrio é necessário que a tomada de decisão seja sensível ao contexto, resultando num grau significante de imprevisibilidade sobre quais

objetivos o agente perseguirá, em quais circunstancias e quais os métodos que serão usados para atingir os objetivos escolhidos [Jennings and Wooldridge 2001].

Já que os agentes são entidades autônomas, os padrões e os efeitos de suas interações são imprevisíveis. Tanto a natureza (uma requisição simples versus uma negociação prolongada) como a conseqüência de uma interação podem não estar determinadas desde o início do projeto de um sistema multi-agentes [Jennings and Wooldridge 2001].

Outro aspecto da imprevisibilidade no projeto de sistemas orientados a agentes refere-se à noção de comportamento emergente. Foi reconhecido que a composição interativa de agentes resulta em um fenômeno comportamental que não pode ser entendido de forma geral exclusivamente em termos do comportamento dos agentes individuais.

Além destes problemas específicos da tecnologia de agentes, também foram identificadas várias armadilhas que ocorrem repetidamente no desenvolvimento de sistemas multi-agentes [Wooldridge 2002]:

- Embora haja muitas aplicações usando agentes, eles não são uma solução universal. Há muitas aplicações para as quais os paradigmas convencionais de desenvolvimento de software (tal como a orientação a objetos) são mais apropriados.
- A ausência de técnicas testadas e confiáveis para dar assistência no desenvolvimento de software multi-agentes encoraja os projetistas a esquecerem que estão na verdade desenvolvendo software. Nesse caso, a falha do projeto não é por causa dos problemas específicos de agente, mas porque a boa prática da engenharia de software foi ignorada.
- Por natureza, sistemas multi-agentes tendem a ter múltiplas linhas de controle (tanto num agente quanto numa sociedade de agentes). Assim, na construção de software multi-agentes, é preciso lidar com problemas inerentes aos sistemas concorrentes e distribuídos, tais como sincronização, exclusão mútua para recursos compartilhados e *deadlock*, pois eles não desaparecem apenas porque foi adotada uma abordagem baseada em agentes.
- O paradigma de agentes é novo e, portanto, não pode ser considerado uma solução mágica (*Silver Bullet*). Isto seria perigosamente ingênuo. No entanto, há bons argumentos que indicam que a tecnologia de agentes levará a melhorias no desenvolvimento de software complexo e distribuído. Mas estes argumentos ainda não foram quantitativamente comprovados.
    - Um dos maiores obstáculos para o grande uso da tecnologia de agentes é que não há nenhuma plataforma que forneça toda a infra-estrutura básica requerida para criar sistemas multi-agentes (tais como manipulação de mensagem, ras-

treamento, monitoração e gerência em tempo de execução). Como resultado, quase todos os projetos de sistemas multi-agentes têm tido uma porção significante dos recursos disponíveis devotados a implementar esta estrutura.

- O comportamento dinâmico de um sistema multi-agentes é complexo e pode se tornar caótico. Se um sistema contém muitos agentes, então a dinâmica entre eles pode se tornar muito complexa de se gerenciar de forma efetiva. É necessário estruturar a sociedade de agentes de forma a reduzir a complexidade do sistema, aumentar a eficiência do sistema e modelar o problema em questão de forma mais exata.
- Ao se focar demais em aspectos de "inteligência", o *framework* de construção de um agente pode ficar muito sobrecarregado com técnicas experimentais (interfaces de linguagem natural, planejadores, provadores de teorema, sistemas de manutenção de razão, etc.). Para ser útil sugere-se construir agentes com um mínimo de técnicas Inteligência Artificial.
- É preciso evitar a tendência de se querer usar agentes para construir todo tipo de entidades computacionais, inclusive as de granulação muito fina. Isto causaria uma sobrecarga na gerencia dos agentes e na comunicação inter-agentes, o que comprometeria os benefícios de se ter uma solução baseada em agentes.

Esta seção não teve a intenção de sugerir que o desenvolvimento baseado em agentes é mais complexo e inclinado a erro quando comparado às abordagens tradicionais de engenharia de software. Ao invés disso, nós reconhecemos que há certas armadilhas que são comuns às soluções baseadas em agentes – assim como há certas armadilhas comuns às soluções baseadas em objetos. Reconhecendo estas armadilhas, não podemos garantir o sucesso de um projeto de software baseado em agentes, mas podemos eliminar algumas das fontes de falha mais óbvias no seu desenvolvimento.

Na próxima seção estaremos abordando a necessidade de se ter um meio sistemático de analisar o problema, estruturá-lo como um sistema orientado a agentes, determinar quais agentes individuais devem ser desenvolvidos e, então, integrá-los para compor o sistema.

5.3. Desenvolvimento de Software Orientado a Agentes

Esta seção apresenta a necessidade de metodologias de desenvolvimento orientado a agentes e apresenta as principais metodologias propostas para a engenharia de software orientada a agentes.

5.3.1. Motivação

A engenharia de software orientada a agentes apresenta um novo nível de abstração na construção de sistemas de software. Esse permite que o engenheiro de software projete um sistema em termos de agentes que interagem entre si. Porém, hoje em dia, os projetistas de software ainda não podem explorar todos os benefícios oferecidos pelo paradigma de agentes devido à ausência de notações diagramáticas, metodologias sistemáticas e ferramentas reconhecidas que suportem todas as fases do ciclo de desenvolvimento de software [Silva et al. 2004].

5.3.2. Metodologias de desenvolvimento de sistemas multi-agentes

Para gerenciar com sucesso a complexidade associada ao desenvolvimento, manutenção e distribuição de sistemas multi-agentes, um conjunto de técnicas e ferramentas de engenharia de software é solicitado ao longo do ciclo de vida do software. Vantagens óbvias serão alcançadas se uma linguagem de agente puder ser facilmente integrada aos processos e ferramentas de engenharia de software existentes.

A construção de sistemas multi-agentes não é fácil, pois se têm todos os problemas dos sistemas distribuídos e concorrentes tradicionais, mais as dificuldades adicionais que surgem dos requisitos de flexibilidade e interações sofisticadas. Qualquer metodologia para a engenharia de software orientada a agentes deve prover abstrações adequadas e ferramentas para modelar não só as tarefas individuais, mas também as tarefas sociais dos agentes. Nesse sentido, várias metodologias para sistemas muti-agentes têm sido propostas nos últimos anos [Zambonelli et al. 2000]. Muitas delas usam técnicas de modelagem e metodologias orientadas a objetos por base. Exemplos dessas metodologias são Gaia [Zambonelli et al. 2003], AUML [Bauer et al. 2000, Odell et al. 2001] e MaSE [DeLoach 2006]. Uma nova abordagem, o Tropos [Castro et al. 2002, Bresciani et al. 2004, Giorgini et al. 2005], baseia-se nos conceitos usados durante a análise de requisitos iniciais e tenta modelar e implementar sistemas multi-agentes visando reduzir, tanto quanto possível, a incompatibilidade entre o sistema e seu ambiente.

Nesta seção são abordadas as principais metodologias orientadas a agentes.

5.3.2.1. Gaia

Essa metodologia [Zambonelli et al. 2003] é aplicável a grande conjunto de sistemas multi-agentes, lidando com características de nível macro (sociedade) e de nível micro (agente) dos sistemas. Baseia-se na visão de um sistema multi-agentes comportando-se como uma organização computacional com vários papéis que interagem para realizar os objetivos do sistema. Permite ir-se do estabelecimento de requisitos até ao projeto suficientemente detalhado, a ponto de ser implementado diretamente. Toma emprestada parte da terminologia e notação da análise e projeto orientados a objetos.

Os principais conceitos dessa metodologia dividem-se em abstratos e concretos. Entidades abstratas são aquelas usadas durante a análise para conceituar o sistema, mas que não têm necessariamente qualquer realização direta no sistema. Estas entidades incluem Papéis, Permissões, Responsabilidades, Protocolos, Atividades, Propriedades Vitais e Propriedades de Segurança. As entidades concretas, por outro lado, são usadas no processo de projeto e, tipicamente, terão realização direta no sistema executável. Estas entidades incluem Tipos de Agente, Serviços e Conhecimentos.

A fase análise (Figura 5.1) cobre as entidades abstratas e visa desenvolver um entendimento do sistema e de sua estrutura (sem referência a nenhum detalhe de implementação). Assim, esse entendimento é capturado na organização do sistema. Uma organização é vista como uma coleção de papéis, que se relacionam entre si, e que participam de padrões sistemáticos e institucionalizados de interação com outros papéis. Como produto dessa fase, têm-se dois modelos: modelo de papéis, que identifica os papéis chave no sistema; e modelo de interações, que identifica as dependências e os relacionamentos entre os vários papéis numa organização multi-agentes.

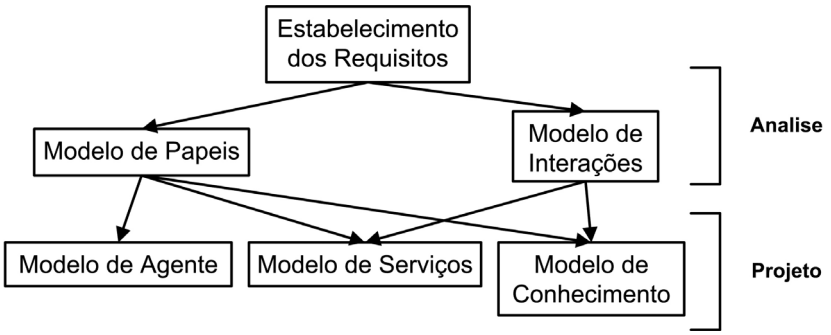

Figura 5.1. Relacionamentos entre os modelos Gaia

Um papel é definido por quatro atributos: responsabilidades, permissões, atividades e protocolos. As responsabilidades determinam funcionalidades e, conseqüentemente, são os atributos chaves associados a um papel, dividindo-se em dois tipos: propriedades vitais (*liveness*) e propriedades de segurança (*safety*). As primeiras, intuitivamente estabelecem que "alguma coisa boa aconteça" e descrevem as situações que um agente pode provocar em determinadas condições ambientais. Já as segundas propriedades são invariáveis e, intuitivamente, estabelece que "nada de ruim aconteça" (isto é, que uma situação aceitável seja mantida durante todos os estados da execução). As permissões são "direitos" associados a um papel, que identificam os recursos que estão disponíveis para aquele papel, de modo que ele possa realizar suas responsabilidades. Permissões tendem a ser recursos de informação. As atividades de um papel são computações associadas a ele, podendo ser

realizadas por um agente sem que ele precise interagir com outros agentes. Finalmente, um papel também é identificado por vários protocolos que definem a maneira com que ele interage com outros papéis.

Ao nível de projeto, (Figura 5.1), Gaia compreende a geração de três modelos: o de agente; o de serviços; e o de conhecimento. O modelo de agente identifica os tipos de agente que vão formar o sistema e as instâncias de agente geradas a partir destes tipos. O modelo de serviços identifica os principais serviços que são requisitados para realizar o papel do agente. Por fim, o modelo de conhecimento documenta as linhas de comunicação entre os diferentes agentes.

*5.3.2.2. AUML (**Agent Unified Modeling Language**)*

Para tornar as notações orientadas a objetos mais padronizadas na especificação, visualização, documentação e construção de artefatos de um sistema, foi definida pelo Grupo de Gerenciamento de Objetos (*Object Management Group* - OMG), uma linguagem de modelagem de objetos denominada de Linguagem de Modelagem Unificada (UML) [Booch et al. 1999, OMG 2005]. Todavia, hoje, a visão de agentes, conduz à exploração de extensões para a UML, a fim de que se possam acomodar as diferentes caracterísiticas específicas de um agente [Bauer 2001]. Com esse propósito, estabeleceu-se uma cooperação entre a Fundação de Agentes Físicos Inteligentes (*Foundation of Intelligent Physical Agents* - FIPA) [FIPA 2005] e a OMG. Como primeiro resultado desta cooperação, foi proposta a abordagem Agent UML ou AUML [Bauer et al. 2000, Odell et al. 2001].

A AUML é uma metodologia que estende a Linguagem de Modelagem Unificada (UML) para suportar o desenvolvimento de sistemas orientados a agentes. A AUML propõe uma representação em três camadas para os protocolos de interação de agentes (Figura 5.2). Essas camadas descrevem um padrão de comunicação entre agentes, definindo a seqüência permitida de mensagens e as restrições sobre o conteúdo destas mensagens. Os protocolos de agentes, representados pela abordagem em camadas, definem agora: modelos e pacotes; diagramas de seqüência e colaboração; e máquinas de estados e diagramas de atividades.

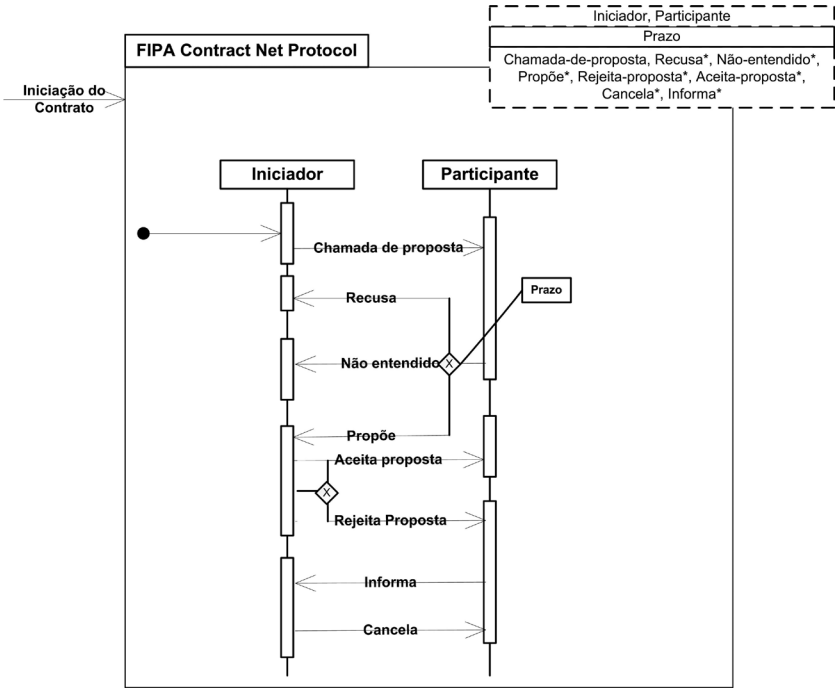

Figura 5.2. Protocolo de interação de agente, genérico, expresso como um pacote modelo

Nos modelos e pacotes o protocolo como um todo é tratado como uma entidade independente. Um pacote é uma agregação conceitual de seqüências de interação. O protocolo empacotado pode ser tratado como um padrão que pode ser personalizado para domínios de problema semelhantes. O quadrado pontilhado na parte superior direita da Figura 5.2 expressa este padrão como a especificação de um modelo que identifica entidades que não se limitam ao pacote, mas que precisam ser limitadas quando o modelo do pacote estiver sendo instanciado.

Os diagramas de seqüência e colaboração fornecem uma visão alternativa das interações entre os agentes (captura a dinâmica inter-agente). Expressar os papéis que um agente pode exercer no curso de uma interação com outros agentes é essencial. Assim, embora os diagramas de colaboração atualmente não permitam representar os papéis de um agente durante uma interação, cada ato de comunicação pode ser rotulado com o papel responsável por sua requisição. Já os diagramas de atividades e máquinas de, estados detalham o comportamento interno do agente quando este estiver executando os protocolos. Os diagramas de atividades também podem ser modificados

para representar papéis de um agente, associando cada atividade com o nome do papel classificador apropriado. Além disso, a mudança de papéis pode ser representada em diagramas de atividades através de notas.

Uma importante propriedade de um agente é a mobilidade e uma forma de representá-la é usando extensões dos diagramas de distribuição para indicar caminhos de mobilidade. Sabendo que os agentes tomam emprestadas muitas analogias da natureza, encontramos também uma proposta de extensão dos diagramas de classes, seqüência e atividades para representar clonagem, mitose e reprodução de agentes, além de representar relacionamentos parasitas e simbióticos entre eles. Destaca-se que a interação de muitos agentes pode provocar um fenômeno chamando de emergência. Este fenômeno pode ser representado usando uma extensão do diagrama de classes.

A AUML fornece modelos para capturar o aspecto dinâmico das interações inter-agente. mas ainda não fornece extensões para capturar o mapa cognitivo do agente (individual/estrutura estática) ou a organização estrutural de agentes (sistema/estrutura estática). Dessa forma, podemos verificar a cobertura da AUML relativa aos seguintes aspectos:

- Agente Individual/ Estrutura Estática: a AUML não fornece extensões da UML que permitam representar o mapa cognitivo de um agente.
- Agente Individual/ Dinâmico: o terceiro nível do protocolo de interação de agentes captura o processamento interno do agente e fornece um mecanismo pra representar a dinâmica do mapa cognitivo do agente. A AUML não configura uma arquitetura cognitiva específica e representa a dinâmica interna do agente usando diagramas de atividades e de estados.
- Sistema Social/ Estrutura Estática: a AUML configura a hierarquia de classes incluídas no modelo UML e adiciona a noção do agente. Nenhum outro mecanismo é fornecido para capturar a estrutura do sistema.
- Sistema Social/ Dinâmico: os dois primeiros níveis do protocolo de interação de um agente relatam a dinâmica do sistema e podem ser usados tanto para o modelo conceitual quanto para o modelo de projeto. No primeiro nível, modelos e pacotes fornecem soluções reusáveis para o sequenciamento de mensagens.

Por fim, tem-se a metodologia AUML cobrindo apenas a fase de projeto do ciclo de vida de software e suportando apenas a especificação da interação entre os agentes. Contudo, outras abordagens têm sido propostas para suportar características estruturais e dinâmicas de sistemas multi-agentes, tais como as abordagens apresentadas em [Bauer and Odell 2002], [Silva and Lucena 2004], [Castro et al. 2003], [Silva, Noya and Lucena 2005] and [Silva et al. 2006].

5.3.2.3. MaSE (Multiagent Systems Engineering)

MaSE [DeLoach 2006] foi originalmente projetada para desenvolver

sistemas multi-agentes de propósito geral e tem sido usada para projetar sistemas robóticos cooperativos. Ela suporta duas fases do ciclo de vida de software: Análise e Projeto.

A meta da fase de análise de MaSE é definir um conjunto de papéis que podem ser usados para atingir os objetivos ao nível de sistema. Este processo é capturado em três passos: Capturar Objetivos, Aplicar Casos de Uso e Refinar Papéis.

- Capturar Objetivos: O primeito passo é capturar os objetivos do sistema extraindo-os dos requisitos, o que é feito por Identificar Objetivos e Estruturar Objetivos. O propósito de Identificar Objetivos é derivar o objetivo geral do sistema e seus sub-objetivos. Isto é feito extraindo cenários dos requisitos e então identificando objetivos dos cenários. Depois que os objetivos foram identificados, o segundo passo, Estruturar Objetivos, categoriza e estrutura os objetivos em uma árvore de objetivos, que resulta em um Diagrama de Hierarquia de Objetivos, representando objetivos e relacionamentos entre objetivos e sub-objetivos.
- Aplicar Casos de Uso: Neste passo, objetivos são traduzidos em casos de uso, que capturam os cenários previamente identificados com uma descrição detalhada e um conjunto de diagramas de sequencia. Estes casos de uso representam os comportamentos desejados do sistema e a sequência de eventos.
- Refinar Papéis: Este passo organiza papéis em um Modelo de Papel, que descreve os papéis no sistema e a comunicação entre eles. Cada papel é decomposto em um conjunto de tarefas, que são projetadas para atingir os objetivos de responsabilidade de cada papel. Estas tarefas são documentadas usando um Diagrama de Tarefas Concorrentes. Tarefas Concorrentes consistem de um conjunto de estados e transições que representam comunicação e raciocínio interno do agente.

O propósito da fase de projeto é converter papéis e tarefas em uma forma mais sugestiva para implementação, especialmente os agentes e os diálogos. Esta fase consiste em quatro passos: Construção de Classes de Agentes, Construção de Diálogos, Agrupar Classes de Agentes e Projeto da Distribuição.

- Construção de Classes de Agentes: O primeito passo da fase de projeto identifica as classes de agentes e seus diálogos e, então, documenta-os em Diagramas de Classe de Agentes. O Diagrama de Classes de Agentes que resulta deste passo é semelhante ao diagrama de classes orientado a objetos, porém com duas diferenças: (i) classes de agentes são definidas por papéis ao invés de atributos e métodos e (ii) relacionamentos entre as classes de agentes são diálogos.

- Construção de Diálogos: Uma vez que as classes de agentes e os seus diálogos estão definidos, o projeto de diálogo detalhado é realizado. Diálogos modelam comunicação entre duas classes de agentes usando um par de automato de estado finito semelhante, na forma e na função, à tarefas concorrentes. Cada tarefa gera múltiplos diálogos, quando solicitam comunicação com mais de uma classe de agente.
- Agrupar Classes de Agentes: Este passo envolve a definição da arquitetura interna de cada agente. MaSE não assume nenhuma arquitetura de agente em particular e permite que sejam usadas uma grande variedade de arquiteturas novas e existentes. A arquitetura do sistema é definida usando componentes semelhantes àqueles definidos na UML.
- Projeto da Distribuição: O passo final de projeto é escolher a configuração real do sistema, que consiste em vários agentes (que podem ter arquiteturas diferentes) e as plataformas onde eles deverão estar distribuídos. Estas decisões são documentadas no Diagrama de Distriuição, que é semelhante ao Diagrama de Distribuição da UML.

5.3.2.4. Tropos

Freqüentemente, sistemas de software falham em suportar interesses das organizações das quais eles são parte integrante. Isto acontece devido à ausência de um entendimento apropriado da organização pelos projetistas do sistema de software, bem como a freqüência das mudanças organizacionais que não podem ser acomodadas pelos sistemas de software existentes. Neste contexto, a engenharia de requisitos vem sendo reconhecida como a fase mais crítica no desenvolvimento de sistemas, porque as considerações técnicas têm de ser balanceadas contra as sociais e organizacionais. O projeto Tropos [Castro et al. 2002, Bresciani et al. 2004, Giorgini et al. 2005] é uma proposta de desenvolvimento de sistemas orientados a agentes, inspirada na análise de requisitos e fundamentada em conceitos sociais e intencionais [Yu 1995], que visa reduzir tanto quanto possível esta incompatibilidade entre o sistema e seu ambiente. Esta abordagem será apresentada em detalhe na próxima seção.

5.4. O Framework Tropos

Nesses últimos anos, os sistemas orientados a agentes têm mostrado um potencial para suportar maior diversidade, oferecer maior flexibilidade e melhor robustez, assim como funcionalidades mais sofisticadas comparadas às tecnologias de software tradicionais. Entretanto, esses benefícios não podem ser realizados a menos que novas metodologias de desenvolvimento, além da tradicional análise estruturada e da orientação a objetos, sejam criadas para ajudar a entender estruturas sociais complexas no domínio do problema da aplicação e traduzi-las em soluções tecnológicas. Neste contexto, passamos a trabalhar no contexto do projeto Tropos [Castro et al. 2002].

5.4.1. Origem e Motivação

O *framework* Tropos (derivado do grego "*tropé*", que significa "facilmente modificável" ou "facilmente adaptável") [Castro et al. 2002, Bresciani et al. 2004, Giorgini et al. 2005] baseia-se nos conceitos utilizados para modelar requisitos iniciais e complementa propostas para o desenvolvimento orientado a agentes. O processo inicia com um modelo do ambiente no qual o sistema que está sendo desenvolvido irá operar. O modelo é descrito em termos de atores, seus objetivos e interdependências. Através de refinamentos incrementais, este modelo é estendido para incluir tanto o sistema a ser desenvolvido quanto aos seus subsistemas, que também são representados como atores a quem foram delegados objetivos para atingir. Também há a definição de planos a serem executados e recursos a serem fornecidos pelos atores.

Tropos é fundamentado nos conceitos oferecidos pelo *framework* i* (i-estrela, que simboliza "intencionalmente distribuído"). O i* é um *framework* de modelagem que inclui os conceitos de ator (atores podem ser agentes, posições ou papéis) e suas interdependências intencionais, incluindo dependências de objetivo, objetivo-*soft*, tarefa e recurso [Yu 1995]. Este *framework* inclui o modelo de dependência estratégica (SD – *Strategic Dependency*) e o modelo de razão estratégica (SR – *Strategic Rationale*). Ambos os modelos são usados para capturar as intenções dos *stakeholders* (usuários, proprietários, gerentes, etc.), as responsabilidades do sistema pretendido em relação a estes *stakeholders*, a arquitetura do sistema pretendido e os detalhes do seu projeto. Estes modelos podem ser usados como parte da documentação de um sistema de software durante todas as fases do desenvolvimento de software cobertas pelo Tropos [Castro et al. 2002]. As fases cobertas pela abordagem são as seguintes: Requisitos Iniciais, Requisitos Finais, Projeto Arquitetural, Projeto Detalhado e Implementação.

5.4.2. Técnicas de Modelagem de Software no Contexto do Tropos

Essa seção descreve as principais técnicas de modelagem usadas nas fases de desenvolvimento cobertas pelo *framework* Tropos.

*5.4.2.1. Framework i**

É sabido que Tropos propõe uma abordagem centrada em requisitos para o desenvolvimento orientado a agentes. Utiliza, sobretudo, a abordagem i* para a captura de requisitos em suas fases de Requisitos Iniciais e Finais.

O *framework* i* [Yu 1995] pretende facilitar o raciocínio sobre o sistema a ser desenvolvido e/ou sob a análise, dando ênfase aos aspectos sociais, representados por atores. Para isso, o *framework* i* fornece uma visão gráfica desses atores e suas intenções, dependências, responsabilidades e vulnerabilidades. Esta abordagem é utilizada para: (i) obter uma melhor compreensão sobre os relacionamentos da organização; (ii) entender as razões envolvidas nos processos de decisões; e (iii) descrever potenciais alternativas do sistema pretendido [Yu 2002]. Assim, com a abordagem i*, as organizações são vistas

como se consistissem de unidades semi-autônomas chamadas de atores, cujo comportamento não é totalmente controlável ou previsível, mas regulamentado por relacionamentos sociais. Os atores têm liberdade de ação, dentro do contexto de restrições sociais, e possuem propriedades intencionais, tais como, objetivos, crenças, habilidades e compromissos. Esses atores dependem uns dos outros para terem seus objetivos (*goals*) e objetivos-*soft* (*softgoals*) alcançados, suas tarefas (*tasks*) realizadas e recursos (*resources*) fornecidos. Nesse contexto, tem-se uma abordagem que procura a compreensão mais aprofundada do processo através de uma visão intencional e estratégica. Assim, podemos afirmar que um ator *é* intencional quando possui motivações, intenções e razões por trás de suas ações. Um ator é estratégico quando não foca apenas seu objetivo imediato, mas quando se preocupa com as implicações de seu relacionamento estrutural com outros atores, ou seja, as oportunidades e vulnerabilidades presentes, devido a falhas que possam existir na relação social. Atores complexos podem ser diferenciados em três tipos: papéis, agentes e posições. A Figura 5.3 ilustra a representação gráfica e as possíveis associações entre agentes, papéis e posições.

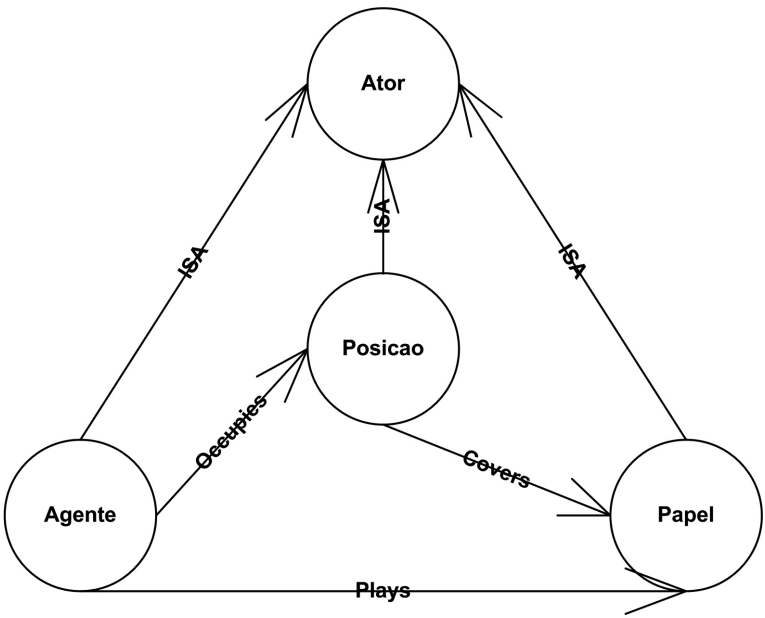

Figura 5.3. Associações e representação de agentes, papéis e posições

Dizemos que um agente executa um determinado papel e ocupa uma determinada posição, que cobre um papel. Um agente é um ator que possui

manifestações físicas concretas e é usado para representar uma entidade real tal como uma pessoa ou um sistema. Já um papel representa a caracterização abstrata do comportamento de um ator social dentro de determinados contextos sociais ou domínio de informação. Apresenta as funções que podem ser exercidas por um agente dentro da organização, ou seja, um conjunto de responsabilidades e intenções que um agente possui. Uma *posição* representa uma entidade intermediária entre um agente e um papel. É o conjunto de papéis tipicamente ocupados por um agente.

A abordagem i* é composta por dois modelos: o Modelo de Dependência Estratégica (SD) (Figura 5.4) e o Modelo de Razão Estratégica (SR) (Figura 5.6). Para auxiliar na apresentação desses modelos, consideremos um pequeno estudo de caso baseado no domínio de uma Redação de Jornal Eletrônico. Assim, quando um usuário deseja ler uma notícia, acessa o *website* de um jornal que é mantido por um *Webmaster*, que será o responsável pela atualização das informações publicadas. O jornal a ser publicado é editado e fornecido ao *Webmaster* pelo Editor Chefe. O Editor Chefe compartilha as pautas do jornal entre os Editores, responsáveis pela edição das notícias de uma categoria específica. Cada Editor contata um ou mais Repórteres que são capazes de cobrir e fornecer as notícias sobre um assunto específico. Cada Repórter, por sua vez, contata um Fotógrafo para realizar a cobertura fotográfica da notícia.

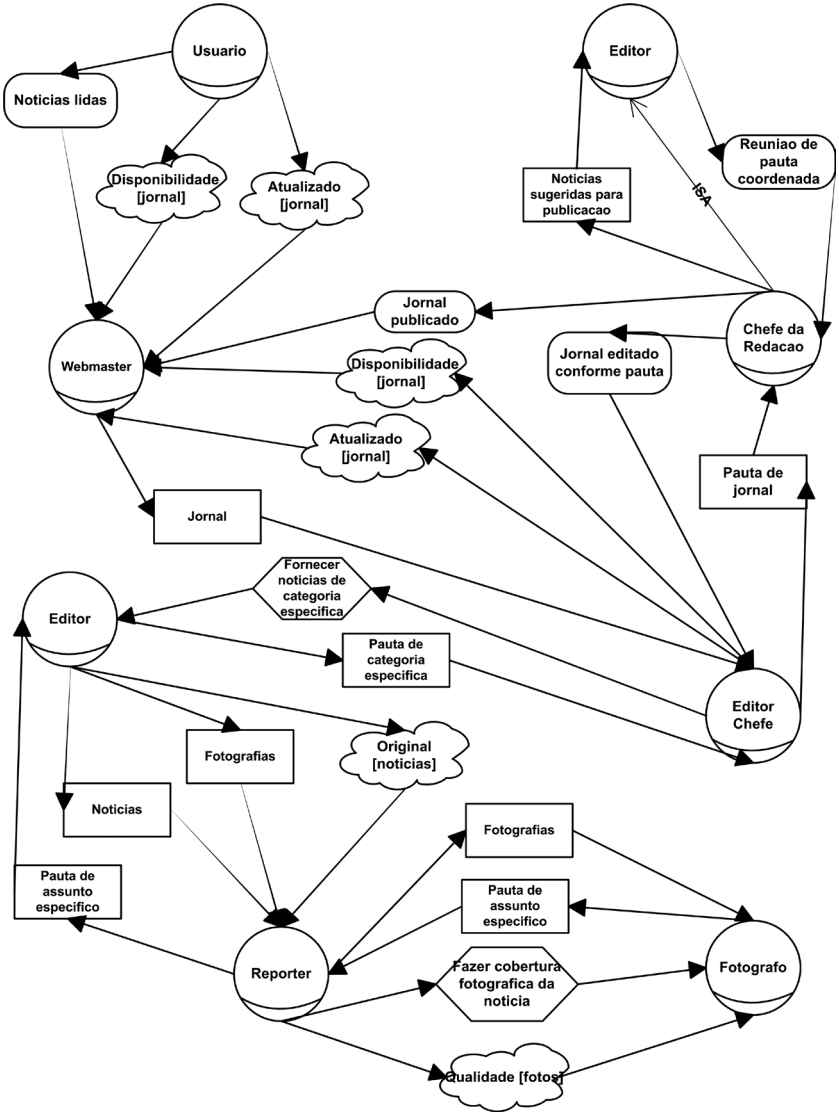

Figura 5.4. Modelo SD para a Redação do Jornal Eletrônico

O modelo SD (Figura 5.4) descreve as relações de dependências externas entre os atores da organização e consiste em um conjunto de nós e ligações entre esses, onde os nós representam os atores e cada ligação in-

dica uma dependência entre dois atores. Neste modelo, distinguem-se quatro tipos de dependências: (i) Recursos, que estão relacionados com a noção de entidades, e.g. fotografias, pauta do jornal, notícias e jornal; (ii) Tarefas, que dizem respeito às atividades, e.g. fazer cobertura fotográfica das notícias; (iii) Objetivos, que são afirmações, e.g. jornal publicado, reunião de pauta coordenada e notícias do dia lidas; e (iv) Objetivos-*Soft*, que estão relacionados com a noção de requisitos não funcionais, e.g. qualidade [fotografias], originalidade [notícias], atualizado [jornal], disponibilidade [jornal].

De uma forma geral, esses tipos de dependências caracterizam como as decisões recaem sobre um dos lados da dependência, indicando quem resolverá os problemas que surgirem. Os atores envolvidos numa relação de dependência (Figura 5.5) foram denominados como: (i) "*depender*" ou ator dependente, se depende que outro ator satisfaça a dependência; e (ii) "*dependee*" ou ator de quem se depende, será o responsável por satisfazer a dependência.

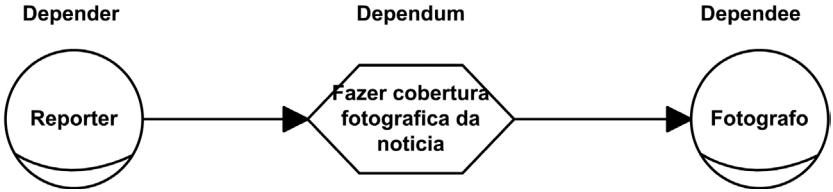

Figura 5.5. Definição de uma relação de dependência entre atores

No Modelo SD podem ser identificados diferentes graus de dependências: (i) aberta (*open*), graficamente expressa por "_". É usada quando em caso de falha da relação de dependência, as intenções são afetadas, mas sem conseqüências sérias; (ii) compromissada (*committed*), graficamente não possui sinal associado. É usada quando em caso de falha da relação de dependência, as intenções são afetadas significativamente, levando-se a questionar a viabilidade da dependência; e (iii) crítica (*critical*), graficamente explicitada por meio do sinal "X". É usada quando em caso de falha da relação de dependência, as intenções são afetadas seriamente, levando-se a uma preocupação quanto a viabilidade de toda a rede de relacionamentos e dependências [Yu 1995]. Nas Figuras 5.4 e 5.6 todas as dependências são do tipo compromissada (*committed*).

O Modelo de Razão Estratégica (SR), Figura 5.6, é usado para: (i) descrever os interesses, preocupações e motivações dos participantes de um processo; (ii) possibilitar avaliação das possíveis alternativas de definição do processo; e, (iii) investigar, mais detalhadamente, as razões existentes por trás das dependências entre os vários atores. No entanto, possíveis restrições temporais não aparecem graficamente neste modelo.

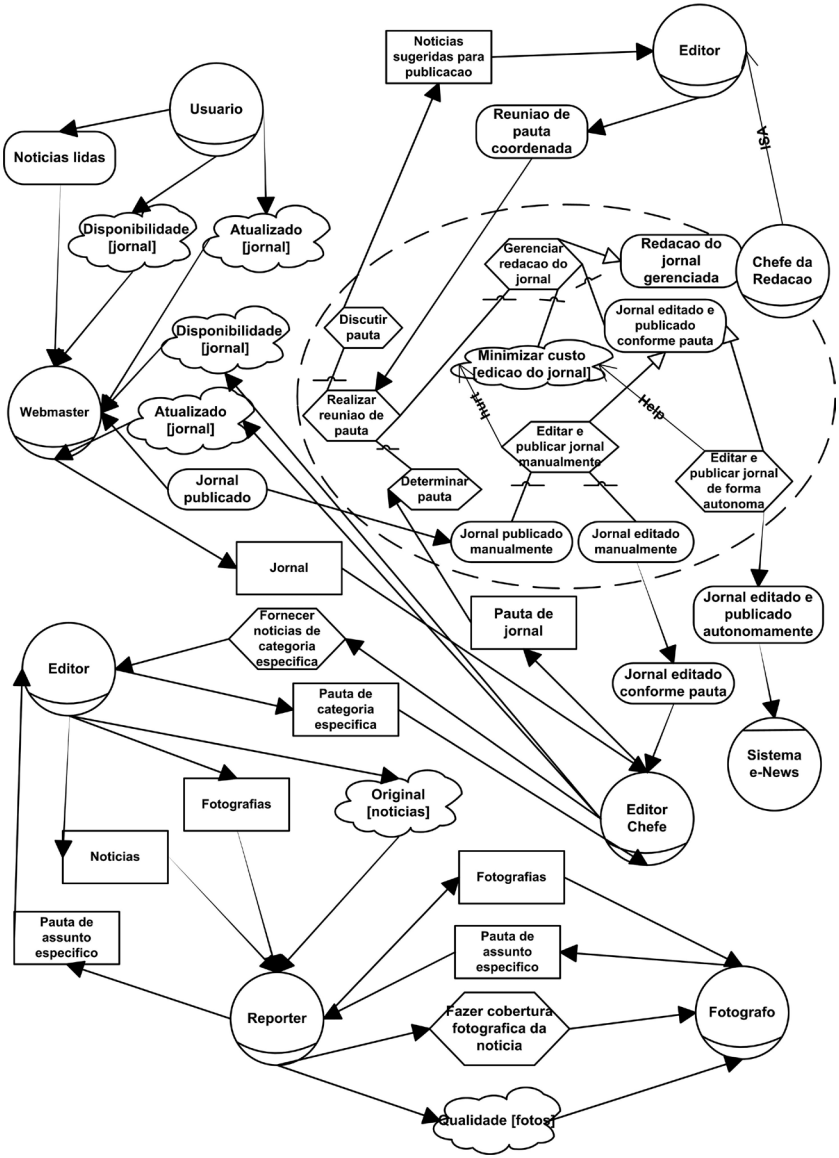

Figura 5.6. Modelo SR para a Redação do Jornal Eletrônico

O modelo SR permite uma compreensão mais aprofundada sobre as razões do processo e é composto pelos elementos objetivo, objetivo-soft, tarefa e recurso, e por relacionamentos que, juntos, fornecem uma estrutura para expressar as razões envolvidas no processo. Estes relacionamentos incluem: (i) meios-fins (*means-ends*); e (ii) decomposição de tarefas (*task-decomposition*). Os relacionamentos meios-fins indicam um relacionamento entre um fim – que pode ser: (i) um objetivo a ser alcançado; (ii) uma tarefa a ser realizada; (iii) um recurso a ser produzido; (iv) ou um objetivo-*soft* a ser satisfeito – e um meio, para se atender a esse fim, que pode ser uma tarefa a ser realizada. Este relacionamento sugere que podem existir outros meios de alcançar o mesmo fim, exprimindo as alternativas existentes. Por exemplo, na Figura 5.6, o objetivo-*soft* minimizar custos [edição do jornal] pode ser alcançado pelas duas tarefas alternativas: editar e publicar jornal de forma manual e editar e publicar jornal de forma autônoma.

Os relacionamentos de decomposição de tarefa exprimem o que deve ser feito para que a tarefa seja realizada. Em particular, uma tarefa pode ser sub-dividida em: (a) objetivo, (b) tarefa; (c) recurso; e (d) objetivo-*soft*. Esses tipos de relacionamento podem interligar-se aos relacionamentos de dependência do modelo SD, quando as razões ultrapassam os limites do ator. Assim, quando existe um relacionamento de decomposição de tarefa, uma sub-tarefa restringe a tarefa original a um curso particular de uma ação. No caso de um objetivo-*soft* ser um componente de uma tarefa, serve como um objetivo de qualidade para esta tarefa, guiando ou restringindo a seleção entre alternativas em adicionais decomposições desta tarefa. Por exemplo, no modelo da Figura 5.6, temos que no ator Chefe da Redação, a sub-tarefa realizar reunião de pauta*,* o objetivo-*soft* minimizar custos [edição do jornal], e o objetivo jornal editado e publicado conforme pauta*,* restringem a tarefa gerenciar redação do jornal, guiando-a para o fato de que não pode ser feita aleatoriamente.

No Modelo de Razão Estratégica, as ligações que envolvem objetivos-*soft* requerem um atributo extra para indicar o tipo de contribuição relacionada com esses objetivos: (i) positiva (+), se o objetivo-*soft* influenciará positivamente. É representada por uma ligação do tipo *Make*; (ii) negativa (-),se o objetivo-*soft* influenciará negativamente. É representada por uma ligação do tipo *Break;* (iii) bastante (sup), se o objetivo-*soft* influenciará de forma suficiente. É representada por uma ligação do tipo *Help;* (iv) não bastante (sub), se o objetivo-*soft* influenciará de forma não suficiente. É representada por uma ligação do tipo *Hurt;* ou (v) desconhecida ou indefinida (?), quando não se sabe qual será a influência do objetivo-*soft* em questão. Neste caso, a influencia é representada por ligações do tipo *Some*+/*Some*-. Na Figura 5.6 temos ambas as tarefas editar e publicar jornal de forma manual e editar e publicar jornal de forma autônoma, contribuindo na satisfação do objetivo-*soft*, minimizar custos [edição do jornal];

Da análise do ambiente organizacional da redação de um jornal eletrônico, identificou-se a necessidade de um sistema de software autônomo para dar suporte ao processo de edição e publicação das notícias no *website* do

jornal. Assim, o sistema *e-News* aparecerá como um ator que contribuirá na satisfação dos objetivos dos participantes da organização (Figura 5.6).

Tendo em vista a escalabilidade das aplicações, começa-se a verificar que os modelos suportados pela abordagem i* tornam-se difíceis de compreender e/ou usar. Isto porque estão se tornando maiores, mais complexos e com requisitos que se repetem e se espalham ao longo dos modelos estratégicos.

5.4.3. Fases do Processo de Desenvolvimento

Com visto na seção 5.4.2.1, o Tropos adota os conceitos e modelos oferecidos pelo framework i* [Yu 1995] durante as fases iniciais do ciclo de desenvolvimento de software. Nesta seção apresentaremos em detalhe e com exemplos as fases que são cobertas pelo Tropos.

5.4.3.1. Requisitos Iniciais

A análise de requisitos representa a etapa inicial do desenvolvimento de software em muitas metodologias. No *framework* Tropos, a modelagem e análise dos requisitos são feitas utilizando-se a abordagem i* e está dividida em duas fases fundamentais: os Requisitos Iniciais e os Requisitos Finais.

Na fase de Requisitos Iniciais, interessa-nos o entendimento do problema através da compreensão do contexto organizacional no qual o sistema a ser desenvolvido será implantado. Durante esta fase, os engenheiros de requisitos modelam os *stakeholders* como atores sociais que dependem uns dos outros e que têm intenções de atingir objetivos, realizar tarefas e fornecer recursos.

> *A análise realizada nesta fase é orientada a objetivos. Cada objetivo é analisado do ponto de vista de seu ator, resultando em um conjunto de dependências entre pares de atores. Através destas dependências, nós podemos realizar questionamentos do tipo "por que", com relação à maneira com a qual os processos estão estruturados. Questionamentos deste tipo ligam as funcionalidades do sistema às necessidades, preferências e objetivos dos stakeholders. Os resultados desta fase são dois modelos: o modelo SD (Figura 5.4) e o modelo SR (Figura 5.6).*

5.4.3.2. Requisitos Finais

A fase de Requisitos Finais está preocupada com o refinamento dos modelos SD e SR produzidos na primeira fase do Tropos [Castro et al. 2001]. Durante a análise dos requisitos finais, os modelos conceituais desenvolvidos na fase anterior são revisados e estendidos para incluir novos atores, que representam tanto o sistema a ser desenvolvido quanto sistemas e atores externos à organização. Na Figura 5.7 pode ser identificado o sistema pretendido *e-News* e as dependências descritas neste modelo representam os requisitos funcionais e não-funcionais deste sistema [Giorgini et al. 2005]. O ator representando o Sistema *e-News* relaciona-se com os atores sociais através das dependências. Nesta fase faz-se uma análise meios-fins do ator que representa o sistema para produzir um novo modelo de razão estratégica.

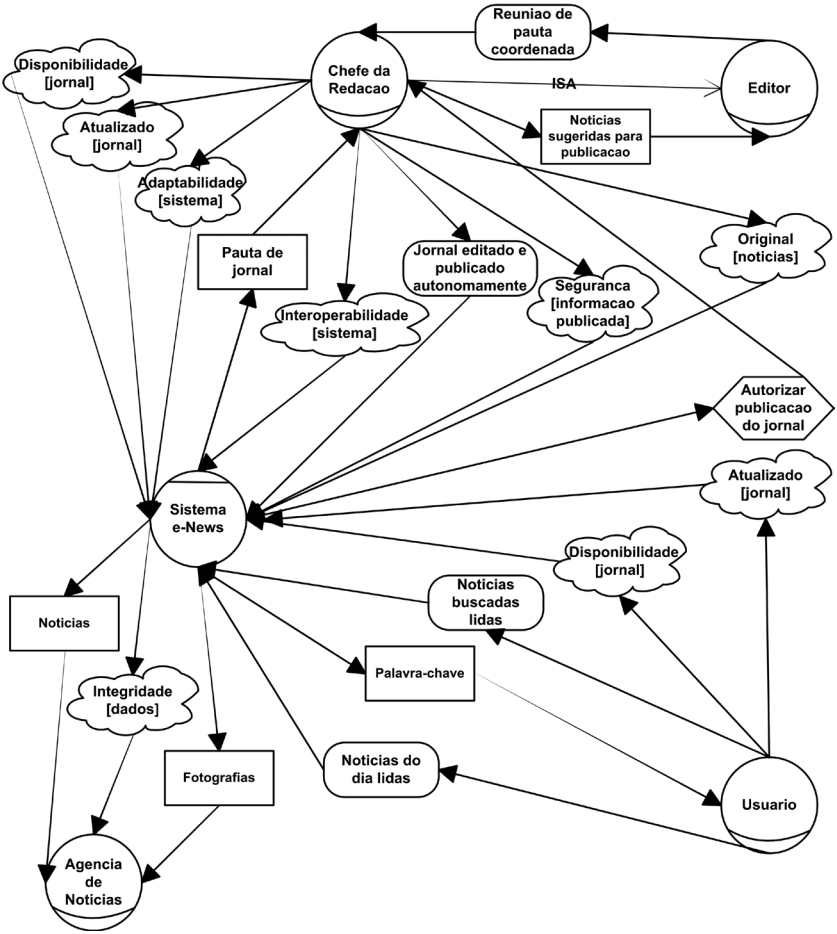

Figura 5.7. Modelo SD para o *Sistema e-News*

Para a construção do modelo SR (Figura 5.8), nesta fase, o ator que representa o sistema deve ser expandido para apresentar as razões que estão por trás de suas dependências. Suas tarefas e objetivos precisam ser revisados, analisados e detalhados através de ligações de decomposição e de meios-fins. O objetivo desta etapa é dar suporte ao processo de sugestão, exploração e avaliação de soluções alternativas para o sistema.

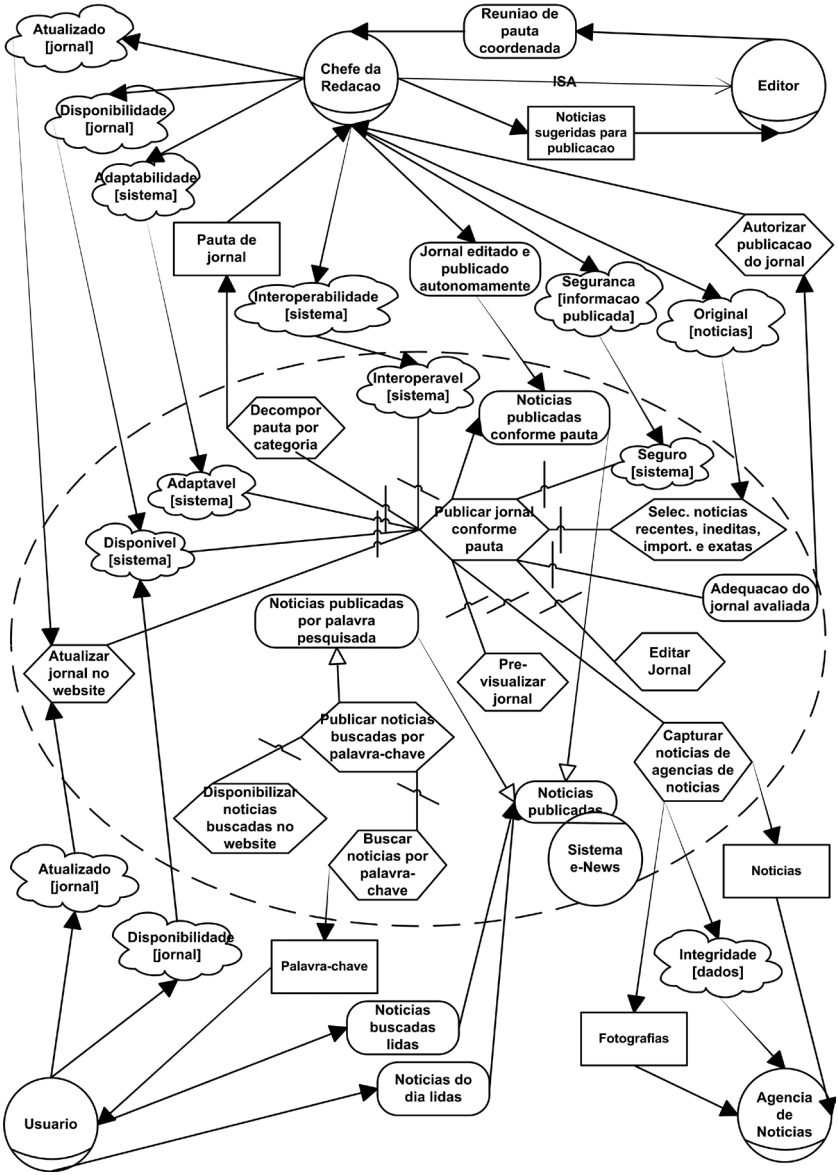

Figura 5.8. Modelo SR para o *Sistema e-News*

5.4.3.3. Projeto Arquitetural

A arquitetura de um sistema constitui um modelo da estrutura do sistema em termos de sub-sistemas, interconectados através de fluxos de controle de dados. Em Tropos, subsistemas são representados como atores e interconexões de dado/controle são representados como dependências entre os atores [Castro et al. 2003, Bastos and Castro 2005].

Um estilo arquitetural define uma família de sistemas relacionados em termos de um padrão de organização estrutural [Shaw and Garlan 1993]. Em [Kolp et al. 2002] foram definidos estilos arquiteturais organizacionais para auxiliar o projeto da arquitetura de aplicações cooperativas, dinâmicas e distribuídas, tais como sistemas multi-agentes. O estilo arquiteturais organizacionais *(pyramid, joint venture, structure in 5, takeover, arm's length, vertical integration, co-optation, bidding,* …) são estruturas genéricas definidas em um nível *meta* que podem ser instanciadas para o projeto da arquitetura de uma aplicação específica. Atualmente, Tropos utiliza a notação do *framework* i* [Yu 1995] para representar os modelos arquiteturais [Kolp et al. 2002].

O primeiro passo durante o projeto arquitetural é selecionar o estilo arquitetural mais apropriado usando como critério as qualidades desejadas que foram identificadas anteriormente, tais como os objetivos-*soft* de Disponibilidade [Sistema], Adaptabilidade [Sistema] e Segurança [Sistema] (Figura 5.8). Para conduzir esta análise Tropos usa o Catálogo de Correlação apresentado na Tabela 5.1 [Kolp et al. 2001] que resume a avaliação dos estilos organizacionais na satisfação dos atributos de qualidade, realizada com o *framework* NFR (*Non-Functional Requirements*) [Chung et al. 2000]. Eventualmente, a análise da Tabela 5.1 nos permite escolher o estilo arquitetural *Joint Venture* para o estudo de caso *e-News*.

Usando a abordagem de [Bastos and Castro 2005] podemos atribuir a operacionalizacao dos requisitos do sistema para cada ator arquitetural de acordo com estilo *Joint Venture*. De fato, esta atribuição segue o princípios de projeto, tais como baixo acoplamento e alta coesão.

A Figura 5.9 apresenta a arquitetura do sistema *e-News* em i* aplicando o estilo *Joint Venture*. A arquitetura do sistema *e-News* é composta em cinco atores: Editor Chefe, *Webmater*, Editor, Repórter e Fotógrafo. Cada um dos atores arquiteturais está ligado ao ator sistema através do relacionamento *is-part-of*. No estilo *Joint Venture*, o ator Editor Chefe coordena tarefas e gerencia o compartilhamento de recursos entre os atores Editor, *Webmaster*, Reporter e Fotógrafo. Cada um deles pode se gerenciar e se controlar em uma dimensão local e podem interagir diretamente uns com os outros para trocar recursos, tais como dados e conhecimento. Entretanto, o Editor Chefe é o unico responsável pela operação e coordenação estratégica do sistema e de seus componentes em uma dimensao global.

Tabela 5.1 – Catálogo de Correlação

	Atributos de Qualidade								
Estilos	*Predicabilidade*	***Segurança***	***Adaptabilidade***	*Cooperatividade*	*Competitividade*	***Disponibilidade***	*Integridade*	*Modulardade*	*Aggregabilidade*
Flat Structure	--	--	-			+	+	++	-
Structure-in-5	+	+		+	-	+	++	++	++
Pyramid	++	++	+	++	-	+	--	-	
Joint-Venture	+	+	++	+	-	++		+	++
Bidding	--	--	++	-	++	-	--	++	
Takeover	++	++	-	++	--	+		+	+
Arm's-Length	-	--	+	-	++	--	++	+	
Hierarchical Contracting			+	+	+	+		+	+
Vertical Integration	+	+	-	+	_	+	--	--	--
Co-optation	-	-	++	++	+	--	-	--	

De acordo com a Figura 5.9, quando um usuário deseja ler notícias, ele acessa o *website* do jornal mantido por um ator *Webmaster*, que é responsável por atualizar a informação publicada. A informação a ser publicada é fornecida e autorizada pelo Editor Chefe. O Editor Chefe depende do Editor para obter as notícias preparadas de acordo com a pauta do jornal. Então, é necessário que o Editor Chefe compartilhe a pauta do jornal com os Editores, que são responsáveis por conseguir notícias de um tipo específico. Por exemplo, um Editor pode ser responsável por notícias sobre política enquanto outro pode ser responsável por notícias sobre esporte. Cada Editor seleciona um ou mais Reporteres e Fotografos para encontrar notícias de um assunto específico (e.g., basquete) de forma a ajudá-lo a cumprir suas sub-pautas (e.g., esporte). O Editor Chefe então edita as notícias fornecidas pelo Editor e repassa-as para o *Webmaster* publicá-las.

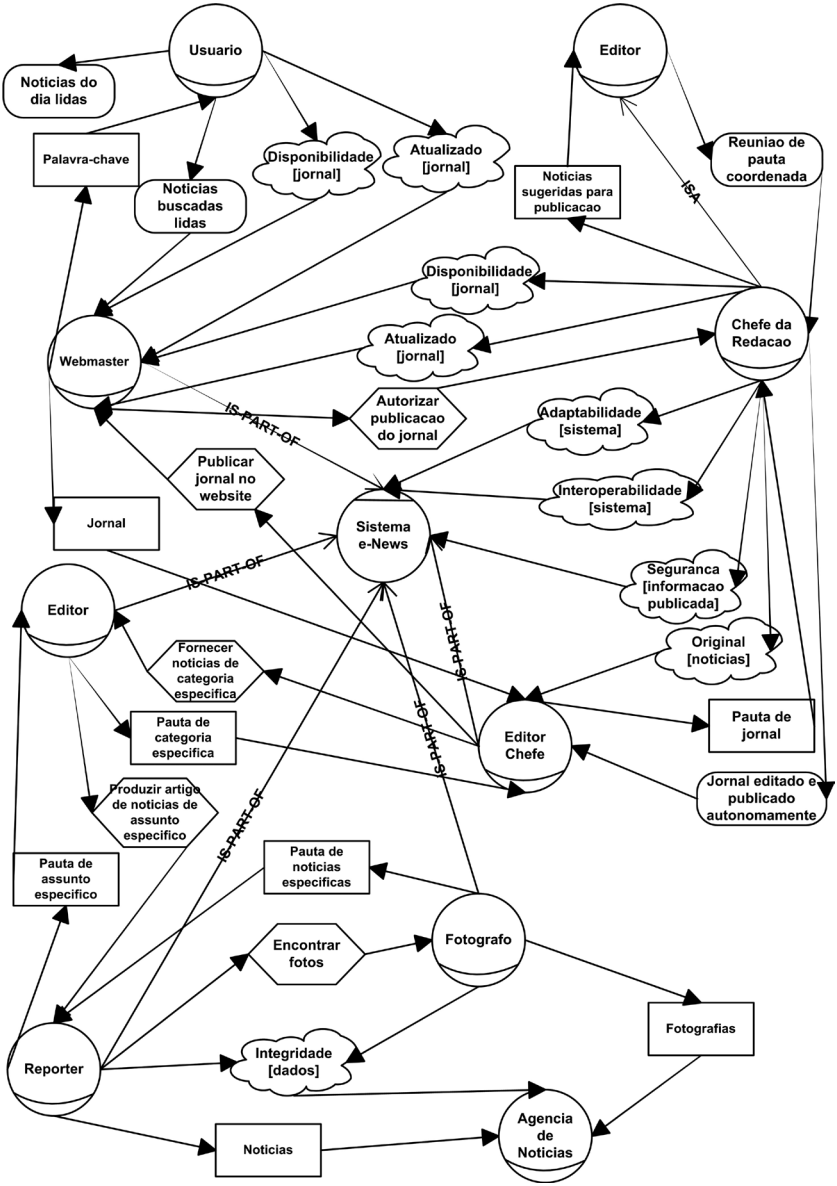

Figura 5.9. Modelo Arquitetural do *Sistema e-News*

5.4.3.4. Projeto Detalhado

A fase de projeto detalhado se foca em introduzir detalhe adicional a cada componente arquitetural do sistema. Ela consiste em definir como as metas atribuídas a cada ator presente no modelo arquitetural, são cumpridas por agentes de acordo com os padrões de projeto. Em particular, projetistas podem ser guiados por um catálogo de padrões de projeto que focam em aspectos sociais e intencionais que são recorrentes em sistemas multi-agentes cooperativos, os chamados padrões sociais [Kolp et al. 2005].

A fase de projeto detalhado ainda inclui a especificação da estrutura, comunicação e comportamento dos agentes adotando extensões da UML [Rumbaugh et al. 1999, OMG 2005] para suportar a especificação de sistemas multi-agentes, tais como a AUML [Bauer et al. 2000, Odell et al. 2001], entre outras [Silva et al. 2005].

Em particular, a proposta apresentada em [Kolp et al. 2005] conceitua um framework para explorar cinco dimensões complementares necessárias pra descrever os modelos de projeto detalhado refletindo aspectos particulares da arquitetura de sistemas multi-agentes, tais como as dimensões social, intencional, estrutural, comunicação e dinâmica. A primeira dimensão é descrita usando o framework i* [Yu 1995], a segunda dimensão descreve os serviços dos agentes em uma tabela, enquanto as últimas três dimensões são descritas usando extensões da UML [Rumbaugh et al. 1999, OMG 2005]. O diagram de classe UML é estendido para suportar o projeto de agente BDI (belief-desire-intention) [Rao and Georgeff 1995] e implementação em JACK [JACK 2005]. Consequente, esta abordagem também suporta a geração de código da especificação de agentes para a plataforma JACK. Entretanto, espera-se que a especificação da estrutura, comunicação e comportamento dos agentes seja tão detalhada quanto possível para direcionar a implementação em qualquer plataforma disponível.

5.4.3.5. Implementação

Implementar sistemas multi-agentes é inerentemente difícil. Por esta razão, muitas plataformas de implementação têm sido contruídas para simplificar o desenvolvimento de sistemas multi-agentes. Em geral, estas plataformas podem ser descritas como uma camada *middleware* onde os agentes podem executar.

Entre os vários ambientes disponíveis para suportar a implementação de sistemas multi-agentes, vamos detalhar três plataformas em conformidade com a FIPA e totalmente implementadas em Java [JAVA 2005]:

- FIPA-OS [FIPA-OS 2005] é uma plataforma baseada em componentes que possibilita rápido desenvolvimento de agentes. FIPA-OS suporta a maioria das especificações experimentais da FIPA e está sendo continuamente melhorada como um projeto *Open Source* gerenciado.

- JACK *Intelligent Agents* [JACK 2005] é um ambiente de desenvolvimento orientado a agentes projetado para estender Java com o modelo de agente BDI [Rao and Georgeff 1995]. Para suportar a programação de agentes BDI, JACK oferece cinco pricipais elementos da linguagem: agentes, capacidades, relações de banco de dados, eventos e planos. Capacidades agregam eventos, planos, bancos de dados ou outras capacidades, cada uma delas assumindo uma função específica atribuída a um agente. Relações de banco de dados armazenam crenças e dados de um agente. Eventos identificam as circunstâncias e mensagens a que um agente pode responder. Planos são instruções.
- JADE *Framework* (*Java Agent DEvelopment framework*) [Bellifemine et al. 2005] é um *middleware* que facilita o desenvolvimento de sistemas multi-agentes conforme os padrões da FIPA. Em JADE, um comportamento representa uma tarefa que um agente pode executar. Uma das mais importantes características dos agentes JADE é a habilidade de se comunicar. Mensagens trocadas por agentes JADE têm um formato especificado pela linguagem ACL (*Agent Communication Language*) definida pela FIPA [FIPA 2005]. Este formato compreende um número de campos e, em particular, o campo de performativa que indica a intenção comunicativa que o agente que envia a mensagem pretende atingir ao enviar a mensagem. O serviço de páginas amarelas em JADE (de acordo com a especificação da FIPA) é fornecido por um agente chamado DF (*Directory Facilitator*). JADE também fornece suporte para lingaugem de conteúdo e ontologias que, entre outras coisas, verifica se as mensagens sendo trocadas estão de acordo com as regras da ontologia definida [Bellifemine et al. 2005].

Da mesma forma que diversas ferramentas CASE, tais como *Rational Rose* [IBM 2006] e *Together* [Borland Software Corporation 2006], incluem geradores de código para padrões de projeto orientados a objetos, a ferramenta SKwyRL [Do et al. 2004] propõe um gerador de código para automatizar o uso dos padrões sociais propostos em [Kolp et al. 2005]. SKwyRL foi desenvolvida com a linguagem Java e produz o código abstrato para os padrões, gerando arquivos (*.agent, .event. .plan, .bel*) para a plataforma JACK. Ainda não existe um gerador de código que auxilie o mapeamento dos modelos de projeto detalhado do Tropos para os conceitos suportados pelas plataformas JADE e FIPA-OS.

5.4.4. Ferramentas CASE (*Computer Aided Software Engineering*)

Esta seção apresenta a principal ferramenta de apoio ao desenvolvimento de sistemas multi-agentes no contexto do *framework* Tropos, para o desenvolvimento dos modelos segundo a abordagem i*.

5.4.4.1. OME (Organizational Modeling Environment)

O Ambiente de Modelagem Organizacional (OME) [Yu and Liu 2005] é uma ferramenta para modelagem e análise, orientada a objetivos e/ou agentes, que ajuda ao desenvolvimento e representação de modelos desenvolvidos com os *frameworks* i* (*i-star*) e NFR (*Non-Functional Requirements*). Ela fornece uma interface gráfica, para que os usuários desenvolvam e modifiquem seus modelos facilmente. Quando o OME é iniciado, uma pequena janela (Figura 5.10) aparece. Essa é a janela usada para organizar os vários modelos desenvolvidos dentro de um projeto.

Essa ferramenta foi projetada de modo que os usuários mais habilidosos pudessem, facilmente, adaptá-la para suportar novos *frameworks* de modelagem. Para tanto, é permitido ao usuário o acesso à base de conhecimento onde está armazenada a semântica dos *frameworks*, bem como o acesso à rotinas de *plugins* (*Application Programming Interfaces* baseadas em Java). Desta forma, outros *frameworks* podem se incluídos ao ambiente, como é o exemplo do XGOOD [Alencar et al. 2006].

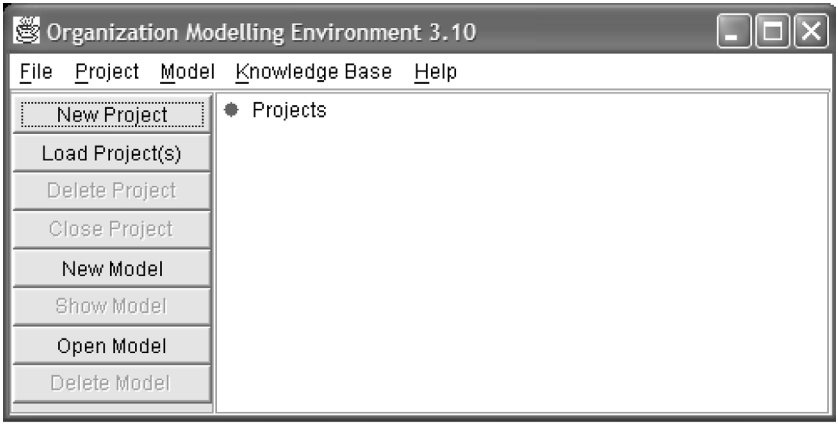

Figura 5.10. Janela inicial do OME

A ferramenta OME está sendo desenvolvida no *Knowledge Management Laboratory*, na Universidade de Toronto. Ela é parte do projeto intitulado, "Abordagem Orientada a Agentes para Arquitetura de Sistema: ferramentas de análise e modelos"; financiado pela *Communication and Information Technology*, de Ontário, Canadá, com suporte da *Mitel Corporation* [Yu and Liu 2005]. Atualmente, a OME encontra-se na versão 3 [Yu and Liu 2005], cuja última atualização de que se tem notícia data de outubro de 2004.

5.5. Conclusão

Esta seção aborda as principais barreiras e desafios para tornar a orientação a agentes um paradigma popular e bem difundido na comunidade de

engenharia de software, bem como discutir quais são os próximos passos na evolução do desenvolvimento de software orientado a agentes.

5.5.1. Estado Atual da Engenharia de Software Orientada a Agentes

Visando demonstrar a eficácia da abordagem orientada a agentes, o ideal seria demonstrar quantitativamente que a adoção desse paradigma ajuda a melhorar, de acordo com alguns conjuntos padrões de métricas de software, o processo de desenvolvimento de software. Entretanto, devido à imaturidade da área, estes dados simplesmente não estão disponíveis (assim como também não há dados precisos para paradigmas mais estabelecidos, tal como a orientação a objetos). Contudo, acredita-se que para certas classes de aplicação, uma abordagem orientada a agentes pode melhorar significantemente o processo de desenvolvimento de software [Jennings and Bussmann 2003]. Estes argumentos surgiram de uma década de experiência no uso da tecnologia de agentes para construir aplicações do mundo-real em larga-escala e em uma grande variedade de domínios industriais e comerciais [Luck et al. 2003].

No momento, existem algumas barreiras para que haja uma maior aceitação da tecnologia de agentes. É necessário estabelecer melhor a relação da tecnologia de agentes com a tecnologia antecedente (tecnologia orientada a objetos), apresentando essa nova tecnologia como uma extensão incremental de métodos conhecidos e confiáveis [Odell et al. 2001]. Também é preciso entender as situações em que soluções de agente são apropriadas [Jennings and Bussmann 2003]. Contudo, uma das maiores carências refere-se à ausência de metodologias sistemáticas e ferramentas de suporte que facilitem a especificação e estruturação de aplicações complexas como sistemas orientados a agentes.

5.5.2. Tendências Futuras

A seguir encontram-se expostos os principais obstáculos que devem ser superados para que a engenharia de software orientada a agentes se torne popular [Wooldridge and Ciancarini. 2001]:

Organizar a relação entre agentes e outros paradigmas – Não está claro como o desenvolvimento de sistemas multi-agentes irá coexistir com outros paradigmas de software, tal como o desenvolvimento orientado a objetos ou aspectos [Kiczales et al. 1997].

Metodologias orientadas a agentes – embora várias metodologias preliminares de análise e projeto orientados a agentes estejam sendo propostas, há comparativamente pouco consenso entre elas e nenhum acordo sobre os tipos de conceitos que a metodologia deveria suportar [Silva et al. 2004].

Engenharia para sistemas abertos – precisa-se de um melhor entendimento de como fazer engenharia em sistemas abertos. Em tais sistemas, acredita-se que é essencial ser capaz de reagir a eventos imprevistos, explorar oportunidades onde estes eventos surgem, e conseguir dinamicamente acordos com os componentes do sistema cuja presença não poderia ser predita em tempo de projeto.

Engenharia de escalabilidade – precisa-se de um melhor entendimento de como fazer de forma segura e previsível a engenharia de sistemas que consistem de um grande número de agentes interagindo dinamicamente uns com os outros a fim de atingir seus objetivos.

Referências

[Alencar et al. 2006] - Alencar, F., Pedroza, F., Castro, J., Silva, C. and Ramos, R. (2006) "XGOOD: A Tool to Automatize the Mapping Rules between i* Framework and UML", In: 9th Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS'06), Argentina.

[AMEC 2006] - AMEC (2006) "Agent-mediated electronic commerce", http://www.agentlink.org/activities/sigs/sig1.html, April.

[Bastos e Castro 2005] - Bastos, L. and Castro, J. (2005) "From Requirements to Multi-Agent Architecture based on Organisational Concepts", In: Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05), Saint Louis, EUA, ACM Press, p. 76 – 82.

[Bauer et al. 2000]-Bauer B., Müller, J. and Odell, J. (2000) "Agent UML: A Formalism for Specifying Multiagent Interaction", In: Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE'00), Edited by Ciancarini, P. and Wooldridge, M., Lecture Notes in Computer Science, Vol. 1957, Limerick, Ireland, Springer, p. 121 – 140.

[Bauer 2001]- Bauer, B. (2001) "UML Class Diagrams and Agent-Based Systems", In: Proceedings of the fifth international conference on Autonomous agents (AGENTS'01), Montréal, Quebec, Canada, ACM Press, p. 104 – 105.

[Bauer e Odell 2002] - Bauer, B. and Odell, J. (2002) "UML 2.0 and Agents: How to Build Agent-based Systems with the new UML Standard", In: Journal of Engineering Applications of AI, 18(2), p. 141 – 157.

[Bellifemine et al. 2003]- Bellifemine, F., Caire, G., Poggi, A. and Rimassa, G. (2003) "JADE - A White Paper**",** Special issue on JADE of the TILAB Journal EXP, http://jade.cselt.it/papers/2003/WhitePaperJADEEXP.pdf, September.

[Bihari et al. 1989] - Bihari, T., Gopinath, P. and Schwan, K. (1989) "Object-Oriented Design of Real-Time Software", In: Proceedings of the 10th Real-Time Systems Symposium, Santa Monica, California, IEEE Comp. Soc. Press, p. 194 – 201

[Booch et al. 1999] - Booch, G., Rumbaugh, J. and Jacobson, I. (1999) "The Unified Modeling Language User Guide", The Addison Wesley Object Technology Series, Addison Wesley.

[Borland 2006]-Borland Software Corporation (2006) "Borland Together", http://www.borland.com/together/, April.

[Bresciani et al. 2004]Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J. and Perini, A. (2004) "Tropos: An Agent -Oriented Software Development Methodology", In: International Journal of Autonomous Agents and Multi Agent Systems, 8 (3), Kluwer Academic Publishers, p. 203 – 236.

[Castro et al. 2001] - Castro, J., Mylopoulos, J. and Kolp, M. (2001) "Developing Agent-Oriented Information Systems for the Enterprise", In: Proceedings of the Second International Conference on Enterprise Information Systems, Stafford, UK, Kluwer Academic Publishers, p. 9 – 24.

[Castro et al. 2002] - Castro, J. Kolp, M., Mylopoulos, J. (2002) "Towards Requirements-Driven Information Systems Engineering: The Tropos Project", In: Information Systems Journal, Vol. 27, Elsevier, p. 365 – 89.

[Castro et al. 2003] - Castro, J., Silva, C. and Mylopoulos, J. (2003) “Detailing Architectural Design in the Tropos Methodology”, In: Proceedings of the 15Th Conference On Advanced Information Systems Engineering (CAiSE’03), Klagenfurt/Velden, Austria, p. 111 – 126.

[Chavez e Maes 1996] -Chavez, A. and Maes, P. (1996) “Kasbah: An Agent Marketplace for Buying and Selling Goods”, In: Proceedings of the 1st International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, p. 75 – 90.

[Chung et al. 2000] - Chung, L. K., Nixon, B., Yu, E. and Mylopoulos, J. (2000) “Non-Functional Requirements in Software Engineering”, Kluwer Publishing.

[DeLoach, 2006] - DeLoach, S. (2006) “Engineering Organization-based Multiagent Systems”, In: Proceedings of the 4th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS’05), Saint Louis, EUA, ACM Press, p. 1 – 7.

[Dignum e Dignum 2001] - Dignum, V. and Dignum, F. (2001) “Modelling agent societies: Co-ordination frameworks and institutions”, In: Proceedings of the 10th Portuguese Conference on Artificial Intelligence (EPIA’01), Edited by Brazdil, P. and Jorge, A., LNAI 2258, Berlin, Springer, p. 191 – 204.

[Do e Faulkner 2004] -Do T. T., Kolp M. and Faulkner, S. (2004) “Agent Oriented Design Patterns: The SKwyRL Perspective”, In: Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS’04), Porto, Portugal, p. 48 – 53.

[FIPA 2005] - FIPA (2005) “FIPA (The Foundation for Intelligent Physical Agents)”, http://www.fipa.org, December.

[FIPA-OS 2005] - FIPA-OS (2005) “A component-based toolkit enabling rapid development of FIPA compliant agents”, http://fipa-os.sourceforge.net/, March.

[Giorgini et al. 2005] - Giorgini, P., Kolp, M., Mylopoulos, J. and Castro, J. (2005) “Tropos: A Requirements-Driven Methodology for Agent-Oriented Software”, In: Agent-Oriented Methodologies, Edited by Henderson-Sellers, B. and Giorgini, P., Idea Group, p. 20 – 45.

[Green et al. 1997] - Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F. and Evans, R. (1997) “Software Agents: A Review”, Technical Report, TCD-CS-1997-06, The Intelligent Agents Group, Broadcom Eireann Reasearch Ltd, Trinity College, University of Dublin, http://www.cs.tcd.ie/research_groups/aig/iag/pubreview.ps.gz, December.

[IBM 2006] - IBM (2006) “Rational Rose”, http://www.ibm.com/rational/, April.

[ISO 1996]- ISO 9126. (1996) “Information Technology - Software quality characteristics and metrics”, ISO/IEC.

[JACK 2005] - JACK (2005) “JACK Intelligent Agents”, http://www.agent-software.com/, March.

[JAVA 2005] - JAVA (2005) Java Technology, http://www.java.sun.com/, March.

[Jennings e Bussmann 2003] - Jennings, N. and Bussmann, S. (2003) “Agent-based control systems”, IEEE Control Systems, 23(3) p. 61 – 74.

[Jennings et al. 2001]Jennings, N. and Wooldridge, M. (2001) “Agent-Oriented Software Engineering”, In: Handbook of Agent Technology, Edited by Bradshaw, J., AAAI/MIT Press.

[Kiczales 1997] - Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. and Irwin, J. (1997) “Aspect-Oriented Programming”, In: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP’97), Edited by Aksit, M. and Matsuoka, S., Lecture Notes in Computer Science, Vol. 1241, Jyväskylä, Fin-

land, Springer-Verlag, p. 220 – 242.

[Kolp et al. 2001] - Kolp, M., Castro, J. and Mylopoulos, J. (2001) "A social organization perspective on software architectures", In: Proceedings of the 1st International Workshop from Software Requirements to Architectures (STRAW'01), Toronto, Canada, p. 5 – 12.

[Kolp et al. 2002] - Kolp, M. Giorgini, P. and Mylopoulos, J. (2002) "Information Systems Development through Social Structures", In: Proceedings 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, ACM Press, p. 183 – 190.

[Kolp et al. 2005] - Kolp, M., Do, T., Faulkner, S. and Hoang, H. (2005) "Introspecting Agent Oriented Design Patterns", In: Handbook of Software Engineering and Knowledge Engineering, Vol. 3: Recent Advances, Edited by Chang, S. K. World Scientific, p. 151-176.

[Li et al. 2000] - Li, Y., Benwell, G., Whigham, P. and Mulgan, N. (2000) "An Agent-oriented software engineering paradigm and the design of a new generation of spatial information system", In: The 12th Annual Colloquium of the Spatial Information Research Centre, Edited by Whigham, P., Dunedin, New Zealand, p. 165 – 179.

[Luck et al. 2003] - Luck, M., McBurney, P. and Preist, C. (2003) "Agent technology roadmap: Enabling next generation computing", In: AgentLink report, http://www.agentlink.org/roadmap/index.html, December.

[Müller 1996] - Müller, J. (1996) "The Design of Intelligent Agents, A Layered Approach", Lecture Notes in Artificial Intelligence, Vol. 1177, Springer-Verlag.

[Müller 1998]- Müller, J. (1998) "The Right Agent (Architecture) to Do the Right Thing", In: Proceedings of the 5th International Workshop on Agent Theories, Architectures, and Languages (ATAL'98), LNCS, Vol. 1555, Springer-Verlag, p. 211 – 225.

[Odell, 2000] - Odell, J. (2000) "Agent Technology", Green paper produced by the OMG Agent Working Group. Document ec/2000-08-01, http://www.jamesodell.com/ec2000-08-01.pdf, December.

[Odell et al. 2001]Odell, J., Parunak, H. and Bauer, B. (2001) "Representing Agent Interaction Protocols in UML", In: Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE'01), Edited by Paolo Ciancarini and Michael Wooldridge, Lecture Notes in Computer Science, Vol. 1957, Limerick, Ireland, Springer, p. 121 – 140.

[OMG 2005] - OMG (2005) Unified Modeling Language (UML): Superstructure. Version 2.0, www.omg.org/docs/formal/05-07-04.pdf, December.

[Rao e Gerorgeff 1995] - Rao, A. and Georgeff, M. (1995) "BDI agents: from theory to practice", In: Technical Note 56, Australian Artificial Intelligence Institute.

[Sauvé 1999] - Sauvé, J. (1999) "Enterprise JavaBeans: Componentes para o Desenvolvimento de Sistemas Distribuídos Corporativos", http://www.dsc.ufcg.edu.br/~jacques/palestras/EJB-4horas.ppt

[Shaw e Garland 1993] - Shaw, M. and Garlan, D. (1993) "An Introduction to Software Architecture", In: Advances in Software Engineering and Knowledge Engineering, Vol. I, Edited by Ambriola, V. and Tortora, G., World Scientific Publishing Company, New Jersey.

[Silva e Lucena 20004] - Silva, V. and Lucena, C. (2004) "From a Conceptual Framework for Agents and Objects to a Multi-Agent System Modeling Language", In: Journal of Autonomous Agents and Multi-Agent Systems, Edited by Sycara, K. and Wooldridge, M., 9(1-2), Kluwer Academic Publishers, p. 145 – 189.

[Silva et al. 2004] - Silva, C., Tedesco, P., Castro, J., Pinto, R. (2004) "Comparing Agent-Oriented Methodologies Using a NFR Approach", In: Proceedings of the 3rd Software

Engineering for Large-Scale Multi-Agent Systems (SELMAS'04), Edinburgh, Scotland, p. 1 – 9.

[Silva et al. 2005] - Silva, V., Noya, R. and Lucena, C. (2005) "Using the UML 2.0 activity diagram to model agent plans and actions", In: 4th Autonomous Agents and Multi-Agent Systems (AAMAS'05), Uthrecht, The Netherlands, p. 594 – 600.

[Silva et al 2005] - Silva, C., Castro, J., Tedesco, P. and Silva, I. (2005) "Describing Agent-Oriented Design Patterns in Tropos", In: Proceedings of the 19th Brazilian Symposium on Software Engineering (SBES'05), Minas Gerais, Brazil, p. 10 – 25.

[Silva et al. 2006] - Silva, C., Castro, J., Tedesco, P., Araújo, J., Moreira, A., Mylopoulos, J. (2006) "Improving the Architectural Design of Multi-Agent Systems: The Tropos Case", In: 5th Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'06), Shangai, China (to appear).

[Sturm e Shehory 2003] - Sturm, A. and Shehory, O. (2003) "A Framework for Evaluating Agent-Oriented Methodologies", In: Proceedings of the International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS'03), LNCS, Vol. 3030, Springer-Verlag, p. 94 – 109.

[SUN 2005] - SUN (2005) "Java Remote Method Invocation (Java RMI)", http://java.sun.com/products/jdk/rmi/, December.

[Tahara et al. 1999] -Tahara, Y., Ohsuga, A. and Honiden, S. (1999) "Agent System Development Method Based on Agent Patterns", In: The 21st International Conference on Software Engineering (ICSE'99), Los Angeles, CA, USA, p. 356 – 367.

[Weiss 2000] - Weiss, G. (2000) "Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence", MIT Press, p. 27 – 78.

[Wooldridge e Jennings 1995] - Wooldridge, M. and Jennings, N. R. (1995) "Intelligent agents: theory and practice", The Knowledge Engineering Review, 10(2), p. 115 – 152.

[Wooldridge e Ciancarini 2001] - Wooldridge, M. and Ciancarini, P. (2001) "Agent-Oriented Software Engineering: The State of the Art", In: Agent-Oriented Software Engineering, Edited by Ciancarini, P. and Wooldridge, M., Springer-Verlag, Lecture Notes in AI, Vol. 1957.

[Wooldridge 2002] - Wooldridge, M. (2002) "An Introduction to Multiagent Systems", John Wiley and Sons, Ltd., England, p. 15 – 103.

[Yu 1995]- Yu, E. (1995) "Modelling Strategic Relationships for Business Process Reengineering", Ph.D thesis. Dept. of Computer Science, University of Toronto, Canada.

[Yu 2002]- Yu, E. (2002) "Agent-Oriented Modelling: Software Versus World", In: Proceedings of the Agent-Oriented Software Engineering (AOSE'01), Edited by Wooldridge, M., Weiss, G. and Ciancarini, P., LNAI, Vol. 2222, Springer-Verlag, p. 206 – 225.

[Yu e Liu 2005] - Yu, E. and Liu, L. (2005) "OME (Organization Modeling Environment)", http://www.cs.toronto.edu/km/ome, December.

[Zambonelli et al. 2000] - Zambonelli, F., Jennings, N., Omicini, A. and Wooldridge, M. (2000) "Agent Oriented Software Engineering for Internet Applications", In: Coordination of Internet Agents: Models, Technologies and Applications, Edited by Omicini, A., Zambonelli, F., Klush, M. and Tolksdorf, R., Springer Verlag, p. 326 – 346.

[Zambonelli et al. 2003] - Zambonelli, F., Jennings, N., Wooldridge, M. (2003) "Developing Multiagent Systems: the Gaia Methodology", In: ACM Transactions on Software Engineering and Methodology, 12(3), p. 317 – 370.