UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

ÁREA DE CIÊNCIA DA COMPUTAÇÃO

ALISSON NERES LINDOSO

UMA METODOLOGIA BASEADA EM ONTOLOGIAS PARA A ENGENHARIA DE APLICAÇÕES MULTIAGENTE

São Luís
2006

ALISSON NERES LINDOSO

UMA METODOLOGIA BASEADA EM ONTOLOGIAS PARA A ENGENHARIA DE APLICAÇÕES MULTIAGENTE

Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Orientadora: Profa. Dra. Rosario Girardi

São Luís
2006

Lindoso, Alisson Neres

Uma metodologia baseada em ontologias para a engenharia de aplicações multiagente / Alisson Neres Lindoso. – São Luís, 2006.

268 f.

Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão.

Orientadora: Rosario Girardi

1. Sistemas multiagente. 2. Reutilização de software. 3. Ontologia. I. Título.

CDU: 004.891

ALISSON NERES LINDOSO

UMA METODOLOGIA BASEADA EM ONTOLOGIAS PARA A ENGENHARIA DE APLICAÇÕES MULTIAGENTE

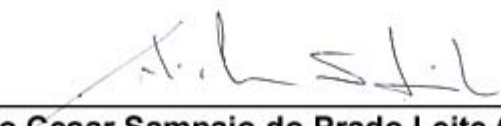
Dissertação de Mestrado apresentada à Coordenação do Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Aprovada em 10 / 03 / 2006

BANCA EXAMINADORA

Profª Drª Maria Del Rosario Girardi Gutierrez (DEINF/UFMA)
Orientadora

Prof. Dr. Francisco José da Silva e Silva (DEINF/UFMA)
Examinador Interno



Prof. Dr. Julio Cesar Sampaio do Prado Leite (LES/PUC-Rio)
Examinador Externo

A Marcelle Barros dos Santos

AGRADECIMENTOS

Ao meu pai Cleverson, à minha mãe Cleonice, aos meus irmãos Alessandro e Cleverson e à minha irmã Lídia, que constituem meu círculo familiar mais próximo e que me fornecem o necessário suporte para tudo;

à minha namorada Marcelle, por ser uma *garotinha* tão especial, que me apóia em todas as situações, sejam pessoais sejam profissionais, sempre na medida exata, sem precisar se esforçar muito para isso;

à professora Rosario Girardi, que vem acompanhando minha evolução acadêmica ao longo dos últimos cinco anos, contribuindo para ela mais que qualquer outra pessoa;

à professora Maria Auxiliadora, quem primeiro me estimulou a experimentar a pesquisa científica, sendo parte fundamental nesse ciclo que ora se encerra;

aos professores Francisco Silva e Julio Leite, que aceitaram compor minha banca examinadora e cujas críticas certamente servirão para engrandecer minha pesquisa;

aos meus professores e às minhas professoras, das Graduações e da Pós-Graduação, que, com seus ensinamentos, possibilitaram-me a formação exigida para o sucesso em tal intento;

aos membros da Coordenação, empenhados em conduzir o Programa satisfatoriamente, tanto do ponto de vista acadêmico quanto administrativo.

ao Alcides, secretário da Coordenação, que sempre se portou de maneira bastante atenciosa e solícita, facilitando o trabalho de todos nós;

à UFMA, ao CNPq e à CAPES, por terem custeado minha dedicação integral aos estudos e às pesquisas, desde a Iniciação Científica, através das bolsas a mim concedidas;

aos meus ex-colegas do grupo de pesquisa GESEC – *Carlinha* Faria, Ivo Serra, Ismênia Oliveira, Leandro Balby, Rafael Robinson, Steferson Ferreira, Geovane Bezerra e José Henrique – com os quais eu muito convivi e que, bem desempenhando suas obrigações, fizeram com que as minhas também assim resultassem;

aos meus atuais colegas do grupo de pesquisa GESEC – Lucas Drumond, Raimundo Osvaldo, Francislene Barros, Jone Correia, Inacio Bouéres e André Santos – próximo dos quais estive trabalhando bastante nos últimos meses e que participaram direta ou indiretamente do andamento desta dissertação, tanto com idéias quanto com produção, de modo secundário, porém essencial;

aos meus colegas do Mestrado – Mauro Jansen, Bysmarck Barros, Rafael Fernandes, Cícero Costa e José Mendes – que estiveram comigo desde minha admissão no Programa e com os quais mantenho um amistoso vínculo até o momento presente;

aos meus colegas do Curso de Ciência da Computação – Antoniel Carvalho, Geraldo Costa, Henry Franklin, Nírondes Tavares e Clauber Muniz – com os quais estudei e desenvolvi diversos trabalhos em equipe;

aos meus colegas do Curso de Direito – Flábio Marcelo, Denys Ronald, Sílvio Romero, Maria Emanuela e Maria Inêz – por quem criei uma grande estima e que também foram importantes para a realização deste trabalho, posto que participaram da minha formação jurídica, a qual aqui foi em parte aplicada;

aos meus amigos – Alexandre Henrique, Maurício Pinheiro, Rânide Varão, Fernando Moreira e Rômulo Souza – e às minhas amigas – Luciana Castro, Thalisse Ramos, Lucélia Costa, Ariadne Garcez, Patrícia Aquino e Aline Vieira – de universidades e cursos variados, mas que juntos partilhamos o saudoso envolvimento com o *Movimento Empresa Junior*, o qual nos propiciou uma forte e duradoura amizade;

à minha tia Iracema e à minha prima Gláucia, que por inúmeras vezes cederam a tranqüilidade de seu lar para que eu pudesse desenvolver meus trabalhos com a concentração demandada;

àqueles e àquelas que foram de alguma forma importantes para a elaboração deste estudo e que têm consciência disso.

“Entrar na briga pra ganhar!!!”
anônimo

RESUMO

A crescente demanda por aplicações de software cuja construção concilie produtividade, baixo custo e alta qualidade, mesmo em domínios complexos e mutáveis, torna necessária a elaboração de técnicas e metodologias que foquem paradigmas de desenvolvimento mais adequados para abordar aquelas características conflitantes, tal como o paradigma multiagente. Por outro lado, o processo de reutilização de software permite promover a criação de novas aplicações empregando artefatos de software reutilizáveis previamente desenvolvidos. Esse trabalho introduz a MAAEM, uma metodologia baseada em ontologias para a análise, o projeto e a implementação de aplicações multiagente através do reuso de modelos e componentes que representam os requisitos de uma família de aplicações em um domínio, assim como as correspondentes soluções orientadas a agentes para tais requisitos. É também apresentada a ONTORMAS, uma ontologia cuja instanciação é útil para modelar e representar aplicações específicas desenvolvidas com a metodologia MAAEM. São descritos ainda dois estudos de caso elaborados no sentido de avaliar a metodologia e a ontologia, explorando os casos com e sem reuso, respectivamente, nos domínios turístico e jurídico.

Palavras-chave: Reutilização de software. Ontologias. Sistemas Multiagente. Processos de desenvolvimento de software. Metodologias de desenvolvimento de software. Turismo. Direito.

ABSTRACT

The increasing demand of software applications constructed conciliating productivity, low cost and high quality, even in complex and changeable domains, turns necessary the elaboration of techniques and methodologies focusing on development paradigms more suitable for approaching these conflicting features, like the multi-agent one. On the other hand, the sotware reuse process promotes the creation of new applications employing reusable software artifacts previously developed. This work introduces MAAEM, an ontology-driven methodology for analysis, design and implementation of multi-agent applications through the reuse of models and components that represent the requirements of a family of applications in a domain as well as the corresponding agent-oriented solutions to these ones. ONTORMAS, an ontology whose instantiation is useful for modeling and representing specific applications developed with MAAEM methodology, is also presented. Two case studies elaborated in order to evaluate the methodology and ontology are also described, exploring the cases with and without reuse, respectively, in the touristic and juridical domains.

Keywords: Software reuse. Ontologies. Multi-agent systems. Software development processes. Software development methodologies. Tourism. Law.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e problemática

A presente dissertação de mestrado enfoca a Engenharia de Aplicações Multiagente, dentro do escopo do Desenvolvimento de Software baseado na Reutilização.

A Engenharia de Aplicações é um processo de desenvolvimento de software através da reutilização de modelos e componentes previamente construídos no processo complementar denominado Engenharia de Domínio.

O paradigma multiagente, por sua vez, introduz um modo bastante interessante de se abordar a crescente complexidade das aplicações desenvolvidas, ainda mais se considerando a necessidade de oferecer soluções computacionais baratas, rápidas e eficientes para os problemas do mundo real.

Na Engenharia de Aplicações Multiagente, pretende-se explorar a elaboração e a devida avaliação de uma metodologia – isto é, um conjunto sistemático de técnicas integradas relativas às diversas fases do ciclo de desenvolvimento – para a construção de aplicações compostas por agentes de software. As ontologias serão empregadas para a representação do conhecimento envolvido nesse processo, devido a diversas vantagens que apresentam, como se constatará ao longo do trabalho.

1.2 Relevância e motivação

No plano geral, a Engenharia de Aplicações Multiagente é um tópico que merece ser profundamente pesquisado porque constitui, provavelmente, uma forma promissora de se desenvolver aplicações de razoável complexidade, mantendo-se um equilíbrio entre a qualidade e a produtividade, características sempre desejáveis, mas muitas vezes não alcançadas.

Isto é, tem-se como grande diferencial o reaproveitamento do trabalho anteriormente realizado quando do estudo do domínio de aplicação, que resulta em um conjunto de modelos prontos para o uso na construção de aplicações.

Por já terem sido amplamente testados aqueles modelos, resta minimizada a quantidade de testes e correções necessárias para se ter uma

aplicação acabada, agregando-lhe qualidade sem acarretar desperdícios, os quais implicam invariavelmente em aumento de custos.

Já de maneira particular, pesquisar a Engenharia de Aplicações Multiagente é interesse do GESEC (2005) porque representa a continuidade dos demais trabalhos que já vêm sendo atualmente desempenhados pelo grupo. No ponto em que se chegou, já há disponíveis técnicas, produtos e ferramentas para a Análise e o Projeto de Domínio, bem como para a Especificação de Linguagens Específicas de Domínio, tudo no contexto da Engenharia de Domínio Multiagente, servindo como subsídios para a Engenharia de Aplicações Multiagente.

Por fim, o que motiva a realização deste trabalho, tornando-o distinto de tantos outros em estágios relativamente mais avançados, é o fato de utilizar, ao mesmo tempo, ontologias e agentes para a construção de abstrações de software e seu reuso no desenvolvimento de aplicações específicas.

1.3 Objetivos

O objetivo geral do presente trabalho é desenvolver e avaliar uma metodologia baseada em ontologias para a Engenharia de Aplicações Multiagente.

No sentido de alcançar tal objetivo pretendido, buscar-se-á atingir os seguintes objetivos específicos:

a) Aprimorar as técnicas, ferramentas e artefatos relacionados com a Engenharia de Domínio Multiagente;
b) Elaborar uma metodologia baseada em ontologias para as fases de análise, projeto e implementação da Engenharia de Aplicações Multiagente;
c) Avaliar a metodologia proposta através do desenvolvimento de estudos de caso comparativos, tanto com quanto sem reutilização.

1.4 Estruturação

Para tanto, o Capítulo 2 traz uma revisão de conceitos referentes à Reutilização de Software, englobando desde os domínios de aplicação até as ontologias com estruturas de representação de abstração de software, passando ainda pelas famílias de sistemas e linhas de produção e pelos processos e

abordagens de reuso de software, isso posto que tal se configura como um dos tópicos mais importantes para o embasamento do presente trabalho.

Já o Capítulo 3, aborda a Engenharia de Software Multiagente, assunto igualmente fundamental no contexto da nossa pesquisa, onde explica-se as noções de agentes de software e de sistemas multiagente; sintetiza-se as fases, passos e produtos de um típico processo de desenvolvimento orientado a agentes; discute-se o reuso no desenvolvimento de software multiagente; apresenta-se levantamento de metodologias e ferramentas para a construção de sistemas multiagente; e, enfim, mostra-se um estudo comparativo dos trabalhos levantados.

O Capítulo 4, por sua vez, entrando no cerne propriamente desta dissertação, descreve a metodologia para a Engenharia de Aplicações Multiagente por nós elaborada, a qual é denominada *MAAEM* e que se caracteriza como a principal contribuição científica resultante deste trabalho. Também relevante é a chamada *ONTORMAS*, uma ontologia que oferece suporte ao desenvolvimento através da nossa metodologia. Ainda aqui, são explicadas passo a passo e por meio de exemplos cada uma das três fases constantes do processo de desenvolvimento, que são a Análise, o Projeto e a Implementação da Aplicação.

Em seguida, os Capítulos 5 e 6 apresentam dois estudos de casos executados com o intuito de avaliar a nossa metodologia, respectivamente, nas hipóteses com e sem reuso de artefatos de software reutilizáveis resultantes do processo da Engenharia de Domínio Multiagente. Assim, no primeiro, é vista a *RecomTour* – Uma Aplicação Multiagente para a Recomendação de Pacotes Turísticos através da Mineração de Uso na Web e da Filtragem Colaborativa, e no segundo, a *InfoNorma* – Uma Aplicação Multiagente para a Entrega Personalizada de Instrumentos Jurídico-Normativos através da Filtragem de Informação baseada no Conteúdo.

Por fim, além do Capítulo 7, em que são expostas as nossas conclusões acerca do trabalho, com ênfase nas principais contribuições, nos resultados alcançados e nas perspectivas futuras, há o ANEXO 1 e o ANEXO 2, os quais trazem, nesta ordem, a hierarquia de classes da ontologia *ONTORMAS* e listagem de código-fonte da aplicação *InfoNorma*, e o APÊNDICE A, que exibe as instâncias da modelagem da *ONTOWUM*, sendo o modelo de domínio (*DM*) e o framework multiagente (*DD*).

2 A REUTILIZAÇÃO DE SOFTWARE

2.1 Considerações iniciais

A crise de software foi uma situação que surgiu em meados da década de setenta em função do aumento na complexidade das aplicações, bem como por um crescimento contínuo na demanda, inclusive em novas áreas, o que implicou diretamente na elevação dos custos de produção e manutenção, afetando ainda a qualidade do software construído (GIRARDI, 2004).

A reutilização de software permite enfrentar tais problemas porque leva ao aumento da produtividade através do emprego de componentes previamente desenvolvidos e testados, causando uma diluição dos custos dentre várias aplicações construídas a partir da mesma base comum, bem como facilitando a manutenção tanto corretiva quanto evolutiva.

Dessa forma, a reutilização de software se caracteriza pela existência de dois processos distintos e complementares durante o ciclo de desenvolvimento: a produção de artefatos reutilizáveis e o seu reuso na construção de aplicações específicas, do modo a seguir detalhado.

2.2 Domínio de aplicação

O termo *domínio* recebe diferentes definições nas diversas disciplinas em que está presente, tais como na Lingüística, Inteligência Artificial, Orientação a objetos, Reuso de software etc, e todas tendem para uma das duas visões a seguir: domínio como o mundo real, ou seja, uma área ou campo de especialização onde o conhecimento humano é usado; e domínio como um conjunto de sistemas, isto é, família de sistemas de software similares.

Assim, um domínio pode ser visto como um conjunto de problemas para os quais as soluções são providas pelas aplicações a serem desenvolvidas. Ou ainda, como uma área de conhecimento delimitada para satisfazer os requisitos dos interessados e que inclui um conjunto de conceitos e terminologia entendida pelos praticantes da área e um conhecimento de como fazer sistemas ou partes de sistemas de softwares nessa área (CZARNECKI, EISENECKER, 2000, p. 40-44).

Para defini-lo, deve-se determinar o que está dentro dele, como conceitos e relacionamentos ou mesmo subdomínios; o que está fora, através de contra-exemplos de aplicações não pertencentes ao domínio; e, por fim, as regras de inclusão e exclusão de elementos a ele relacionados (HARSU, 2002).

2.3 Famílias de sistemas e linhas de produção de software

As pesquisas e experiências no contexto da reutilização de software, à proporção que avançam, têm demonstrado que a tentativa de reuso de um mesmo artefato em um número grande de sistemas é muitas vezes impraticável. Isso porque na medida em que se aumenta a reusabilidade de certo componente de software, ou seja, quando cresce a possibilidade de ele ser reutilizado em aplicações bastante diversificadas, diminui-se sua funcionalidade, isto é, ele passar a contemplar menos funções significativas, tendo que abranger apenas problemas mais gerais.

Por outro lado, também vem sendo provado que é improdutivo continuar desenvolvendo aplicações de software tão somente de maneira individual, devendo-se promover a reutilização sempre que seja possível, valendo-se das semelhanças que elas guardam entre si quando consideradas em conjunto.

Essa visão vem levando os engenheiros de software a tentar migrar para o desenvolvimento de famílias de sistemas, em contraponto à tradicional construção de sistemas individuais. Nessa perspectiva, busca-se explorar as semelhanças existentes entre as diversas aplicações de um determinado domínio, ao mesmo tempo em que se procura as diferenças de uma maneira sistemática. Assim, novos sistemas podem ser rapidamente criados com base em um conjunto de artefatos reusáveis, tais como arquiteturas, componentes e modelos (CZARNECKI, EISENECKER, 2000).

Desse modo, a expressão *família de sistemas* é normalmente empregada para denotar um conjunto de aplicações pertencentes a um mesmo domínio, as quais compartilham requisitos e soluções comuns. Associada a essa, costuma-se falar de *linha de produção*, que pode entendida como uma forma de promover o reuso de software por meio da construção conjunta de aplicações que apresentam características similares, mas que também designa o conceito de família de sistemas sob um ponto-de-vista mais mercadológico (HARSU, 2002).

Famílias de sistemas e linhas de produção de software caracterizam a reutilização vertical, ou seja, que ocorre dentro de um domínio, como o jurídico, turístico, médico, dentre outros, em oposição ao reuso horizontal, que se relaciona a uma dada funcionalidade presente em várias áreas de aplicação, como a interação com usuários, persistência de dados etc (CZARNECKI, EISENECKER, 2000).

2.4 Abordagens para a reutilização de software

Como já se observou inicialmente, o desenvolvimento de software baseado na reutilização passa por dois processos complementares: um primeiro, quando são produzidas as bases de software para o reuso; e um segundo, na qual se constroem as aplicações propriamente ditas (GIRARDI, 2001).

Tal processo de reutilização pode ser realizado sob as abordagens composicional ou gerativa, conforme seja o reuso feito de forma manual ou automática (CZARNECKI, EISENECKER, 2000).

2.4.1 Reutilização composicional

A reutilização composicional é uma forma de desenvolvimento de novos sistemas, consoante diretrizes pré-estabelecidas, a partir tanto de modelos quanto de componentes propriamente ditos, que normalmente se encontram armazenados em bibliotecas de software.

A realização do reuso por composição exige, portanto, em um primeiro momento, que os componentes sejam produzidos já se pensando na reutilização futura, para em seguida serem classificados e armazenados em repositórios de componentes de software.

Tais componentes devem ser considerados relativamente estáveis, de modo a poderem ser reutilizados com segurança, porém, novos componentes podem surgir a partir da modificação de outros previamente construídos, assim como podem se originar de novas idéias ou da agregação de componentes menores.

As vantagens da abordagem composicional são a larga aplicabilidade dos componentes no desenvolvimento de variados sistemas e a facilidade de sua manutenção, devido ao fato de serem eles modulares independentes entre si.

De outro lado, componentes raramente são perfeitos, acabados e suficientemente gerais, assim, quase sempre se tem que adaptá-los antes da composição final. Outra desvantagem, decorrente dessa, é a queda de produtividade devida ao tempo despendido nessas modificações, além do crescimento exagerado das bases de componentes, muitas vezes havendo vários que apresentam apenas pequenas variações entre si.

2.4.2 Reutilização gerativa

A reutilização gerativa é um processo automatizado de desenvolvimento de software baseado em geradores de aplicações, que são utilitários que escondem do desenvolver a tarefa de interconexão de componentes, a qual, de outro modo, seria realizada manualmente (CZARNECKI, EISENECKER, 2000).

Nesse caso, o sistema é escrito através não de uma linguagem de propósito geral, mas de uma específica para o domínio abordado, elevando o nível de abstração de tratamento do problema. Uma vez especificada a aplicação, o gerador se encarrega de produzir o código de programação desejado.

A geração automática de aplicações leva bastante em consideração as semelhanças e diferenças entre os diversos membros de uma família em um domínio, visto que as partes invariáveis dos sistemas devem constar de uma estrutura que não sofre mudanças durante o desenvolvimento, apenas são reutilizadas como estão. Já os pontos variantes devem ser tratados particularmente em cada caso.

A abordagem gerativa para o reuso de software permite alcançar uma melhor produtividade. Nos geradores baseados em linguagens específicas de domínio, que ao mesmo tempo descrevem o problema e ocultam a implementação, escrever uma aplicação por meio dessas linguagens permite reusar padrões de implementação de linguagens de programação do mesmo modo que essas o fazem em relação aos padrões de montadores de código. A vantagem decorrente disso é que tais padrões reutilizáveis podem ser projetados cuidadosamente por programadores mais experientes.

Em contrapartida, a geração automática tem a desvantagem de ser aplicável somente a certos domínios, e em poucas situações, sendo diversas vezes ou especifica ou geral demais para o desenvolvimento pretendido.

2.5 Processos do reuso de software

A construção de aplicações de software através da reutilização envolve dois processos distintos e complementares, que são o *desenvolvimento para reuso* e o *desenvolvimento com reuso*. Ambos são também denominados, respectivamente, *Engenharia de Domínio* e *Engenharia de Aplicações*, sendo a primeira responsável por fornecer artefatos reusáveis ao segundo, enquanto que este retroalimenta aquele com novos requisitos e novas soluções (CZARNECKI, EISENECKER, 2000).

A Engenharia de Domínio passa pelas fases de Análise de Domínio, Projeto de Domínio e Implementação de Domínio, já a Engenharia de Aplicações compreende as fases de Análise de Aplicações, Projeto de Aplicações e Implementação de Aplicações (GEYER, BECKER, 2002).

É interessante destacar que cada uma dessas fases admite insumos e resulta em produtos, os quais alimentam as fases seguintes, tanto de maneira linear, ou seja, no mesmo processo, quanto de modo colateral, isto é, no outro processo, conforme se observa na Figura 2.1 e é detalhado nas subseções a seguir.

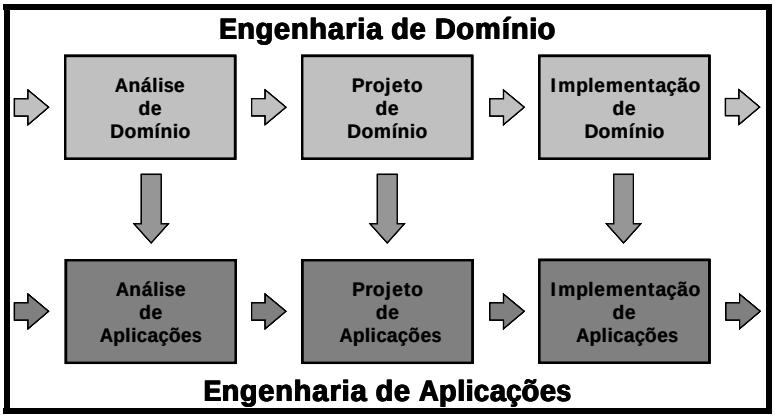


Figura 2.1: Processos da Engenharia de Domínio e da Engenharia de Aplicações

2.5.1 Engenharia de Domínio

A Engenharia de Domínio é um processo de criação de subsídios para a Engenharia de Aplicações em uma família de sistemas similares, cobrindo todas as atividades de construção de artefatos de software reutilizáveis, tais como componentes, geradores de aplicações, linguagens específicas de domínio etc (WITHEY, 1994) (CMU-SEI, 2004).

Similarmente ao que ocorre no desenvolvimento de sistemas tradicional, na Engenharia de Domínio também estão presentes as fases de análise, projeto e implementação, entretanto, elas objetivam a construção de classes de sistemas.

A Engenharia de Domínio se diferencia, entretanto, em relação ao tipo de domínio, se vertical ou horizontal. Quando voltada para o primeiro caso, isto é, domínios de sistemas completos, resulta principalmente em arquiteturas reutilizáveis de sistemas como um todo. Já no segundo caso, ou seja, dos domínios de partes de sistemas, os produtos são componentes reusáveis os quais, uma vez juntados, formarão novos sistemas (HARSU, 2002).

2.5.1.1 Análise de Domínio

A Análise de Domínio (Figura 2.2) tem o objetivo de identificar, coletar, organizar e representar as informações relevantes no domínio, a partir do estudo de sistemas pré-existentes e de seus históricos de desenvolvimento, de conhecimento de especialistas, teorias relativas e novas tecnologias surgidas no domínio (KANG *et al*., 1990).

Assim, essa tarefa compreende a precisa delimitação das fronteiras do domínio em análise, a identificação das semelhanças e diferenças entre os sistemas em questão, a compreensão dos relacionamentos entre os vários elementos do domínio e a representação disso tudo de maneira adequada ao futuro reuso (NILSON *et al*., 1990).

A Análise de Domínio, portanto, envolve a determinação do escopo de uma família de aplicações, identificando as características comuns e as variáveis entre os seus membros. O resultado é a especificação estrutural e comportamental da família em um modelo de domínio (CZARNECKI, EISENECKER, 2000).

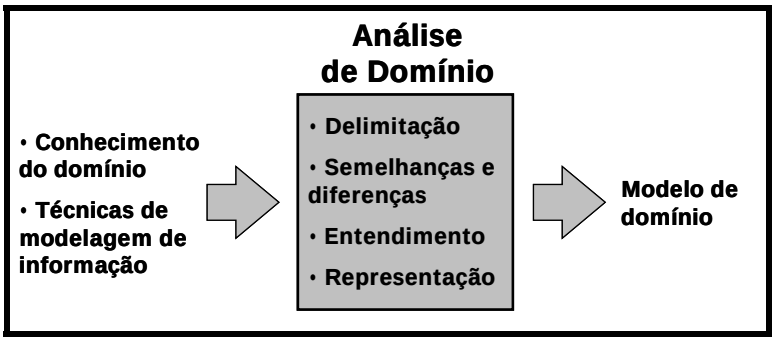


Figura 2.2: Fase de Análise do processo da Engenharia de Domínio

2.5.1.2 Projeto de Domínio

O Projeto de Domínio (Figura 2.3) é o meio através do qual se constrói um modelo de projeto a partir dos artefatos anteriormente produzidos e do conhecimento adquirido no estudo dos requisitos do software.

Desse modo, ele cobre o desenvolvimento de uma arquitetura compartilhada e um plano de como sistemas individuais da família serão criados como base nos artefatos previamente elaborados (CZARNECKI, EISENECKER, 2000).

A idéia básica por trás do Projeto de Domínio é desenvolver e documentar uma arquitetura genérica o bastante para a maioria dos sistemas no domínio e que suporte o reuso de componentes de código do domínio.

O produto resultante é um Modelo de Projeto, que representa a arquitetura genérica desenvolvida e provê o arcabouço para o desenvolvimento de componentes reutilizáveis durante a Implementação de Domínio.

A arquitetura genérica é representada pelo padrão arquitetural mais adequado para a família de produtos esperados. Ele define a plataforma computacional para todas as aplicações nessa família e orienta a elaboração de um projeto genérico, resultando em uma estratégia de particionamento, que permite alocar características do domínio a elementos de software, e em um modelo de coordenação, que descreve como esses elementos são ativados e compartilham informação.

A estratégia de particionamento define o conjunto de elementos de software disponíveis e como são a eles atribuídas características do domínio. Tal alocação pode ocorrer por entidades reais, processos de controle, modelos de usuário ou por objetos. Em determinados contextos, algumas estratégias podem ser mais adequadas que outras e a sua seleção depende em parte da maioria dos fatores de mudança identificados nos modelos de domínio.

Um modelo de coordenação estabelece os protocolos de ativação e comunicação para elementos de software que implementam o modelo de comportamento. Normalmente ele define os tipos de interações permitidas entre os elementos em uma hierarquia de abstrações, assim como os mecanismos que dão suporte a essas interações. A aplicação deste aos modelos de domínio permite

identificar as operações internas do sistema, juntamente com os mecanismos do sistema e seus parâmetros de operação.

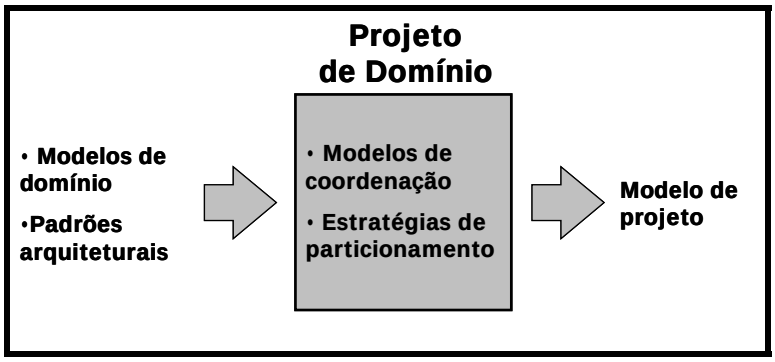


Figura 2.3: Fase de Projeto do processo da Engenharia de Domínio

2.5.1.3 Implementação de Domínio

A Implementação de Domínio (Figura 2.4) é um processo de identificação de componentes reutilizáveis, para a abordagem composicional – e também de linguagens específicas de domínio e geradores de aplicação, para a abordagem gerativa – baseados no modelo de domínio, produzido a partir do conhecimento adquirido na Análise de Domínio, e na arquitetura genérica, resultante do Projeto de Domínio (CZARNECKI, EISENECKER, 2000).

Assim, os Engenheiros de Domínio criam e catalogam em bibliotecas os artefatos reusáveis, para que sejam utilizados pelos Engenheiros de Aplicações. A criação, o gerenciamento e a manutenção de repositórios de artefatos reutilizáveis também são importantes nesse processo. Ao lado dos geradores de aplicações e das linguagens de domínio, tais componentes constituem os principais produtos da Engenharia de Domínio.

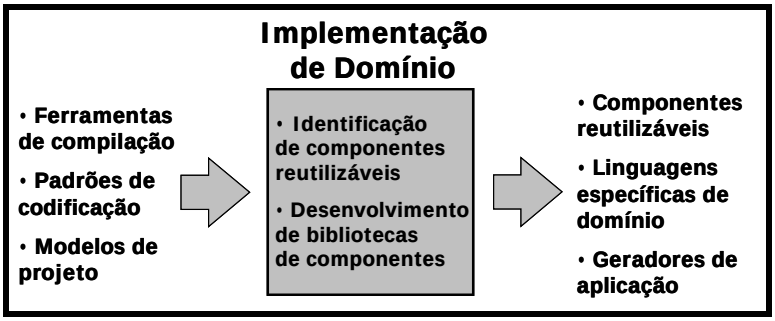


Figura 2.4: Fase de Implementação do processo da Engenharia de Domínio

2.5.2 Engenharia de Aplicações

A Engenharia de Aplicações é o processo relacionado com a Engenharia de Domínio, sendo a ela complementar, que visa a construção aplicações concretas de software a partir do reuso de modelos e componentes previamente produzidos, (WITHEY, 1994) (CMU-SEI, 2004).

Como um processo tradicional da Engenharia de Software, ele se inicia com o levantamento, a análise e a especificação de requisitos. No entanto, tais requisitos são identificados de acordo com o modelo genérico de requisitos da família de sistemas obtido na fase de análise da Engenharia de Domínio.

É com base nessa especificação que se deriva o novo sistema, manual ou automaticamente a partir dos artefatos reusáveis (CZARNECKI, EISENECKER, 2000).

Enquanto a Engenharia de Aplicações (*com reuso*) foca o desenvolvimento de sistemas individuais, a Engenharia de Domínio (*para reuso*) objetiva uma família de aplicações dentro de um domínio.Em essência, as atividades que compõem a Engenharia de Aplicações envolvem o uso dos seguintes insumos:

- um modelo de domínio para levantar os requisitos do usuário;
- um modelo de projeto (arquitetura genérica) para especificar a configuração do produto;
- uma estratégia de particionamento e um modelo de coordenação (padrão arquitetural) para guiar o desenvolvimento específico;
- geradores de aplicações e componentes de software para produzir o código da aplicação.

2.5.2.1 Análise de Aplicações

Na Análise de Aplicações (Figura 2.5), os requisitos para as aplicações em desenvolvimento são explicitados a partir dos modelos de domínio e das necessidades dos usuários. Tal informação é usada para gerar uma lista de características que o sistema pretendido deve apresentar (WITHEY, 1994) (CMU-SEI, 2004).

Assim, sendo possível, as funcionalidades requeridas pelos usuários são enquadradas em projetos e componentes pré-existentes através dos modelos de

domínio. Aquelas necessidades não cobertas pelos modelos existentes são então consideradas com novos requisitos.

Uma vez feito isso, são analisadas as interações entre essas características para se avaliar a sua viabilidade e se identificar requisitos adicionais e contextos não descritos nos modelos de domínio, que pode ser atualizado de acordo com as novas descobertas.

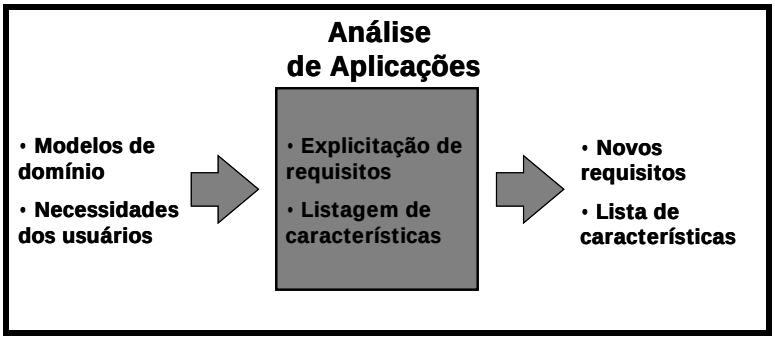


Figura 2.5: Fase de Análise do processo da Engenharia de Aplicações

2.5.2.2 Projeto de Aplicações

No Projeto de Aplicações (Figura 2.6), o próximo passo é comparar a lista de características e os novos requisitos com a especificação de projeto, que é especializada para a aplicações em questão. Então são identificadas as variações em relação ao projeto genérico, decorrentes dos novos requisitos, com a especificação do produto, além de serem também verificados os impactos dessas mudanças sobre os requisitos não funcionais (WITHEY, 1994) (CMU-SEI, 2004).

Em relação aos novos requisitos não satisfeitos pelos componentes já existentes, são ponderadas as vantagens e desvantagens de se construir novos ou modificar os prontos. Estando tudo isso acertado e documentado, resultam as especificações de novos componentes, a configuração do produto ou uma revisão do projeto.

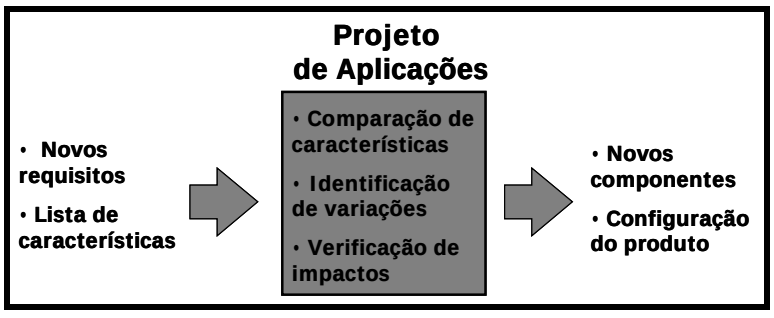


Figura 2.6: Fase de Projeto do processo da Engenharia de Aplicações

2.5.2.3 Implementação de Aplicações

Na Implementação de Aplicações (Figura 2.7), de posse da configuração do produto, que corresponde a uma instância do projeto para o um sistema em particular, são descobertos os componentes – ou o gerador de aplicações, se fosse o caso – que serão utilizados na integração final do sistema (WITHEY, 1994) (CMU-SEI, 2004).

Apesar do esforço em ter na aplicação uma solução para todo o escopo do problema, dificilmente isso é conseguido em virtude da complexidade dos sistemas atuais e das limitações dos modelos. Tais entraves fazem com que o projeto tenha que ser adaptado e que algum código personalizado seja escrito.

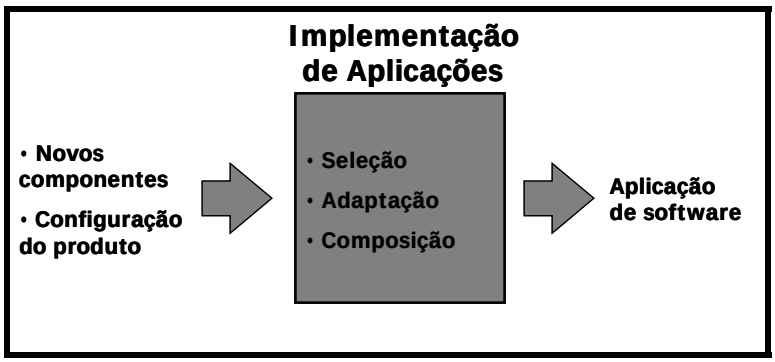


Figura 2.7: Fase de Implementação do processo da Engenharia de Aplicações

O Desenvolvimento de Software Baseado em Componentes tenta introduzir no campo do software aquilo que acontece com o hardware, ou seja, a possibilidade de se construir algo maior reutilizando componentes anteriormente desenvolvidos (SPARLING, 2000) (GILL, 2003).

Tais componentes, para que esse processo seja possível, devem apresentar funcionalidades comumente requeridas e ser intercambiáveis (WANG, 2000). Esse desenvolvimento envolve três epatas: seleção, adaptação e composição, a seguir detalhadas (VITHARANA *et al*., 2003) (VITHARANA, 2003).

a) Seleção

Através da atividade de seleção, procura-se determinar, dentre os componentes reutilizáveis, quais satisfazem os requisitos do sistema em construção. Quando há mais de um que sirva, é preciso escolher o que seja mais adequado.

Uma prática fundamental nesta etapa é verificar se os componentes utilizados realmente realizam as funções que prometem, visto que as características influenciam diretamente o aspecto final do sistema.

b) Adaptação

Como os componentes são construídos para variados propósitos, um tanto tendentes ao contexto em que eles tenham sido elaborados, fazem-se necessárias adaptações para que seja maximizada a utilidade deles e minimizados os conflitos entre eles. A forma como a adaptação é feita depende, em considerável medida, das características de construção dos componentes, que podem ser:

- Caixa branca: quando permitem acesso ao código-fonte interno, possibilitando alterações livres;
- Caixa cinza: quando provêem meios apropriados para a sua extensão;
- Caixa preta: quando estando compilados, não podem ser modificados, possuindo uma interface de uso bem definida.

c) Composição

Nessa atividade, os componentes anteriormente selecionados e adaptados devem ser integrados de acordo com uma infra-estrutura cuidadosamente definida, permitindo que, uma vez feita a composição, todos os componentes se comportam de modo sistêmico, atendendo os requisitos especificados.

Sendo o sistema uma reunião de componentes, sua evolução ocorre normalmente por meio da substituição de parte daqueles por outros que se mostrem mais apropriados para a finalidade pretendida. Para tanto, devem eles apresentar a propriedade de serem facilmente retirados e colocados na estrutura do sistema.

2.6 Ontologias e reutilização de software

2.6.1 Conceito de ontologia

O termo ontologia, em oposição a Ontologia, uma disciplina filosófica, pode se referir tanto a um sistema particular de categorias, independente de uma

linguagem específica, e que serve para descrever uma certa percepção do mundo, como a ontologia de Aristóteles, quanto pode denotar um artefato da Engenharia de Software constituído por um vocabulário específico usado para descrever certa realidade, acompanhado de um conjunto de afirmações explícitas dando sentidos pretendidos às palavras do vocabulário (GUARINO, 1998).

Tais afirmações normalmente tomam a forma da teoria da lógica de primeira ordem, ou lógica de predicados, na qual as palavras aparecem como predicados unários ou binários, conhecidos, respectivamente, como conceitos e relações. No modo mais simples, uma ontologia descreve uma hierarquia de conceitos relacionados entre si. Em casos mais complexos, acrescentam-se axiomas adequados, que são sentenças compostas por predicados unidos através de conectores lógicos, e que servem para expressar restrições a sua interpretação (GUARINO, 1998).

Uma ontologia (CHANDRASEKARAN, 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.

2.6.2 Componentes de uma ontologia

Os componentes básicos presentes em uma ontologia, que servem para representar conhecimento, são os seguintes (CANTELE, 2004):

- conceitos: são idéias básicas que se tentam formalizar, normalmente organizados em taxonomias, podendo ser classes de objetos, métodos, planos, estratégias, processos, entre outros;
- relações: representam relacionamentos semânticos entre os conceitos do domínio, sendo alguns exemplos típicos *é_uma*, *instância_de*, *subclasse_de* e *parte_de*;
- instâncias: representam determinados objetos de um conceito, considerado como um tipo;
- axiomas: são regras declaradas sobre relações que os elementos da ontologia devem cumprir, podendo-se inferir através deles novos conhecimentos.

2.6.3 Vantagens das ontologias

O uso de ontologias torna possível definir uma infra-estrutura para integrar sistemas inteligentes no nível do conhecimento, que se difere do nível de implementação, trazendo as seguintes vantagens:

- colaboração: possibilitam o compartilhamento do conhecimento entre os membros interdisciplinares de uma equipe;
- interoperação: facilitam a integração da informação, especialmente em aplicações distribuídas;
- informação: podem ser usadas como fonte de consulta e de referência do domínio;
- modelagem: as ontologias são representadas por blocos estruturados que podem ser reusáveis na modelagem de sistemas no nível de conhecimento.

2.6.4 Construção de ontologias

Alguns princípios são de essencial observância durante a construção de uma ontologia, tais como (GÓMEZ-PÉREZ, BENJAMINS, 1999):

- os termos devem ser acompanhados de definições objetivas e também de documentação em linguagem natural;
- coerência para permitir realizar inferências que sejam consistentes com as definições;
- flexibilidade para a inclusão de novos termos sem revisão das definições existentes;
- as classes definidas na ontologia devem ser disjuntas, sem superposição de conceitos;
- diversificação das hierarquias para aproveitar ao máximo os mecanismos de herança múltipla;
- minimização da distância semântica entre conceitos similares, de forma a agrupá-los e representá-los utilizando as mesmas primitivas;
- padronização dos nomes sempre que possível.

Uma grande dificuldade encontrada no desenvolvimento de sistemas baseados em conhecimento é ter um repositório compartilhado para a formalização de conhecimento declarativo acerca de uma área de aplicação. Desse modo, fica-se livre de problemas relacionados, principalmente, com o surgimento de inconsistências quando da evolução do domínio, caso a conceitualização fosse descentralizada.

Um interessante recurso de que se dispõe para contornar essas deficiências são as ontologias de domínios. Isso porque elas fornecem um meio de representar formalmente grupos de conceitos e seus relacionamentos em um determinado domínio (MUSEN, 1998).

As ontologias de domínios também podem facilitar a manutenção de bases de conhecimento na medida que apresentam definições de conceitos compartilhadas entre as diversas bases (YAMAGUCHI, KUREMATSU, 1997).

2.6.5 Abstrações de software baseadas em ontologias

As ontologias são particularmente úteis para representar abstrações de software de alto nível, como modelos de domínio e arquiteturas reutilizáveis. Elas fornecem uma terminologia não ambígua 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 (GIRARDI, 2004).

Assim, uma ontologia é a representação de um domínio a partir de seus conceitos abstratos e também a forma como esses conceitos se relacionam entre si. Tal conceitualização é dita formal, explícita e compartilhada. O termo formal significa que pode ser processada por computador; o caráter explícito denota que os conceitos utilizados e as restrições a eles aplicadas são prévia e explicitamente definidos; e, por fim, compartilhamento se refere a um conhecimento consensual, utilizado por mais de um indivíduo e aceito por um grupo.

Uma razão fundamental para a utilização de ontologias no processo de desenvolvimento de sistemas multiagente (COSSENTINO *et al.*, 2002) (DILEO *et al.*, 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. E para que os agentes possam entender o significado dessas expressões, eles precisam acessar uma ontologia comum.

2.7 Considerações finais

Este capítulo apresentou diversos aspectos do reuso de software. Assim, em primeiro lugar, foi discutido o conceito de domínio de aplicação, fundamental para o tema abordado. Em seguida, famílias de sistemas e linhas de produção de software foram definidas, já que constituem uma forma de se promover a reutilização. As abordagens composicional e gerativa para o reuso, bem como os processos da engenharia de domínio e da engenharia de aplicações, também foram detalhadas. Por fim, foram descritas vantagens do emprego de ontologias na reutilização de software.

Os tópicos acima listados são importantes como referencial teórico para o desenvolvimento do restante do trabalho, posto que todo ele tem embasamento no reuso de software, focando especificamente a abordagem composicional da engenharia de aplicações, fazendo uso de ontologias para a representação dos artefatos envolvidos.

O próximo capítulo tratará do paradigma de desenvolvimento baseado em agentes, outro assunto fundamental para a evolução do tema proposto.

3 A ENGENHARIA DE SOFTWARE MULTIAGENTE

3.1 Considerações iniciais

A Engenharia de Software tradicional tem desenvolvido técnicas e ferramentas apropriadas para a construção de sistemas de computação com comportamento predeterminado, que executam exatamente aquilo que foi planejado, projetado e programado na sua construção. Qualquer situação não contemplada nessas instâncias pode provocar falhas no sistema e conseqüentemente perdas tanto no nível econômico como humano (GIRARDI, 2004).

O paradigma baseado em agentes, surgido nos últimos anos, apresenta-se como uma interessante forma de se lidar com as abstrações do mundo real e com a representação do conceito de organização, visto que ele permite um melhor entendimento e uma maior comunicabilidade dos requisitos de usuários, dos artefatos de projeto, da arquitetura do sistema e da implementação final. Além disso, a proximidade entre os conceitos das organizações reais, compostas principalmente por atores e objetivos, e os elementos de projeto e implementação, baseados em agentes e também objetivos, torna essa abordagem bastante adequada para facilitar o intercâmbio de informações entre engenheiros de software e especialistas no domínio.

A Engenharia de Software Multiagente 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).

Assim, o paradigma multiagente é interessante porque permite construir sistemas flexíveis, mas com comportamentos complexos e sofisticados, através da combinação de componentes altamente modulares, que são os agentes, cuja inteligência e habilidade de estabelecer interações sociais resulta em um sistema multiagente com capacidades além da simples reunião das competências isoladas de cada agente (MASSONET *et al.*, 2002).

A sistematização de técnicas e metodologias para a Engenharia de Software Multiagente 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.

Em geral, a maioria delas foca um entendimento maior que o atingido com o desenvolvimento baseado em componentes na abordagem orientada a objetos, bem como a uniformidade, a aceitabilidade e a facilidade de gerenciamento. Para esse fim, as linguagens de modelagem por elas propostas procuram incorporar as noções de agente, papel, objetivo, tarefa, dentre outras. Isso tem sido conseguido ou se estendendo as notações da Orientação a Objetos ou se introduzindo representações especializadas.

Dentre as várias metodologias de desenvolvimento de sistemas multiagente, assim como há consenso quanto a certas fases que compõem tal sistemática, existe também muitas divergências. Assim, nas próximas seções, após se conceituar agentes de software e sistemas multiagente, serão descritas as fases mais comumente aceitas para a Engenharia de Software Multiagente, bem como as metodologias de maior notoriedade.

3.2 Agentes de software

Os agentes são artefatos de software tradicionalmente caracterizados por apresentarem traços bastante peculiares, como os seguintes (WOOLDRIDGE, JENNINGS, 1995):

- autonomia: funcionam sem intervenção humana e executam ações por iniciativa própria segundo o conhecimento que possuem do ambiente e sua racionalidade;
- sociabilidade: implica no estabelecimento de canais de comunicação de uns com os outros, através de sofisticados protocolos de interação, a fim de cooperar, competir, negociar etc;
- percepção: eles percebem o que acontece no ambiente físico ou computacional a sua volta e reagem a isso atuando sobre ele, podendo inclusive provocar-lhe alterações;

- pró-atividade: seus comportamentos são direcionados ao alcance de objetivos, além da mera reatividade a estímulos do ambiente;
- aprendizagem: são capazes de aprender a partir das ações realizadas, considerando sucessos e fracassos, de forma a melhorar seu desempenho.

Mesmo sendo similares a objetos, essas características dos agentes listadas acima levam ao claro estabelecimento das seguintes diferenças entre ambos (PONT, MOREALE, 1996):

- objetos são essencialmente passivos, reagindo a estímulos externos, mas sem apresentar comportamento voltado para objetivos como os agentes possuem;
- agentes tipicamente utilizam uma linguagem comum para a troca de mensagens, diferente das mensagens passadas para objetos que dependem da classe da qual eles são instâncias.

Portanto, um agente é uma entidade autônoma que percebe seu ambiente através de sensores e age sobre o mesmo utilizando-se dos executores. Na sua construção, deve-se decidir como fazer o mapeamento de percepções a ações (GIRARDI, 2004).

A estrutura básica de um agente tem uma forma bem simples, possuindo 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.

3.3 Sistemas multiagente

Um sistema multiagente pode ser caracterizado como um grupo de agentes que atuam em conjunto no sentido de resolver problemas que estão além das suas habilidades individuais, realizando interações entre eles de modo cooperativo para atingir uma meta (GIRARDI, 2004).

Os principais desafios enfrentados no desenvolvimento de tais sistemas estão relacionados com (SYCARA, 1998):

a) a decomposição dos problemas e a alocação de tarefas individuais aos agentes;
b) a coordenação do controle e da comunicação dos agentes;
c) a coerência entre as múltiplas ações dos agentes;
d) o raciocínio de um agente sobre os outros e o estado de coordenação;
e) a conciliação de objetivos conflitantes de diferentes agentes;
f) a construção desses sistemas na prática.

Um sistema multiagente corretamente construído, nos termos do sexto item, deve prover um arcabouço para a resolução dos outros cinco dilemas (DELOACH, 1998).

A crescente complexidade do software é uma preocupação cotidiana dos desenvolvedores, o que acaba implicando na qualidade resultante e nos custos gerados.

Através da decomposição de software baseada em agentes, 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 melhor a complexidade (GIRARDI, 2004).

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.

3.4 Fases, passos e produtos do desenvolvimento orientado a agentes

A maioria das propostas para o desenvolvimento orientado a agentes 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 *et al.*, 1998a) utilizam conceitos de modelagem da engenharia do conhecimento. A seguir, ambas as fases são definidas.

3.4.1 Fase de 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 a eles (GIRARDI, 2004).

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.

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.

Às vezes, os indivíduos têm que se comunicar com outros indivíduos internos ou externos para cooperar na execução de uma atividade.

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, como se observa na Tabela 3.1.

Passos	Descrição	Produtos
Modelagem de objetivos	Definido o problema a ser resolvido, o objetivo geral do sistema é identificado, sendo ele refinado em objetivos específicos.	Modelo de objetivos
Modelagem de papéis	Para cada objetivo específico, é identificada uma responsabilidade que será exercida por um papel, e então são estabelecidas atividades a se realizar.	Modelo de papéis
Modelagem de interações entre papéis	Através de uma análise das suas atividades respectivas, são conhecidas as interações entre os papéis.	Modelo de interações entre papéis
Modelagem de conceitos	Paralelamente às outras atividades, são levantados os conceitos e os relacionamentos presentes nas ontologias que representam o conhecimento do sistema multiagente.	Modelo de conceitos

Tabela 3.1: Atividades e resultados da Análise de Requisitos de sistemas multiagente

3.4.2 Fase de 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.

Essa fase geralmente compreende o projeto arquitetural, onde são definidos os diferentes componentes da arquitetura e a forma em que eles cooperam para a alcançar o objetivo do sistema, e o projeto detalhado, em que se define o comportamento e os atributos de cada um dos componentes da arquitetura.

No projeto orientado a agentes, papéis são atribuídos a agentes e serviços podem ser implementados em termos de tarefas, sendo decompostos em termos de ações diretas sobre a representação interna que o agente possui do ambiente. A comunicação de tais ações se dá através do envio e do recebimento de mensagens de acordo com um protocolo de interação, que detalham as interações entre papéis. O produto dessa fase é uma descrição conceitual do sistema que independe da implementação. A Tabela 3.2 e a Tabela 3.3 resumem cada uma dessas subfases.

Passos	Descrição	Produtos
Modelagem de agentes	Identifica os agentes que compõem a arquitetura do sistema multiagente através da análise dos papéis especificados anteriormente.	Modelo de agentes
Modelagem de interações entre agentes	Estabelece a origem, destino e seqüência das interações entre os agentes que compõem a arquitetura do sistema multiagente.	Modelo de interações entre agentes
Modelagem da arquitetura do sistema multiagente	Especifica uma solução computacional ao problema de acordo com as interações estabelecidas, na forma de mecanismos de cooperação e coordenação entre agentes.	Modelo arquitetural do sistema multiagente

Tabela 3.2: Atividades e resultados do Projeto Arquitetural de sistemas multiagente

Passos	Descrição	Produtos
Modelagem detalhada dos agentes	Especifica a arquitetura de cada agente da sociedade, envolvendo a análise do seu comportamento e de seu mecanismo para a tomada de decisões, além de sua estruturação como deliberativo ou reativo.	Modelo de projeto detalhado dos agentes

Tabela 3.3: Atividades e resultados do Projeto Detalhado de sistemas multiagente

3.5 O Reuso no desenvolvimento de software multiagente

A Engenharia de Domínio Multiagente e a Engenharia de Aplicações Multiagente são processos complementares e interdependentes que possibilitam o reuso no desenvolvimento de software multiagente.

No contexto da Engenharia de Software Multiagente (GIRARDI, 2004), a Engenharia de Domínio Multiagente objetiva desenvolver artefatos de software reutilizáveis orientados a agentes a partir do conhecimento acerca de um determinado domínio, de requisitos comuns e variáveis de uma família de aplicações nesse domínio, bem como de experiências passadas de desenvolvimento.

Tais artefatos são, então, reutilizados na Engenharia de Aplicações Multiagente para a construção de aplicações multiagente a partir da definição de requisitos específicos, que guiam a seleção, adaptação e composição daqueles artefatos, havendo a influência de padrões de software nas diversas fases desses.

Na Figura 3.1, são ilustradas as relações entre a Engenharia de Domínio Multiagente e a Engenharia de Aplicações Multiagente. Como se observa na figura, ambos os processos são apoiados em metodologias de desenvolvimento próprias, respectivamente MADEM e MAAEM. A metodologia MAAEM, objeto principal desse trabalho, será apresentada no próximo capítulo, onde também será brevemente descrita a metodologia MADEM, já que os produtos desta são empregados como insumos por aquela.

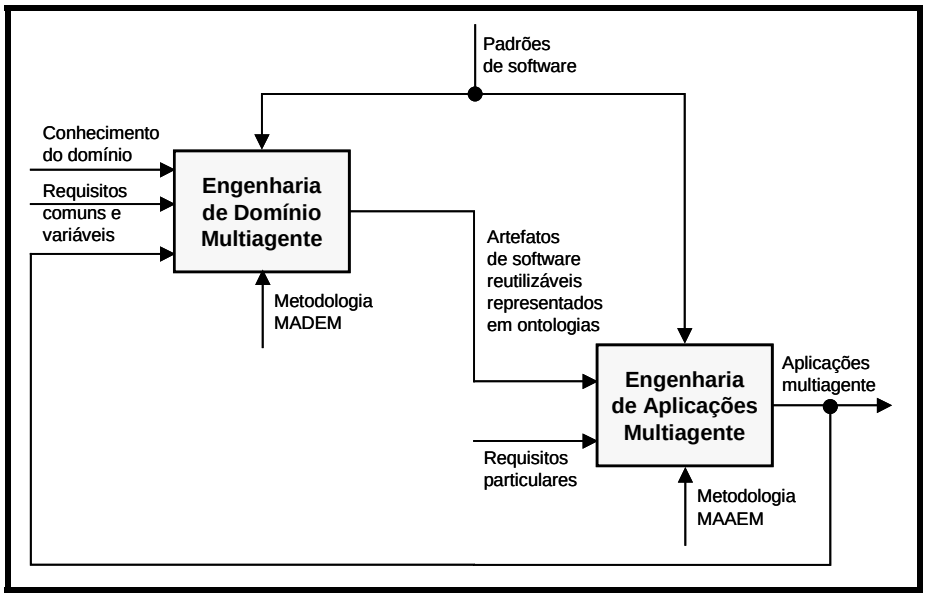


Figura 3.1: Engenharia de Software Multiagente baseada na Reutilização

Nas subseções subsequentes são detalhados em termos de fases os processos da Engenharia de Domínio Multiagente e da Engenharia de Aplicações Multiagente.

3.5.1 O processo da Engenharia de Domínio Multiagente

A Engenharia de Domínio Multiagente, por sua vez, caracteriza uma tentativa de introdução das vantagens decorrentes da reutilização na Engenharia de Software Multiagente, sendo, portanto, encarregada da construção de artefatos baseados em agentes representando o domínio abordado.

A Figura 3.2 mostra as três fases da Engenharia de Domínio Multiagente, que são a Análise, o Projeto e a Implementação de Domínio, com as respectivas técnicas que as suportam: GRAMO, DDEMAS e DIMAS, todas sofrendo influência de padrões extraídos de experiências de desenvolvimento passadas.

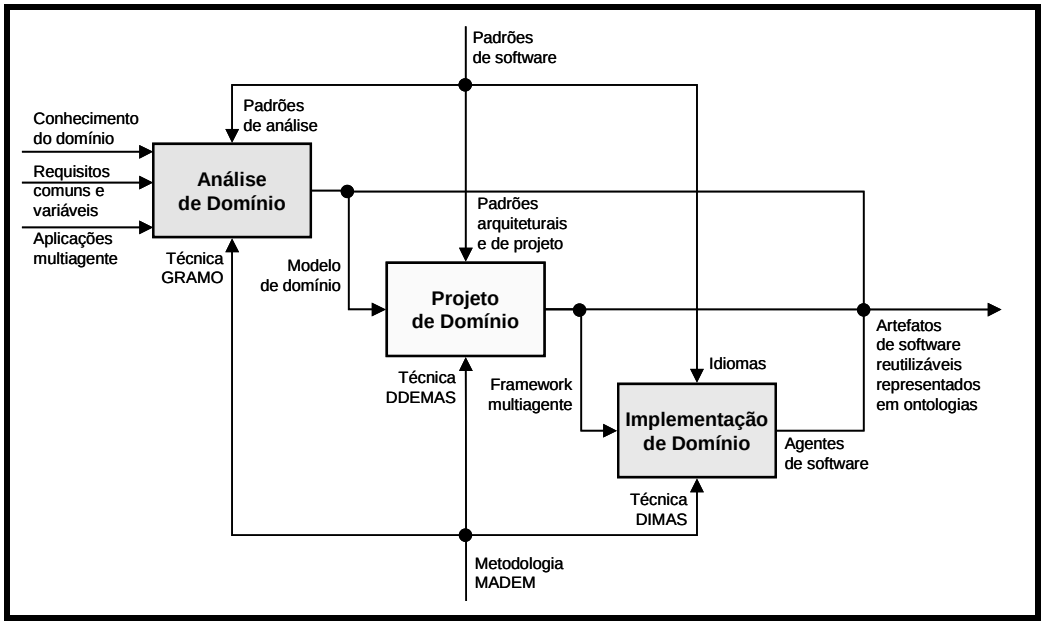


Figura 3.2: O processo da Engenharia de Domínio Multiagente

3.5.2 O processo da Engenharia de Aplicações Multiagente

A Engenharia de Aplicações Multiagente foca o desenvolvimento de sistemas individuais, enquanto a Engenharia de Domínio Multiagente objetiva a construção de uma família de aplicações dentro de um domínio.

A Figura 3.3 ilustra as fases de Análise, Projeto e Implementação da Engenharia de Aplicações Multiagente, acompanhas das técnicas SRAMO,

ADEMAS e AIMAS que as orientam e serão objeto do próximo capítulo. Os padrões extraídos na Engenharia de Aplicações e revisados na Engenharia de Domínio exercem influência neste momento, reiniciando todo o ciclo de descoberta e emprego de padrões.

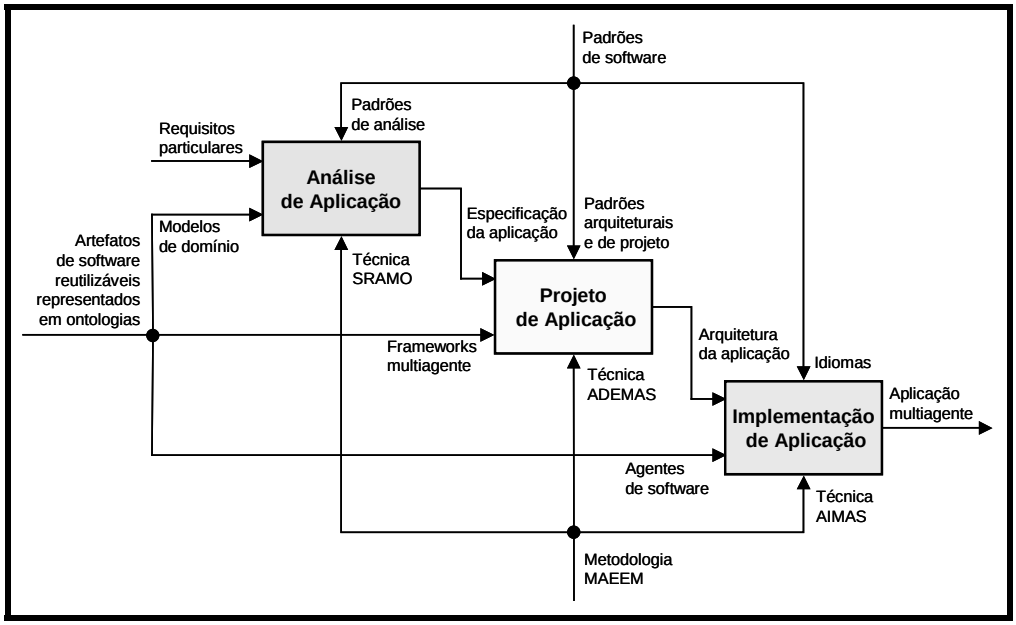


Figura 3.3: O processo da Engenharia de Aplicações Multiagente

3.6 Levantamento de metodologias e ferramentas para a construção de sistemas multiagente

Novas áreas de aplicação, tais como o comércio eletrônico e o compartilhamento de arquivos ponto-a-ponto, requerem sistemas de software cujas arquiteturas sejam abertas e evolutivas e que explorem os recursos disponíveis em seus ambientes. Mecanismos de comunicação, negociação e coordenação entre componentes de software têm se mostrado uma boa solução para estas necessidades, sendo que as tecnologias orientadas a agentes parecem bastante promissoras para a construção daquelas aplicações. Para tanto, metodologias bem descritas em amplitude e profundidade, cobrindo todo o processo de desenvolvimento, são fundamentais (GIUNCHIGLIA *et al.*, 2002).

O advento dos sistemas multiagente trouxe uma nova forma de se abordar a construção de aplicações distribuídas, inteligentes e robustas, que provêem soluções para problemas em ambientes complexos (JENNINGS, 2000), mas que suscitam problemas relativos ao comportamento individual dos agentes (DILEO *et al.*, 2002).

Os meios tradicionais para concepção e desenvolvimento de sistemas não se adequam exatamente a esse paradigma. Muitas tentativas foram feitas na criação de metodologias e ferramentas para esse propósito, mas a maioria delas falha por ser focada em arquiteturas simples com um único agente, não dando o suporte adequado para o tratamento da complexidade do software (IGLESIAS *et al.*, 1998b).

Na abordagem dos agentes, intenta-se fugir da tradição de se elaborar técnicas de desenvolvimento dirigidas pela tecnologia de implementação, como já ocorreu quanto às linguagens de programação estruturadas e orientadas a objetos, para baseá-las nos requisitos, no sentido que os conceitos usados na análise de requisitos são empregados adiante no projeto e na implementação (MYLOPOULOS, CASTRO, 2000).

O projeto de sistemas baseados em agentes se difere do convencional porque o conceito de agente envolve noções como a de autonomia, cooperação etc. Assim, para uma técnica ser apropriada a esse desenvolvimento, ela deve aproveitar as lições adquiridas das abordagens clássicas, mas precisa ir além para abranger também aquelas peculiaridades, bem como questões relativas a ontologias, comunicação, mobilidade, dentre outras (COSSENTINO, POTTS, 2002).

Em face disso, tem-se a necessidade de se dispor de diretrizes próprias para orientar a construção de sistemas no contexto da Engenharia de Software Multiagente. A seguir são descritas a PASSI, a MaSE, a Gaia, a MESSAGE e a Tropos, algumas das principais metodologias – e respectivas ferramentas – disponíveis para o desenvolvimento baseado em agentes.

3.6.1 PASSI

A PASSI (*Process for Agents Societies Specification and Implementation*) é uma metodologia para o projeto e desenvolvimento de sociedades multiagente, abrangendo desde a especificação de requisitos até a codificação, através da integração de modelos e conceitos tanto da Engenharia de Software Orientada a Objetos quanto da Inteligência Artificial por meio da notação UML – Unified Modeling Language (COSSENTINO *et al.*, 2002).

Na PASSI, um agente é uma unidade de software que representa uma entidade autônoma capaz de perseguir seus objetivos através de decisões, ações e

relacionamentos sociais. Isso vale tanto no nível abstrato – quando a modelagem é distante da implementação, sendo orientado ao problema – quanto no concreto – quando os modelos são próximos do código, sendo dirigido à solução.

Assim, um agente é a implementação de software de uma entidade autônoma capaz de perseguir objetivos através de decisões, ações e relações sociais autônomas, podendo se ocupar de vários papéis funcionais durante suas interações com outros agentes com o intuito de alcançar objetivos, sendo um papel uma coleção de tarefas desempenhadas por um agente para atingir um sub-objetivo. Uma tarefa, por sua vez, é uma unidade motivada de comportamento individual ou interativo.

A UML foi adotada como linguagem de modelagem por ser amplamente aceita tanto no meio acadêmico quanto no industrial. Outro fator importante foi a facilidade de se adaptá-la para a representação de projetos orientados a agentes, através de seus mecanismos de extensão, tais como restrições, valores rotulados e estereótipos. Existe também uma variante da UML, denominada AUML – Agent Unified Modeling Language (ODELL *et al.*, 2000), que vai além das extensões padrão, mas ainda assim é fortemente inspirada na orientação a objetos.

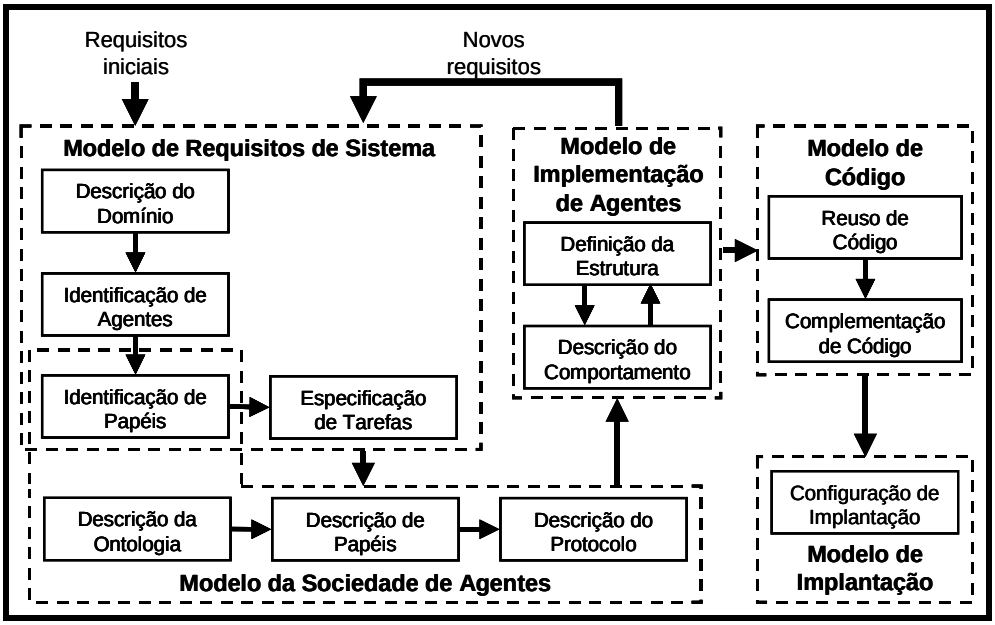


Figura 3.4: Fases e passos da metodologia PASSI

A aplicação da PASSI compreende cinco fases, cada uma resultando em um diferente modelo, num total de doze passos, que parte de uma representação informal de requisitos, seguida pela modelagem da sociedade de agentes e da solução arquitetural, as quais precedem a produção ou reuso do código necessário

para a construção dos agentes, a serem configurados para uso na etapa final de implantação, como se observa na Figura 3.4 (COSSENTINO, POTTS, 2002).

O Modelo de Requisitos de Sistema constitui uma representação em linguagem natural dos requisitos do sistema, em termos de agências e propósitos, envolvendo os quatro seguintes passos:

- Descrição do Domínio: descrição funcional do sistema composta de seqüências hierárquicas de diagramas de casos de uso, correspondentes a funcionalidades do sistema, cujos respectivos detalhamentos de cenários são então explicados em diagramas de sequência;
- Identificação de Agentes: a partir da decomposição funcional do passo anterior, agentes são identificados como pacotes compostos por um ou mais casos de uso, sendo cada um destes uma função a ser desempenhada pelo agente;
- Identificação de Papéis: sob a forma de instâncias de agentes, os papéis são apresentados em diagramas de sequência referentes aos diversos cenários, sendo que em cada um destes um mesmo agente pode desempenhar um ou mais papéis diferentes;
- Especificação de Tarefas: o comportamento de um agente é decomposto na forma de tarefas, desenhando-se um diagrama de atividades para cada agente, no qual há duas raias, a da direita com as tarefas do agente e a da esquerda com os demais agentes.

O Modelo da Sociedade de Agentes representa as interações e dependências sociais entre os vários agentes envolvidos na solução, havendo três passos novos e um repetido da fase precedente:

- Identificação de Papéis: continuação do passo iniciado na fase de modelagem de requisitos do sistema;
- Descrição da Ontologia: através de dois diagramas de classes, é expressa a riqueza semântica requerida por um sistema multiagente, seja através de ontologias explícitas seja por meio de terminologias específicas de domínio. O primeiro descreve a Ontologia de Domínio, cujas entidades envolvidas são representadas como classes. Já o

segundo representa a Ontologia de Comunicação, contendo o conhecimento dos agentes e as relações de comunicação entre eles;

- Descrição de Papéis: modela o ciclo de vida dos agentes de um determinado cenário em um diagrama de classes no qual, para cada agente há um pacote agrupando os vários papéis que ele possa desempenhar. Além disso, são enfocadas as tarefas, colaborações e comunicações envolvidas, com destaque para a introdução das regras organizacionais da sociedade de agentes;
- Descrição do Protocolo: os protocolos são necessários para comunicação entre agentes, devendo ser padronizados, a exemplo do da FIPA, cuja documentação normalmente se apresenta em diagramas de sequência da AUML. Somente são especificados em casos específicos não cobertos pelos padrões.

O Modelo de Implementação de Agentes abrange a solução arquitetural em termos de classes e métodos, compreendendo dois passos mutuamente influentes:

- Definição da Estrutura dos Agentes: composta por diagramas de classes sob duas visões: uma de agentes múltiplos e outra de agentes singulares. Naquela, as diversas classes de agentes aparecem representando a arquitetura geral do sistema, cada qual com suas respectivas tarefas internas. Já nesta, uma determinada classe de agente é mostrada em um diagrama juntamente apenas com as classes de suas tarefas, cada uma com seus respectivos métodos e atributos, restando pouco para uma implementação automática delas;
- Descrição do Comportamento dos Agentes: também apresenta duas visões distintas. Na múltipla, são representados os fluxos de eventos por meio de invocações de métodos e trocas de mensagens, envolvendo tanto as classes de agentes quanto as de tarefas. Para isso, são usados diagramas de atividades, nos quais cada raia comporta uma daquelas classes e cujas atividades correspondem aos métodos destas. Já na singular, o que há é quase somente a implementação dos métodos enunciados no passo precedente, podendo-se utilizar o meio mais adequada para sua descrição, que

pode ser através de diagramas de estados, fluxogramas ou mesmo texto semi-formal.

O Modelo de Código leva a solução para o nível da codificação, envolvendo os dois próximos passos:

- Reuso de Código: sendo possível, algum código previamente escrito é reaproveitado a partir de bibliotecas de diagramas de classes e de atividades, que representam padrões de agentes e tarefas. Àqueles diagramas está associada a correspondente codificação;
- Complementação de Código: é finalizada a programação iniciada com o esqueleto advindo do projeto e com o reuso de padrões.

O Modelo de Implantação realiza a distribuição das partes do sistema dentre as diversas unidades processamento, bem como eventuais migrações, através do passo:

- Configuração de Implantação: busca descrever a alocação dos agentes, através de diagramas de implantação, incluindo restrições de migração e mobilidade.

Conforme se observa, a PASSI se revela como uma metodologia apropriada para o desenvolvimento de sistemas multiagente por dar a devida relevância a fatores estratégicos nesse processo, tais como: o uso de uma linguagem de projeto padronizada e bem conhecida; o suporte de uma ferramenta CASE específica para facilitar o trabalho projetista (a *PTK*, descrita a seguir); e atenção à geração automática de grande parte do código.

3.6.1.1 PTK

A PTK (*PASSI Tool Kit*) é uma ferramenta especialmente concebida para prestar total suporte à metodologia PASSI, sendo composta conceitualmente por duas partes completamente integradas: um plugin para o Rational Rose destinado à construção dos modelos; e uma ferramenta Java, que permite o reuso de padrões tanto na plataforma JADE quanto no FIPA-OS através da representação do código dos agentes em uma meta-linguagem baseada em XML (COSSENTINO, POTTS, 2002).

A opção por uma ferramenta CASE comercial baseada em UML como o Rational Rose se deu pelo fato de ser ela amplamente conhecida e utilizada, o que

ajuda a minorar a dificuldade que os iniciantes no paradigma multiagente enfrentam ao projetar novos sistemas, mesmo sendo experientes, por exemplo, na orientação a objetos.

Assim, acredita-se que haverá ganhos em reuso de código através de bases de dados de padrões de agentes/tarefas, além de se facilitar a produção automática de considerável parte do código remanescente. Especificamente, a ferramenta realiza operações de verificação baseadas na correção de diagramas isolados e na consistência entre passos e modelos relacionados. As suas principais funcionalidades são:

- compilação automática de diagramas;
- suporte automático à execução de operações recorrentes;
- consistência de projeto;
- compilação automática de relatórios e de documentos de projeto;
- acesso a bases de dados de padrões;
- geração de código e engenharia reversa.

3.6.2 MaSE

A MaSE (*Multi-agent Systems Engineering*) é uma metodologia de propósito geral para análise e projeto de sistemas multiagente heterogêneos. Ela utiliza uma série de modelos gráficos para descrever os objetivos do sistema, comportamentos, tipos de agentes e interfaces de comunicação dos agentes, além de prover um meio para especificação da definição detalhada independente da arquitetura do projeto interno de um agente (DELOACH *et al.*, 2001) (WOOD, DELOACH, 2001).

A sua motivação é a falta de metodologias testadas e de ferramentas industriais para a criação de sistema baseados em agentes. O objetivo da MaSE é guiar o desenvolvedor ao longo do processo que se inicia com a especificação de requisitos e se encerra com a implementação do sistema multiagente.

A metodologia MaSE procura focar não a construção de agentes individuais, mas sim o trabalho cooperativo destes em ambientes heterogêneos para a solução de problemas complexos. Assim, em um sistema multiagente

satisfatoriamente funcional o que interessa é o comportamento coordenado de todos os agentes que dele fazem parte.

Para tanto, a MaSE considera os agentes como pertencentes a um nível de abstração superior ao dos objetos, sendo capazes de coordenar suas ações por via da comunicação entre si de modo a atingir objetivos individuais e coletivos da sociedade multiagente, em vez de simplesmente ter métodos passivos e invocáveis, como ocorre na orientação a objetos. Essa visão, entretanto, deixa abertura para a convivência de agentes inteligentes ou não em um mesmo sistema (DELOACH, 1999).

Em virtude da consideração de agentes como “objetos ativos”, devido à semelhança existente para esses efeitos, na MaSE, a modelagem dos sistemas multiagente é feita como se fosse de um sistema orientado a objetos, mas com objetivos a atingir. Assim, a sua abordagem é baseada em tecnologias herdadas da Orientação a Objetos, tais como as técnicas de análise e projeto da OMT e os diagramas da UML, acrescendo-se a isso conceitos como os de objetivo, sensor, atuador etc, além de outras modificações na semântica da notação.

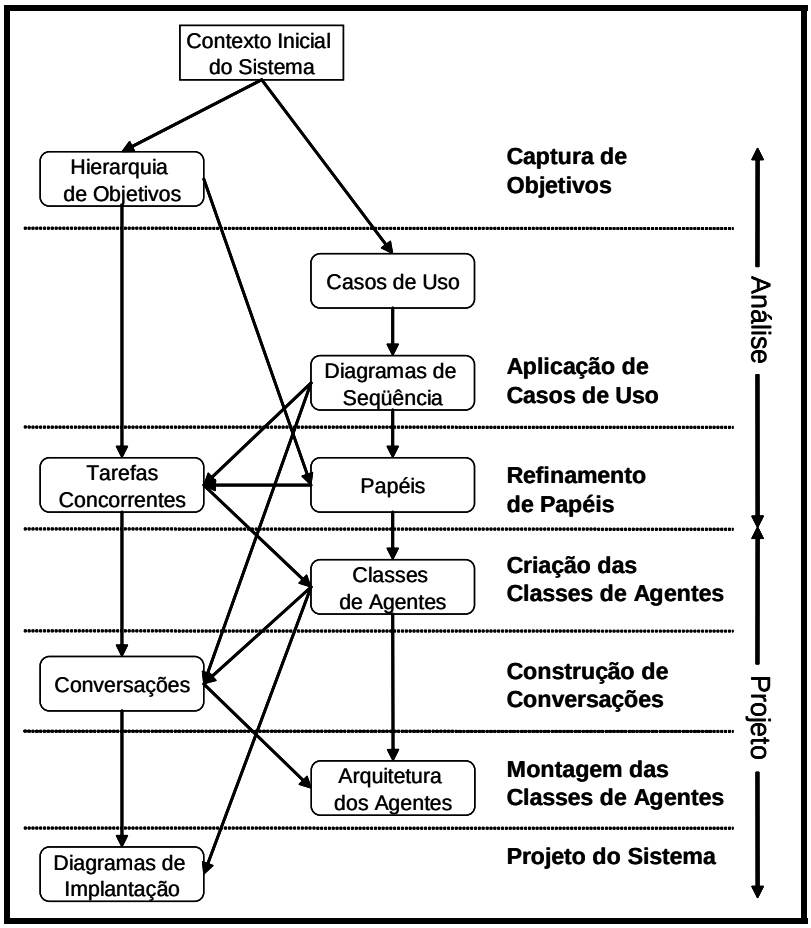


Figura 3.5: Fases e passos da metodologia MaSE

O foco principal da MaSE é auxiliar o projetista a reunir um conjunto inicial de requisitos para ser analisado, projetado e implementado um sistema multiagente funcional, sendo a Análise e o Projeto as duas fases da metodologia, cada uma subdividida em passos, conforme visto na Figura 3.5 (DELOACH *et al.*, 2001).

Os retângulos arredondados denotam os modelos usados para capturar os produtos de cada passo, enquanto as setas entre eles mostram como cada modelo afeta os demais.

Apesar de haver um fluxo único e direto entre passos e fases, na prática, a metodologia MaSE é iterativa, permitindo que o analista ou projetista possa realiza-los livremente com o intuito de obter modelos do sistema completos e consistentes.

A fase de análise da MaSE busca a construção de um conjunto de papéis cujas tarefas descrevem o que o sistema deve fazer para atender os seus requisitos. Cada papel descreve uma entidade que desempenha uma função no sistema e é responsável pelo alcance dos objetivos e sub-objetivos do sistema. Ela compreende três passos: Captura de Objetivos; Aplicação de Casos de Uso e Refinamento de Papéis.

Os objetivos capturados, utilizados para abstrair os requisitos funcionais do sistema, são hierarquizados de acordo com sua importância, havendo um superior e outros tantos subordinados, que uma vez atingidos levam ao alcance daquele. Os papéis, cujo desempenho leva a consecução dos objetivos, sendo empregados na modelagem de sistemas multiagente por serem facilmente mapeáveis a agentes, são refinados com o auxílio de diagramas, previamente elaborados, de casos de uso, por apresentar o segmento lógico de interações entre os papéis, e de seqüência, por determinar o número mínimo de mensagens a serem trocadas no sistema, permitindo uma melhor validação de objetivos e derivação de papéis, sendo que, em geral, um destes representa um ou mais daqueles. Também são definidas tarefas dos papéis, na forma de diagramas de estados, as quais orientam como atingir o objetivo relacionado.

A fase de projeto da MaSE objetiva a construção do modelo final do sistema, composto por classes e instâncias de agentes, bem como por interações entre estes, a partir dos papéis identificados na fase de Análise. Ela envolve quatro passos: Criação das Classes de Agentes; Construção de Conversações; Montagem das Classes de Agentes e Projeto do Sistema.

Inicialmente, é criado um diagrama de classes de agente, sendo a cada uma destas atribuído um papel. Os relacionamentos, diferentemente do diagrama de classes de objetos, não expressam uma estrutura hereditária, mas sim conversações de alto nível, as quais são identificadas com base nas tarefas concorrentes dos papéis desempenhados por cada agente e constituem um protocolo de coordenação na forma de diagramas de estados.

Tais conversações são incrementadas com as mensagens e os estados relacionados com cada ato de comunicação. A arquitetura interna de cada classe de agente é então definida, podendo ser de um dos seguintes tipos: BDI (Crenças – Desejos – Intenções); reativa; deliberativa; baseada em conhecimento; ou definida pelo usuário. Procura-se assegurar que todas as comunicações serão suportadas por um correspondente método. Ao final, instâncias reais das classes de agentes são criadas, resultando em um diagrama de implantação, que estabelece a estrutura definitiva do sistema e, no futuro, levará a cabo a completa geração automática de código.

O maior mérito da MaSE é a possibilidade se acompanhar as mudanças ao longo do processo, sendo possível a sua propagação em qualquer sentido. Por exemplo, de um objetivo capturado se derivam papéis, tarefas e classes de agentes assim como destes se pode obter aqueles facilmente, indo-se, assim, da especificação inicial do sistema multiagente a um conjunto de modelos formais de projeto, sempre adequadamente guiada pela metodologia a passagem de um passo para outro (DELOACH *et al.*, 2001).

Apesar disso, a MaSE não aborda o projeto do domínio de informação do sistema, resultando em uma documentação que não especifica a semântica dos dados passados entre os agentes, o que pode conduzir a sérios problemas, principalmente quando se trata de sistemas legados ou de agentes reutilizados, podendo estes adotar diferentes termos para representar um mesmo conceito (DILEO *et al.*, 2002).

Tal questão compromete o comportamento de sistemas multiagente, posto que as interações entre agentes normalmente se dão na forma de comunicações, as quais nem sempre correspondem apenas a mensagens performativas, mas muitas vezes carecem da passagem de parâmetros, que devem seguir tipos previstos para se adequarem ao fluxo de informação.

Uma solução para tanto advém da incorporação à MaSE de ontologias definindo o domínio de informação do sistema (DILEO *et al.*, 2002). Assim, a partir dos objetivos, requisitos e casos de uso, é construída a ontologia do sistema, cujas classes e atributos são usados pelos agentes para compartilhar informações entre si. Isso torna a metodologia mais completa, já que passa a contemplar, além dos modelos estruturais e comportamentais, também os modelos de dados, permitindo que o sistema codificado realmente atenda aos requisitos especificados.

3.6.2.1 agentTool

A MaSE é a base do agentTool, um sistema de desenvolvimento de aplicações multiagente que, em contrapartida, serve como plataforma de validação da metodologia. Ela é uma ferramenta gráfica e completamente interativa que suporta os sete passos das duas fases da MaSE, assim como realiza a verificação automática das comunicações entre agentes e a geração de código para frameworks multiagente. Tanto a MaSE quanto a agentTool são independentes de qualquer arquitetura particular de agentes, de linguagens de programação e de modelos de comunicação (DELOACH *et al.*, 2001).

3.6.3 Gaia

A Gaia é uma metodologia geral para a análise e o projeto orientados a agentes que suporta tanto o nível micro do desenvolvimento, ou seja, a estrutura dos agentes, quanto o macro, isto é, a sociedade multiagente e a estrutura da organização. A sua motivação é tentar representar com sucesso as características naturais dos agentes, tais como a autonomia e a capacidade de resolver problemas, além das formas como eles realizam interações e criam organizações. Assim, tem-se como resultado um projeto elaborado a partir dos requisitos do sistema e pronto para ser implementado (WOOLDRIDGE *et al.*, 2000).

Na fase de Análise da Gaia, o objetivo é compreender as funcionalidades do sistema, concebido como uma organização composta de um conjunto de papéis que interagem entre si, e representá-las em modelos de papéis e de interações. Ela é composta por dois passos: Modelagem de Papéis e Modelagem de Interações.

Em primeiro lugar, são identificados os papéis que participarão do sistema, os quais possuem quatro atributos, que são: responsabilidades, que podem obrigar o papel a realizar algo de bom ou proibí-lo de fazer algo de ruim para o escopo do sistema; permissões, que representam o que o papel pode executar e, em particular, que dados pode acessar; atividades, que são tarefas que o papel desempenha sem precisar interagir com outros papéis; e protocolos, que são padrões específicos de interação. A Gaia define formalmente operadores e modelos para representar papéis e seus atributos, assim como esquemas para as interações.

Já durante o projeto, são realizados três passos: Modelagem de Agentes; Modelagem de Serviços e Modelagem de Comunicação. No primeiro, ocorre o mapeamento dos papéis em tipos de agentes, para então ser criado o número adequado de instâncias de cada tipo de agente. No segundo, busca-se determinar o modelo de serviços requerido para satisfazer um papel, seja ele desempenhado por um ou vários agentes. No terceiro e último passo, os agentes são tornados familiares entre si através da criação de um modelo de comunicação.

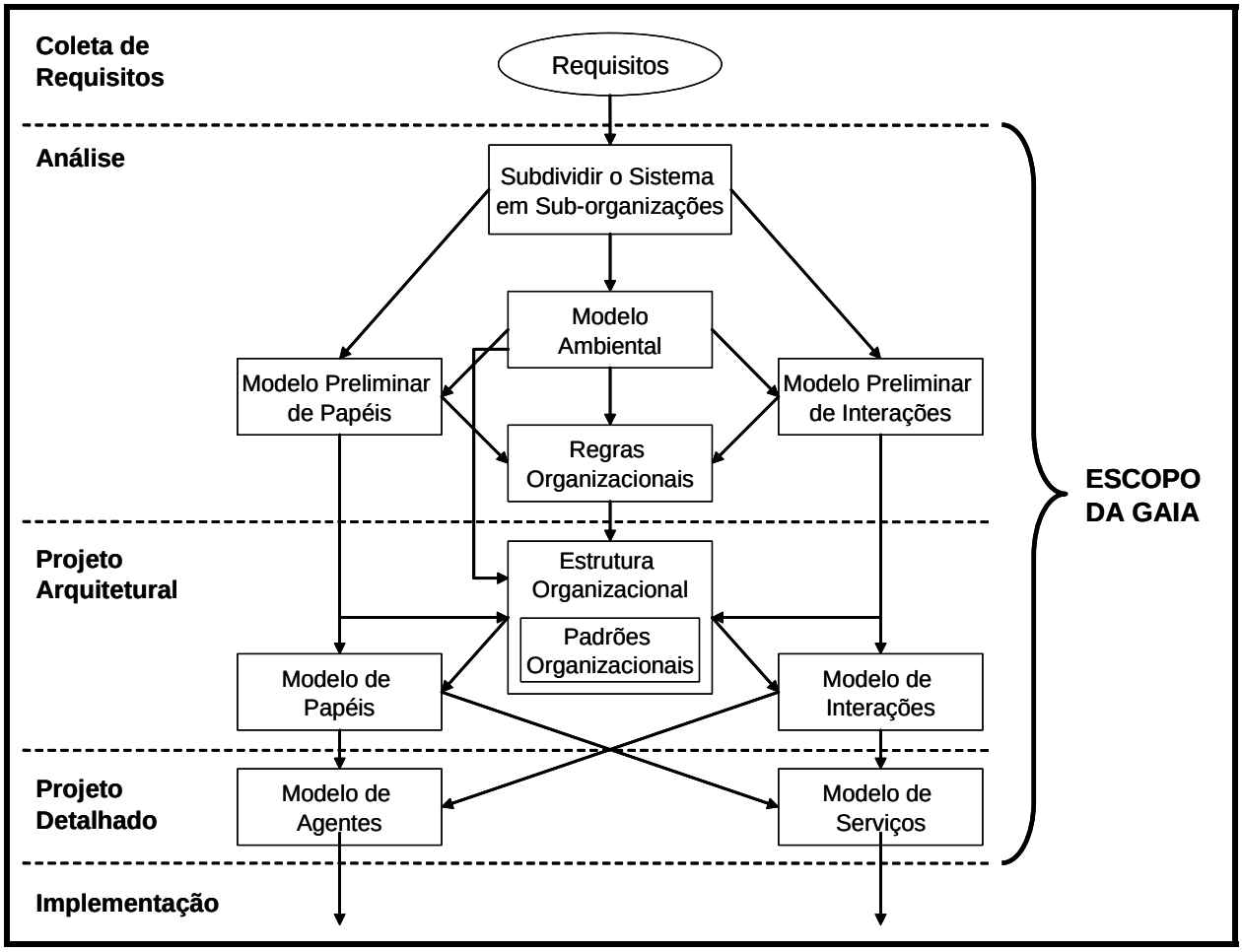


Figura 3.6: Fases e passos da metodologia Gaia

Apesar de se configurar como uma interessante abordagem para o desenvolvimento de sistemas multiagente em domínios fechados, a metodologia Gaia apresenta limitações que reduzem sua adoção em ambientes abertos e imprevisíveis como o domínio das aplicações para Internet. Então, para superar tais restrições, Zambonelli (2003) propôs uma extensão da metodologia tomando por base a especificação original da Gaia (WOOLDRIDGE *et al.*, 2000). A Figura 3.6 ilustra essa versão estendida e as novas abstrações organizacionais que são exploradas para se ampliar de maneira significativa a sua aplicabilidade.

Como se observa, a metodologia Gaia passou a ser mais ainda pautada na noção de organização, considerando as abstrações a ela inerentes como sendo fundamentais para se analisar e projetar sistemas multiagente. Isso permite definir uma seqüência ordenada de passos, compondo as fases da metodologia, os quais produzem um conjunto de modelos, os relacionados entre eles, e regras sobre como e quando utilizá-los para o desenvolvimento de um sistema multiagente (ZAMBONELLI *et al.*, 2003).

Sob essa nova abordagem, a fase de Análise da Gaia tem como foco coletar e organizar a especificação que será a base para o projeto da organização computacional, o que inclui a identificação de:

- objetivos da organização, que constituem o sistema como um todo e seu comportamento global esperado, através da decomposição da organização geral em sub-organizações fracamente acopladas;
- modelo ambiental, pensado com uma representação computacional abstrata do ambiente em que o sistema multiagente irá se situar;
- modelo preliminar de papéis, que reúne as habilidades básicas requeridas pela organização, expressas na forma de papéis que podem ser identificados, mas não completamente definidos, sem comprometimento com determinada estrutura organizacional e independentemente do mapeamento futuro a agentes;
- modelo preliminar de interações, que identifica as interações básicas exigidas pelos papéis preliminares, também sendo incompleto e abstraindo a estrutura da organização;
- regras organizacionais, que devem ser impostas e respeitadas em seu comportamento global para expressar restrições à execução de

atividades dos papéis e protocolos e que são de suma importância para promover a eficiência do projeto e para estabelecer a forma como o sistema multiagente em desenvolvimento poderá suportar um comportamento aberto e independente.

Em seguida, na fase de Projeto, os produtos da análise são explorados em dois momentos distintos: o Projeto Arquitetural e o Projeto Detalhado. A primeira subfase inclui a:

- definição da estrutura organizacional do sistema, em termos de sua topologia e regime de controle, para o quê pode contribuir o uso de catálogos de padrões organizacionais e que deve considerar: a eficiência organizacional; a organização real em que o sistema multiagente se situará; e a necessidade de imposição de regras organizacionais;
- conclusão dos modelos preliminares de papéis e de interações, com base na estrutura organizacional adotada e envolvendo a separação, quando possível, dos aspectos independentes da organização, detectados na fase análise, e dos dependentes, derivados da adoção estrutural feita, o que promove uma perspectiva projeto aberto a mudanças em razão da separação entre a estrutura do sistema e os seus objetivos.

Uma vez definida a estrutura geral do sistema, bem como todos papéis e interações, passa-se para a segunda subfase, o Projeto Detalhado, que envolve a:

- definição do modelo de agentes, que identifica as classes e respectivas instâncias de agentes que irão compor o sistema, podendo haver uma correspondência de um para um entre papéis e tipos de agentes, a não ser quando há vários papéis intimamente relacionados, sendo mapeáveis a uma mesma classe de agente, por razões de conveniência e eficiência;
- definição do modelo de serviços, que identifica os principais serviços e propriedades, considerados como blocos coerentes de atividade em que agentes irão se engajar, os quais são necessários para o desempenho dos papéis dos agentes.

Mesmo com essas melhorias, a Gaia apresenta algumas características peculiares e limitações de escopo. Assim, ela não se vincula particularmente a

nenhuma técnica de modelagem; também não se preocupa com aspectos de implementação, sendo o resultado de seu processo neutro quanto à tecnologia de programação a ser empregada nos agentes, apesar de não se desconsiderar a possível influência desta decisão naquele processo; por fim, a Gaia não lida com as atividades referentes à captura e modelagem de requisitos, entretanto, ela é facilmente integrável às modernas abordagens orientadas a objetivos da Engenharia de Requisitos.

3.6.4 MESSAGE

A MESSAGE (*Methodology for Engineering Systems of Software Agents*) é uma metodologia para a engenharia de software orientada a agentes que cobre a análise e o projeto de sistemas multiagente. Ela estende a UML para incluir conceitos orientados a agentes no nível do conhecimento. Assim, diagramas de classes e de atividades adquirem uma notação que permite visualizar tais conceitos (CAIRE *et al.*, 2001).

Na MESSAGE, vários conceitos são descritos em um alto nível de abstração, ou seja, o nível de conhecimento dos agentes. A Linguagem de Modelagem Unificada (UML) foi adotada na especificação da MESSAGE em razão de:

- ser um padrão de fato para a modelagem orientada a objetos, havendo muitos desenvolvedores treinados em seu uso e existindo diversas ferramentas comerciais lhe dando suporte;
- haver uma proximidade entre o paradigma dos objetos e o dos agentes, podendo os conceitos destes ser prontamente definidos em termos dos daqueles.

Assim, os conceitos da UML são usados para modelar as entidades da MESSAGE em um nível detalhado (ou micro), isto é, de um ponto de vista estrutural essas entidades são objetos com atributos e operações. Já sob o aspecto comportamental, elas são máquinas de estados expressos em termos de ações, eventos e estados. Desse modo, uma visão do mundo é definida como um conjunto de máquinas cujos estados são conhecidos em um certo instante no tempo, bem como a sua evolução é referida pela sequência de eventos que o afetam. Uma

situação é, então, um nível de conhecimento (ou macro) análogo a um estado do mundo.

Os conceitos em nível de conhecimento da MESSAGE são de três categorias: *Entidade Concreta* (*Agente*, *Organização*, *Papel* e *Recurso*); *Atividade* (*Tarefa*, *Interação* e *Protocolo de Interação*) e *Entidade de Estado Mental* (*Objetivo*). Todos esses conceitos são descritos a seguir e ilustrados de maneira relacionada na Figura 3.7.

Um agente pode ser compreendido como uma entidade atômica e autônoma capaz de desempenhar uma série de funções, sendo essas expressas como serviços. Um serviço de um agente equivale, no nível de conhecimento, a uma operação de um objeto. A autonomia significa que as ações do agente não dependem unicamente de eventos externos ou interações, mas também podem seguir motivação própria. Tais motivações são designadas como propósitos do agente, o que irá definir, por exemplo, se ele aceita ou não executar um serviço requisitado e como fará isso.

Uma organização é um conjunto de agentes que trabalham juntos almejando um propósito comum. Mas ela existe apenas virtualmente, já que não há uma entidade computacional que a caracterize, senão seus próprios agentes constituintes, que coletivamente provêem serviços e atingem metas. Sua estrutura é representada, então, por relacionamentos – por exemplo, hierárquicos – entre os seus membros e por mecanismos de coordenação tidos como interações entre os agentes.

Um papel se distingue de um agente assim como a interface de um objeto se difere de sua classe. Desse modo, um papel descreve as características particulares de um agente em um contexto específico. Um agente pode desempenhar vários papéis e vice-versa.

Um recurso representa uma entidade não-autônoma que é usada pelos agentes, tal como um banco de dados ou um programa externo.

Uma tarefa é uma unidade em nível de conhecimento de uma atividade e que possui um único executor principal. Ela tem ainda uma série de pares de pré e pós-condições. Se a tarefa é executa diante de uma pré-condição valida, então é esperado que a pós-condição associada seja obtida quando a tarefa se completar. Tarefas compostas são conjuntos de subtarefas ligados por relações de causalidade.

Uma interação sempre tem dois ou mais participantes e um propósito comum que eles pretendem atingir, sendo esse alvo normalmente o alcance de uma visão consistente de algum aspecto do domínio do problema, os termos do acordo acerca de um serviço ou a troca de resultados de um ou vários serviços.

Um protocolo de interação define o padrão das mensagens que devem ser trocados em razão das interações.

Um objetivo associa um agente a uma situação. Para tanto, considera-se que a arquitetura interna de um agente é formada por: um mecanismo de inferência que não se confunde com os demais elementos; uma base de conhecimento, que contém conhecimentos fixos ou pouco mutáveis sobre um domínio ou sobre a solução de um problema; e uma memória de trabalho, que corresponde a uma base de dados abstrata que guarda o estado do conhecimento do agente.

Assim, se um objetivo está presente nessa memória de trabalho, então o agente pretende alcançar a referida situação. Alguns objetivos são parte da identidade do agente e persistem por toda sua vida. Outros são transitórios e estratégicos, sendo úteis para expressar um propósito em termos de uma função-utilidade que relaciona valores ideais a situações, sendo escolhida aquela que maximiza tal valor. Os objetivos táticos são declarados e eliminados conforme regras definidas na base de conhecimento do agente.

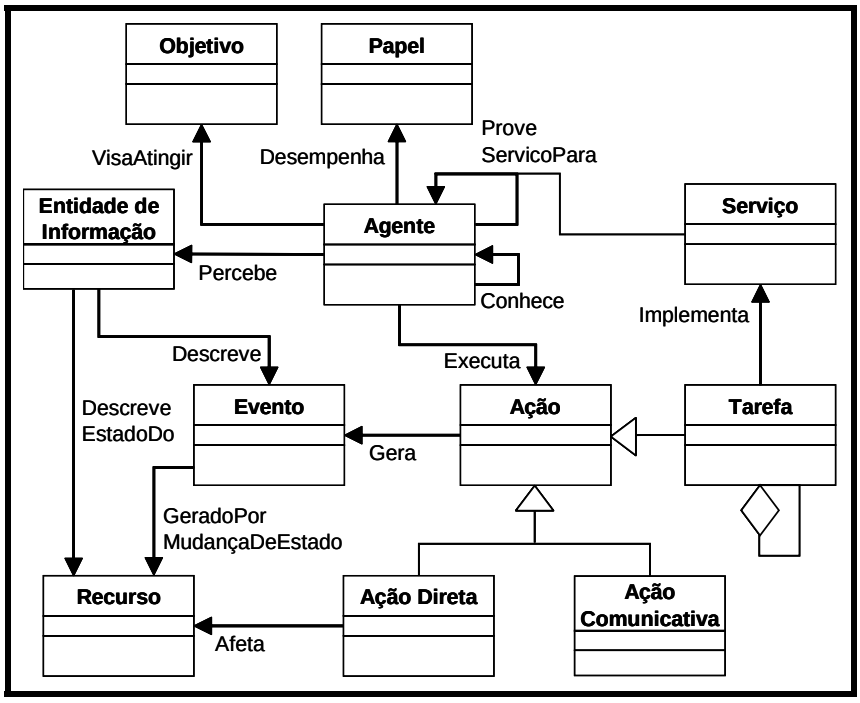


Figura 3.7: Conceitos em nível de conhecimento da metodologia MESSAGE

Para representar os conceitos acima descritos no processo de modelagem, alguns diagramas foram definidos, sendo ele de: organização, objetivo, tarefa, delegação, fluxograma, interação e domínio. Todos eles são extensões do diagrama de classes da UML, com exceção do de tarefa, que estende o diagrama de atividades da UML.

Um modelo de análise segundo a MESSAGE é um conjunto de classes e instâncias inter-relacionadas que são derivadas dos conceitos previamente definidos num metamodelo, havendo uma série de visões sobrepostas que permitem focar diferentes sub-conjuntos dessas entidades e relacionamentos:

- Visão de Organização (OV): mostra entidades concretas, tais como agentes, organizações, papéis e recursos, seu ambiente e os relacionamentos de agregação, poder e familiaridade entre elas;
- Visão de Objetivos/Tarefas (GTV): exibe objetivos, tarefas e situações e as dependências entre eles. Considerando que tanto objetivos quanto tarefas possuem situações como atributos, então podem ser estabelecidos vínculos lógicos entre ambos formando gráficos que ilustram como objetivos podem ser decompostos em sub-objetivos e como estes tarefas levam ao alcance daquelas;
- Visão de Agentes/Papéis (AV): focam os agentes e papéis individuais, que são representados incluindo suas características, como objetivos a alcançar, eventos a perceber, recursos a controlar, tarefas a executar etc;
- Visão de Interação (IV): é mostrado, para cada interação entre agentes/papéis, o iniciador, os colaboradores, o motivador, as informações relevantes fornecidas ou obtidas por cada participante, os eventos que disparam a interação e outros efeitos relevantes;
- Visão de Domínio (DV): exibe os conceitos de domínio e relacionamentos que são relevantes para o sistema em desenvolvimento.

Na fase de análise da MESSAGE, o objetivo é produzir um ou vários modelos do sistema e de seu ambiente, conforme acordado entre o desenvolver e o cliente. Isso permite uma melhor comunicação entre ambos os lados e provê uma

base sólida para a fase de projeto. Tais modelos são resultantes de um refinamento compassado, conforme descrito a seguir.

O *nível 0* e mais alto da decomposição é responsável por definir o sistema levando em conta os clientes e o ambiente. O sistema é visto como um conjunto de organizações que interagem com recursos, atores e outras organizações. Atores podem ser humanos ou agentes. Estágios posteriores de refinamento resultam em modelos nos *níveis 1*, *2*, etc.

No nível 0, o processo de modelagem se inicia com a construção das visões de organização e de objetivos/tarefas, que servem de base para as visões de agente/papéis e de domínio. Por fim, a visão de interações é construída a partir das anteriores. O modelo do nível 0 fornece um apanhado geral do sistema, do seu ambiente e da sua funcionalidade global, sendo que sua granularidade permite focar entidades e relacionamentos conforme o metamodelo. Detalhes sobre cada um desses elementos são adicionados nos níveis mais baixos.

No nível 1, detalhes de entidades tais como organizações, agentes, tarefas, objetivos e outras são definidos. Aspectos relacionados a requisitos funcionais e não-funcionais, como desempenho, distribuição, tolerância a falhas, segurança etc também são definidos.

Diversas estratégias podem ser adotadas para refinar os modelos do nível 0. Por exemplo, as abordagens centradas na organização analisam propriedades do sistema como estrutura, serviços oferecidos, tarefas e objetivos globais, principais papéis etc, sendo que os agentes responsáveis por cada coisa aparecem naturalmente. Já as focadas nos agentes buscam a identificação dos agentes necessários para prover as funcionalidades do sistema. A existência de diferentes visões deixa o projetista livre para escolher a estratégia que lhe pareça mais apropriada.

A principal contribuição da MESSAGE é a adição à UML dos conceitos em nível de conhecimento do agente e dos diagramas que permitem visualizar tais conceitos, mas ressalvando que uma completa modelagem com essa metodologia envolve também a notação convencional da UML.

3.6.5 Tropos

A Tropos é uma metodologia de desenvolvimento de software orientado a agentes que se baseia em duas características-chave: as noções de agente, objetivo, plano, bem como vários outros conceitos em nível de conhecimento. Tais são primitivas fundamentais e uniformemente usadas durante todo o processo; um papel crucial é atribuído à análise e especificação de requisitos quando o futuro sistema é analisado em relação ao seu ambiente alvo (GIUNCHIGLIA *et al.*, 2002).

O processo de analise da Tropos permite que o projeto flua dos atores externos aos do sistema por meio de uma análise e delegação de objetivos. Ainda, provê uma sintaxe abstrata para a construção de seus diagramas e de outras estruturas lingüísticas.

A metodologia se inicia com um modelo do ambiente em que o sistema irá operar, que é descrito em termos de atores, de seus objetivos e suas interdependências. Através de refinamentos incrementais, o modelo é estendido para incluir o próprio sistema e seus subsistemas, também representados como atores aos quais objetivos a alcançar são delegados, planos para executar e recursos a fornecer.

A linguagem da Tropos é baseada em um pequeno conjunto de conceitos e provê ferramentas e técnicas para construir os modelos que representam atores (agentes, posições e papéis), seus objetivos e suas interdependências. Tais modelos são usados para capturar as intenções dos interessados (usuários, proprietários, gerentes etc), as responsabilidades do novo sistema com relação a esses interessados, bem como sua arquitetura e os detalhes de seu projeto. Esses modelos oferecem uma interface comum para varias fases do desenvolvimento, desde o levantamento de requisitos até a implementação, podendo também ser usados como parte da documentação durante a operação e manutenção do software.

As cinco principais fases do desenvolvimento de sistemas multiagente nos termos da metodologia Tropos são descritas a seguir (BRESCIANI *et al.*, 2004):

- Requisitos Iniciais: é por onde se começa a reunir o conjunto de requisitos funcionais e não-funcionais do sistema em desenvolvimento. Os principais interessados do sistema são identificados, sendo

representados como atores, bem como se estabelecem seus respectivos objetivos;

- Requisitos Finais: o sistema propriamente dito é introduzido também como um ator, sendo devidamente relacionado com os atores que representam os interessados em termos de dependências, que indicam as obrigações do sistema para com o seu ambiente e, em contrapartida, o que aquele pode esperar deste;
- Projeto Arquitetural: alguns outros atores do sistema são acrescentados e a eles são atribuídos sub-objetivos e subtarefas derivados dos objetivos e das tarefas designadas para o sistema;
- Projeto Detalhado: atores do sistema são definidos em mais detalhes, sendo especificadas as comunicações e protocolos de coordenação estabelecidos entre eles;
- Implementação: a especificação produzida nas fases precedentes é, então, transformada em um esqueleto para a implementação, o que é feito através do mapeamento dos elementos da metodologia aos conceitos de uma plataforma de programação de agentes escolhida.

A Tropos confere fundamental importância aos requisitos porque é durante seu levantamento que as principais considerações técnicas têm que ser confrontadas com outras sociais e pessoais, sendo nessa fase em que são cometidos os piores e mais custosos erros no desenvolvimento de software. Pois é nela que é respondida a recorrente pergunta: “o que se pretende que o sistema faça?” (MYLOPOULOS *et al.*, 2000).

Além disso, a Tropos inclui um arcabouço de modelagem que contempla visões de software sob quatro perspectivas complementares: social, relativa a atores e suas obrigações e capacidades; intencional, referente a objetivos, seus inter-relacionamentos e responsáveis; orientação a processos, que foca os processos computacionais e de negócio relevantes; e orientação a objetos, que define as classes e objetos principais.

Adicionalmente, a metodologia propõe três níveis de especificação de software: o primeiro é estritamente diagramático, que é importante para a comunicação humana, mas é impreciso e oferece pouco suporte para a análise; o segundo envolve anotações formais complementares, que podem estabelecer precedências entre obrigações, baseando-se em formulários de análise; e o terceiro

constitui-se de uma linguagem de especificação formal, com todos os elementos da metodologia embutidos, suportando formulários complexos de análise.

3.7 Estudo comparativo das metodologias e ferramentas levantadas

Na seção anterior, foram abordadas algumas das principais metodologias atualmente disponíveis para o desenvolvimento de sistemas multiagente: a PASSI, a MaSE, a Gaia, a MESSAGE e a Tropos. Também foram apresentadas as respectivas ferramentas proprietárias que as apóiam, quando existentes.

Cada uma das metodologias possui vantagens e desvantagens, carecendo-se de uma análise cuidadosa para se poder avaliar e escolher a metodologia mais apropriada para um projeto orientado a agentes específico.

Tanto aquele levantamento quanto este estudo comparativo são de fundamental importância para a propositura da metodologia alvo deste trabalho, a ser descrita no próximo capítulo.

Assim, nas próximas seções, são descritos, respectivamente, os critérios – definidos a partir do trabalho de STURM e SHEHORY (2003), com considerações baseadas também no estudo de DAM e WINIKOFF (2003) – adotados para o presente comparativo e as conclusões resultantes desta comparação.

3.7.1 Critérios de comparação

Para empreender qualquer comparação, o ponto de partida é a definição dos critérios sob os quais irá ela se fundar. A despeito disso, muitas dificuldades se impõem ao se tentar comparar diferentes metodologias, posto que normalmente elas se diferem pelos aspectos que enfocam, pela terminologia que adotam, pelas abordagens que as influenciam, pela abragência a que se propõem etc (STURM, SHEHORY, 2003).

No entanto, a utilidade desses comparativos é justamente eliminar uma série de problemas atualmente encontrados no paradigma orientado a agentes e decorrentes da vastidão de metodologias existentes, porém, muitas vezes, imaturas e imcompletas. Isso leva a incertezas ao se querer, por exemplo, selecionar uma metodologia para um caso particular, além de impossibilitar o estabelecimento de padrões razoavelmente reconhecidos e aceitos e de impedir a concentração de

suficientes esforços e recursos de pesquisa no sentido de se obter algumas dessas metodologias realmente completas e maduras (DAM, WINIKOFF, 2003).

No caso das metodologias baseadas em agentes – entendidas como conjuntos de diretrizes para todo o ciclo de vida da construção de sistemas, sob os pontos de vista técnico e gerencial – é necessário que tais critérios sejam relacionados a suas características essenciais, como: conceitos e propriedades, notações e técnicas de modelagem, processos e pragmáticas.

Assim, a seguir, cada um desses critérios é brevemente explicado, para ser então aplicado às diversas metodologias e ferramentas.

3.7.1.1 Conceitos e propriedades

Um conceito é uma abstração ou uma noção inferida ou derivada de instâncias específicas em um domínio de problema. Já uma propriedade é uma capacidade especial ou uma característica. Quanto a isso, a questão aqui gira em torno de se avaliar em que medida uma determinada metodologia emprega os conceitos e propriedades consideradas básicas para o desenvolvimento orientado a agentes, tal como adiante definidos (STURM, SHEHORY, 2003):

- Conceitos:
  - Agente: uma entidade que pode desempenhar tarefas, sendo capaz de saber como executá-las, e realmente faz isso de maneira autônoma;
  - Crença: um fato acerca do mundo considerado verdadeiro;
  - Desejo: o fato de certo valor ser falso, mas se preferindo que fosse verdadeiro, sendo especializado como um objetivo;
  - Intenção: um fato representando a forma de realização de um desejo, sendo também chamado de plano;
  - Mensagem: um meio de intercâmbio de fatos ou de objetos entre entidades;
  - Norma: uma diretriz que caracteriza uma sociedade, exigindo-se dos membros sua observância, e às vezes denominada de regra;
  - Organização: um grupo de agentes trabalhando juntos para atingir o propósito comum, consistindo em papéis que caracterizam os agentes, os quais são seus membros;

- Protocolo: um conjunto ordenado de mensagens que define os padrões admissíveis de um tipo particular de interação entre entidades;
- Papel: uma representação abstrata de uma função ou de um serviço de um agente, ou de sua identificação em um grupo;
- Serviço: uma interface fornecida por um agente para o mundo externo, sendo um conjunto de tarefas que oferece alguma operação funcional, e podendo consistir de outros serviços;
- Sociedade: uma coleção de agentes e de organizações que colaboram para promover seus objetivos individuais;
- Tarefa: uma porção de trabalho que pode ser atribuído a um agente ou por ele desempenhado, podendo ainda ser uma função e ter restrições temporais, e sendo às vezes referida como uma ação;

- Propriedades:
  - Autonomia: habilidade de um agente de operar sem supervisão;
  - Reatividade: habilidade de um agente de responder em tempo às mudanças no ambiente;
  - Proatividade: habilidade de um agente de perseguir novas metas;
  - Sociabilidade: habilidade de um agente de interagir com outros agentes através do envio e do recebimento de mensagens roteáveis e compreensíveis.

3.7.1.2 Linguagem de modelagem

Uma linguagem de modelagem é o meio utilizado para representar os conceitos nos quais se funda uma metodologia, tipicamente consistindo em conjunto de símbolos, bem como em uma sintaxe e em uma semântica incidente sobre tal notação. Além disso, há os modelos, que servem para expressar os diferentes aspectos e níveis de abstração do sistema, oferecendo diversas visões dele, tais como comportamental, estrutural e funcional (DAM, WINIKOFF, 2003). Assim, considerando que uma metodologia normalmente pode envolver complexas tarefas em suas diversas fases, então as suas notações e suas técnicas de modelagem

devem facilitá-las, apresentando características que a tornem (STURM, SHEHORY, 2003):

- Acessível: enuncia a facilidade ou simplicidade em se compreender e empregar a metodologia, tanto pelos usuários novatos quanto pelos experientes;
- Analisável: possibilita a verificação da consistência interna e das implicações dos modelos ou a identificação de aspectos que pareçam obscuros, tais como as interrelações entre operações aparentemente não relacionadas, sendo usualmente viabilizada por ferramentas automatizadas;
- Gerenciável: refere-se à habilidade de lidar com vários níveis de abstração, os quais às vezes são mais amplos, como a especificação de requisitos do sistema, e outras são mais detalhados, como os atributos dos agentes;
- Executável: relaciona-se com a capacidade que se tem de executar simulações e testes ou de gerar protótipos a partir de alguns aspectos da especificação, demonstrando certos comportamentos do sistema e se os requisitos pretendidos estão sendo contemplados;
- Expressiva: abrange a possibilidade de se aplicar a metodologia a múltiplos domínios e também de se apresentar conceitos relativos a:
  - a estrutura do sistema,
  - o conhecimento encapsulado no sistema,
  - a ontologia do sistema,
  - o fluxo de dados no sistema,
  - o fluxo de controle no sistema,
  - as atividades concorrentes no sistema e nos agentes,
  - as restrições de recursos do sistema,
  - a arquitetura física do sistema,
  - a mobilidade dos agentes,
  - a interação do sistema com entidades externas,
  - as definições da interface com os usuários;

- Extensível: permite que um sistema seja especificado de maneira iterativa e incremental, de modo que novos requisitos adicionados não afetem as especificações atuais;
- Precisa: evita que os usuários cometam erros de interpretação dos modelos existentes, que não devem conter ambigüidades.

3.7.1.3 Processo

Um processo de desenvolvimento corresponde a um conjunto de atividades e passos desempenhados como parte do ciclo de vida do software (DAM, WINIKOFF, 2003). Quer dizer, é uma série de ações, mudanças e funções, as quais, quando executadas, resultam em um sistema computadorizado acabado, envolvendo os tópicos que seguem (STURM, SHEHORY, 2003):

- Contexto de desenvolvimento: especifica se a metodologia é destinada à engenharia de um novo software, à reengenharia ou engenharia reversa de um software existente, à prototipação, ou ao projeto para ou com o reuso de componentes;
- Abrangência do ciclo de vida: envolve a apuração de que partes do desenvolvimento de software são tratadas pela metodologia, sendo previstos os seguintes:
  - Coleta de requisitos: oportuniza a especificação, normalmente em texto livre, das necessidades dos usuários que devem ser contempladas pelo sistema;
  - Análise: possibilita a descrição das características observáveis do sistema, tais como funcionalidade, desempenho, capacidade etc;
  - Projeto: indica como o sistema cumprirá seus requisitos, através do refinamento ou da transformação dos modelos da análise nos do projeto, mostrando as naturezas lógica e física do software;
  - Implementação: lida com a conversão dos modelos projetados em software executável, através da codificação manual das unidades do programa, da sua geração automática, ou da montagem de componentes reusáveis já construídos e testados, guardados em bibliotecas;

- Teste: serve para assegurar que cada elemento resultante de cada estágio do desenvolvimento está em conformidade com os requisitos especificados.

Além disso, para uma metodologia ser considerada razoavelmente adequada e, portanto, bem aceita pela comunidade de desenvolvedores, ela deve estabecer uma série de atividades detalhando seu processo de desenvolvimento, as quais podem ser averiguadas através das indagações a seguir:

- Quais são as atividades em cada estágio da metodologia?
- Quais são os produtos gerados no processo?
- Que verificação o processo provê?
- Que validação o processo provê?
- Que parâmetros de garantia de qualidade são supridas?
- Que diretrizes de gerenciamento de projeto são fornecidas?

3.7.1.4 Pragmática

Pragmática diz respeito à consideração dos aspectos práticos do desenvolvimento e do uso da metodologia, devendo ser avaliada com base nas seguintes questões:

- Recursos disponíveis: são considerados como tais aqueles que podem ser úteis na aplicação da metodologia, a exemplo de livros, grupos de discussão, treinamentos e consultorias, ferramentas de automatização;
- Destreza requerida: é tida como o conhecimento prévio que o desenvolvedor deve possuir para aprender a utilizar a metodologia;
- Adequação da linguagem (paradigma e arquitetura): é relativa às tecnologias de implementação cujo uso a metodologia permite, já que, às vezes, ela própria já é baseada em conceitos de uma arquitetura específica ou de uma linguagem de programação em particular, limitando o desenvolvimento à adoção dessas;
- Aplicabilidade ao domínio: é referente ao possível direcionamento de certa metodologia para um ou alguns domínios determinados;

- Escalabilidade do emprego: é consequência da possibilidade de se empregar uma metodologia em aplicações de portes bastante variados.

3.7.2 Quadros comparativos

Na presente subseção, é apresentada uma série de quadros comparando as metodologias abordadas sob os diversos critérios definidos para este estudo comparativo. Assim, a partir da pesquisa realizada e dos resumos elaborados, procurou-se recolher os elementos necessários para o preenchimento de cada tabela. Apesar disso, algumas omissões podem ser encontradas, sendo indicadas por um traço (—), sejam devidas à certeza do ponto em questão não ser contemplado por determinada metodologia, sejam ocasionadas pela impossibilidade de identificá-lo.

Quanto aos conceitos manipulados no curso do desenvolvimento de um sistema multiagente, a Tabela 3.4 sintetiza o mapeamento deles em cada metodologia, de acordo com o acima estabelecido.

Metodol. / Conceit.	**PASSI**	**MaSE**	**Gaia**	**MESSAGE**	**Tropos**
Agente	Agente	Agente	Tipo de agente	Agente	Ator
Crença	Ontologia de domínio	—	—	Base de conhecimento	—
Desejo	Objetivo	Objetivo	—	Objetivo	Objetivo
Intenção	—	Papel	—	—	—
Mensagem	Protocolo	Mensagem	Protocolo	Interação	Comunicação
Norma	—	—	Regra organizacional	—	—
Organização	Sociedade	Sistema	Sistema	Organização	Sistema
Protocolo	Protocolo	Conversação	Protocolo	Protocolo de interações	Protocolo de coordenação
Papel	Papel	Papel	Papel	Papel	Papel
Serviço	—	—	—	Serviço	—
Sociedade	Sociedade	Sistema	Sistema	Sistema	Sistema
Tarefa	Tarefa	Tarefa	Atividade/Res-ponsabilidade	Tarefa	Tarefa

Tabela 3.4: Quadro comparativo de conceitos manipulados pelas metodologias

Já em relação às propriedades conferidas aos agentes enquanto abstração de software, a Tabela 3.5 resume a forma como as várias metodologias estudadas enfocam cada uma daquelas anteriormente enunciadas.

Propried. / Metodol.	**Autonomia**	**Reatividade**	**Proatividade**	**Sociabilidade**
PASSI	Expressa pela capacidade do agente de perseguir objetivos através de decisões, ações e relações sociais autônomas.	Expressa por meio das tarefas que caracterizam o comportamento dos agentes, sendo algumas meras reações ao ambiente.	Expressa pela possibilidade de um agente se ocupar de vários papéis funcionais durante seu ciclo de vida.	Expressa pela existência de uma sociedade multiagente e de seus respectivos protocolos de interação.
MaSE	Considera os agentes como objetos ativos, ou seja, que não necessitam que seus métodos sejam passivamente invocados.	Expressa pela previsão da arquitetura interna do tipo reativo dentre os possíveis para as classes de agentes.	Expressa através de papéis, os quais designam entidades responsáveis por funções para o alcance de objetivos e sub-objetivos.	Percebida pela capacidade dos agentes de coordenar suas ações por meio da comunicação entre si.
Gaia	Expressa por meio do encapsulamento pelos papéis de suas funcionalidades, tornando-os independentes.	Expressa através da propriedade de sobrevivência das responsabilidades de cada papel, mas precariamente.	Expressa por meio da propriedade de sobrevivência das responsabilidades de cada papel.	Expressa através do uso da estrutura e das regras organizacionais.
MESSAGE	Significa que as ações do agente não dependem apenas de eventos externos ou interações, podendo ter também motivação própria.	—	Expressa por meio do estado mental do agente, que contém objetivos, alguns transitórios que mudam no decorrer do seu ciclo de vida.	Expressa através de organizações, que são conjuntos de agentes trabalhando com um propósito comum, por meio da coordenação.
Tropos	Expressa pelo fato de os atores possuirem objetivos, bem como planos para alcançá-los.	Considera o ambiente de operação do sistema, descrito em termos de atores, para que os agentes possam a ele reagir.	—	Expressa por meio da especificação de comunicações e protocolos de coordenação estabelecidos entre os atores do sistema.

Tabela 3.5: Quadro comparativo de propriedades contempladas pelas metodologias

Por sua vez, referente às linguagens de modelagem empregadas na representação dos conceitos, a Tabela 3.6 e a Tabela 3.7 descrevem o modo como cada uma das características precedentemente discutidas é levada em consideração pelas metodologias pesquisadas.

Caract. / Metodol.	**Acessível**	**Analisável**	**Gerenciável**	**Executável**
PASSI	Com um conhecimento prévio de UML, os modelos podem ser facilmente compreendidos.	Oferece, por meio da ferramenta PTK, checagem de consistência de diagramas e modelos.	Lida tanto com os agentes quanto com a sociedade, indo do nível abstrato ao concreto.	Tem suporte à compilação automática de diagramas e à execução de operações recorrentes.

MaSE	Utiliza técnicas de modelagem da orientação a objetos acrescidas de alguns conceitos, facilitando seu uso por parte daqueles familiarizados com tal paradigma.	Oferece a ferramenta agentTool, que serve como uma plataforma de validação, realizando a verificação automática das comunicações.	O modelo final é composto por diagramas que representam as classes e instâncias dos agentes, bem como a implantação do sistema como um todo.	Além de contemplar todas as fases e os passos do desenvolvimento, a agentTool permite a geração automática de código para frameworks multi-agente.
Gaia	Tem modelos fáceis de entender e de usar; porém, o comportamento do sistema depende de um conjunto de expressões lógicas, dificultando em parte a compreensão.	—	Não há uma apresentação hierárquica ou qualquer outro mecanismo para tratar da complexidade; portanto, a descrição do sistema é um tanto simplória.	—
MESSAGE	Descreve os conceitos em um alto nível de abstração e adota a UML como linguagem de modelagem o que facilita sua compreensão.	—	A construção dos modelos se dá através de sucessivos refinamentos, abrangendo, portanto, vários níveis de abstração.	—
Tropos	Provê uma sintaxe abstrata para a construção de diagramas, cuja linguagem é baseada num pequeno conjunto de conceitos, sendo providas técnicas e ferramentas para facilitar seu uso.	—	Possui um arcabouço de modelagem que contempla o software sob quatro perspectivas: social, intencional, orientação a processos e a objetos.	—

Tabela 3.6: Primeira parte do quadro comparativo de características das linguagens de modelagem adotadas pelas metodologias

Caract. / Metodol.	**Expressiva**	**Extensível**	**Precisa**
PASSI	Permite lidar com diferentes tipos de sistemas em vários domínios, tendo modelos para expressar os diversos conceitos relativos a tais sistemas.	A existência de elementos de construção como papéis, agentes e protocolos a torna extensível. Porém, as dependências sociais entre os agentes restringem essa extensibilidade.	A adoção da UML como linguagem de modelagem permite que os modelos sejam representados de maneira formal, evitando interpretações divergentes sobre o mesmo modelo.
MaSE	Permite descrever tanto aspectos do sistema, como objetivos, tipos de agentes e comportamentos, quanto aspectos do projeto interno de cada agente através de uma série de modelos gráficos.	Apesar de haver um fluxo único e direto entre passos e fases, na prática ela é iterativa, permitindo que o analista ou projetista possa realizá-los livremente.	Não possui uma linguagem específica para a modelagem. Utiliza modelos gráficos para representar os conceitos, o que pode dar margem a ambigüidades .

Gaia	Permite lidar com variados sistemas, mas se adequa melhor aos pequenos e médios, devido aos limites à modelagem de grandes quantidades de detalhes.	É em grande parte modular devido à existência de certos elementos de construção, mas ainda assim enfrenta algumas restrições.	Possui determinados elementos de modelagem que evitam interpretações errôneas quanto à maior parte de seus modelos.
MESSAGE	Permite expressar as entidades sob o ponto de vista estrutural como objetos com atributos e operações e sob o comportamental, como máquinas de estados em termos de ações, eventos e estados.	A divisão do sistema em organizações, sendo estas compostas por vários agentes possibilita a extensibilidade.	Estende a UML para incluir conceitos orientados a agentes e assim seus diagramas de classes e de atividades adquirem uma notação que permite visualizá-los sem enganos.
Tropos	Permite que o projeto flua dos atores externos aos do sistema por meio de uma análise e delegação de objetivos, possibilitando expressar a interação do sistema com as entidades externas.	Considera o próprio sistema como um ator, relacionado com os demais, representano os interessados em termos de dependências, sendo em certa medida extensível.	Possui um arcabouço de modelagem que contempla visões de software sob quatro perspectivas complementares. Desse modo é possível obter uma descrição mais formal do sistema.

Tabela 3.7: Segunda parte do quadro comparativo de características das linguagens de modelagem adotadas pelas metodologias

No tocante ao processo de desenvolvimento orientado a agentes, a Tabela 3.8 exibe a maneira como os principais aspectos acima analisados são abordados pelas metodologias levantadas.

Aspect. / Metodol.	**Contexto de desenvolvimento**	**Abrangência do ciclo de vida**	**Atividades por estágio**	**Produtos resultantes**
PASSI	Adequa-se à criação de software novo, reengenharia, reuso de componentes e prototipação através da fase de implementação.	Abrange a coleta de requisitos, análise, projeto e implementação, permitindo o teste através da PTK.	Cada fase prevê alguns passos bem definidos para a obtenção do produto resultante e, conseqüentemente conclusão da fase.	Cada fase tem como produto um modelo bem definido que será usado nas fases posteriores.
MaSE	É uma metodologia de propósito geral para a análise, o projeto, mas que não trata da implementação.	Refere-se à análise e ao projeto e oferece suporte ao desenvolvimento através da ferramenta agentTool.	Compreende duas fases, a de análise e de projeto, cada uma com passos bem definidos para a respectiva conclusão.	Os produtos resultantes de cada passo da metodologia não estão muito bem definidos.
Gaia	É adequada para a criação de software novo, reengenharia e reuso de componentes, mas não suporta a engenharia reversa, de código a modelos, nem prototipação.	É bastante limitada, referindo-se apenas à análise e ao projeto, requerendo que o desenvolver ajuste seus produtos a uma tecnologia de programação em particular.	Tanto a fase de análise quanto a de projeto prevêem algumas atividades cada uma, mas em ambos os casos são dadas poucas explicações em relação à execução dessas atividades.	Cada estágio origina seus próprios produtos, o mesmo valendo para as respectivas atividades, sendo bem formulada no que diz respeito a esse ponto.

MESSAGE	É uma metodologia que compreende apenas a análise e o projeto do sistema, não oferecendo suporte à engenharia reversa e nem à prototipação.	É limitada à análise e projeto da aplicação requerendo que o desenvolvedor ajuste seus produtos a uma tecnologia de programação em particular.	A fase de análise possui passos e modelos bem claros, porém a fase de projeto não está muito bem definida.	Os produtos da fase de análise são modelos do sistema e do ambiente conforme acordado entre desenvolvedor e cliente.
Tropos	É adequada para a criação de software novo, reengenharia, reuso de componentes e à prototipação.	Abrange desde a coleta inicial de requisitos, passando pela fase de projeto arquitetural até a implementação.	Dividida em cinco fases principais, cada uma com seus passos bem definidos, principalmente na fase de requisitos.	Cada fase origina seus produtos, que serão refinados ou utilizados pela fase subseqüente.

Tabela 3.8: Quadro comparativo de aspectos dos processos abordados pelas metodologias

Por fim, referente à pragmática que envolve a construção de uma determinada aplicação, a Tabela 3.9 detalha as questões mais importantes que são tratadas pelas metodologias pesquisadas.

Quest. / Metodol.	**Recursos disponíveis**	**Destreza requerida**	**Adequação da linguagem**	**Aplicabilidade ao domínio**	**Escalabilidade do emprego**
PASSI	Dispõe de uma ferramenta, a PTK, bem como de uma vasta documentação bastante elucidativa.	Exige uma familiaridade com a UML e com o paradigma orientado a objetos.	Não se destina a uma em particular, mas através do PTK é possível reusar padrões na plataforma JADE, implementada em Java.	Como abrange todo o processo de desenvolvimento, adequa-se a uma grande gama de aplicações.	É apropriada à construção de projetos de grande porte, devido a atenção dada à geração automática de código.
MaSE	Dispõe do agentTool, uma ferramenta para a construção de aplicações multiagente, e de bons documentos sobre seu emprego.	Exige uma familiaridade com a UML, o paradigma orientado a objetos e as técnicas de análise e de projeto da OMT.	É independente de qualquer tipo de arquitetura de agente e de linguagem de programação.	Apresenta um propósito geral, focando o trabalho cooperativo de agentes em ambientes heterogêneos.	Destina-se principalmente à solução de problemas complexos, mas contempla também outros mais simples.
Gaia	Apesar de bem conhecida, não se possui muito material sobre ela, nem grupos de usuários, trainamentos, ferramentas de automação etc.	Exige um sólido embasamento acerca de lógica simples e temporal, impedindo o uso por não familiarizados com metodos formais.	Não se destina a uma linguagem em particular, podendo ser a especificação implementada através de qualquer linguagem.	É adequada para desenvolver aplicações com metas estabelecidas, agentes heterogêneos, estrutura estática, tamanho reduzido etc.	Não é possível o emprego de apenas uma parte sua no desenvolvimento, mas consegue abranger bom número de aplicações.
MESSAGE	É pouco documentada e não conta com uma ferramenta de suporte.	Requer conhecimento acerca da Linguagem de Modelagem Unificada.	Não se destina a uma linguagem de programação específica.	Propõe-se a desenvolver sistemas multiagente de maneira geral.	Não se observa considerações quanto a isso.

Tropos	A documentação existente permite se ter uma noção do seu uso, mas não muito claramente.	Convém ter alguma noção do paradigma de desenvolvimento orientado a agentes.	É feito o mapeamento dos elementos da modelagem aos conceitos de uma plataforma de programação de agentes.	Propõe-se a desenvolver sistemas multi-agente de maneira geral.	Não se observa considerações quanto a isso.

Tabela 3.9: Quadro comparativo de questões pragmáticas que envolvem as metodologias

3.8 Considerações finais

Neste capítulo foram vistos vários tópicos relativos à Engenharia de Software Multiagente. Assim, procurou-se estabelecer os fundamentados necessários para a propositura de uma metodologia de desenvolvimento nesse paradigma.

Quanto a isso, começou-se pela explicação dos conceitos de agente de software e de sistema multiagente ambos essenciais para o entendimento da abordagem orientada a agentes.

Em seguida, o reuso de software, que havia sido analisado cuidadosamente no capítulo precedente, foi retomado sob a ótica dos agentes, sendo apresentados os processos complemetares e interdependentes da Engenharia de Domínio Multiagente e da Engenharia de Aplicacões Multiagente, os quais, juntos, levam a cabo a reutilização nesse paradigma.

Finalmente, foi apresentado um levantamento de metodologias e ferramentas para a construção de sistemas multiagente. Vale lembrar que, tendo sido baseado apenas na literatura, tal pesquisa apresenta algumas limitações. No entanto, dela resultou um estudo comparativo, segundo uma série de critérios adotados, que serão úteis para a delineação da metodologia proposta no próximo capítulo.

4 MAAEM – UMA METODOLOGIA BASEADA EM ONTOLOGIAS PARA A ENGENHARIA DE APLICAÇÕES MULTIAGENTE

4.1 Considerações iniciais

Antes de entrar no cerne do presente capítulo, o mais importante deste trabalho, por conter a sua principal contribuição científica, cabe uma recapitulação do que foi exposto até agora.

Assim, inicialmente foram apresentados os principais conceitos que embasam a Reutilização de Software, tais como domínio de aplicação, famílias de sistemas, linhas de produção, desenvolvimento baseado em componentes e ontologias. Logo após, estudou-se a Engenharia de Software Multiagente, abrangendo a definição de agentes de software e de sistemas multiagente e a descrição das fases, passos e produtos desse processo, mas na sua abordagem convencional, sem reuso. Então, a partir disso, com a introdução da reutilização de software, também se apresentou os processos complementares e interdependentes da Engenharia de Domínio Multiagente e da Engenharia de Aplicações Multiagente, tendo-se encerrado com um levantamento e um comparativo das técnicas, metodologias e ferramentas para tanto disponíveis.

Deste ponto em diante, passa-se à proposição da MAAEM – *Multi-agent Application Engineering Methodology*, uma Metodologia para a Engenharia de Aplicações Multiagente.

A MAAEM é uma metodologia para a análise, o projeto e a implementação de aplicações multiagente através da reutilização de artefatos de software anteriormente produzidos na Engenharia de Domínio Multiagente. Sendo inspirada no desenvolvimento baseado em componentes, ela envolve a seleção, adaptação e composição desses artefatos para construção de uma aplicação.

Dessa maneira, a metodologia proposta pressupõe a possível existência de um conjunto de abstrações de software reutilizáveis baseadas em agentes, tais como: modelos de domínio, frameworks multiagente, sistemas de padrões e agentes de software. Todos esses artefatos são produtos da Engenharia de Domínio Multiagente, que no contexto de pesquisa do grupo GESEC (2005) é guiada pela metodologia MADEM – *Multi-agent Domain Engineering Methodology* (GIRARDI, LINDOSO, 2005a), e se encontram resumidos na Tabela 4.1.

<table>
<tr><th colspan="2">Fases</th><th colspan="3">Produtos</th></tr>
<tr><td colspan="2" rowspan="4">Análise de Domínio</td><td>Modelo de Conceitos</td><td colspan="2" rowspan="4">Modelo de Domínio</td></tr>
<tr><td>Modelo de Objetivos</td></tr>
<tr><td>Modelo de Papéis</td></tr>
<tr><td>Modelo de Interações entre Papéis</td></tr>
<tr><td rowspan="6">Projeto de Domínio</td><td rowspan="3">Projeto Arquitetural</td><td>Modelo da Sociedade Multiagente</td><td rowspan="3">Modelo da Arquitetura</td><td rowspan="6">Framework Multiagente</td></tr>
<tr><td>Modelo das Interações entre Agentes</td></tr>
<tr><td>Modelo dos Mecanismos de Cooperação e Coordenação</td></tr>
<tr><td rowspan="2">Projeto Detalhado</td><td>Modelo do Conhecimento e das Atividades do Agente</td><td rowspan="2">Modelos de Agentes</td></tr>
<tr><td>Modelo dos Estados do Agente</td></tr>
<tr><td colspan="2">Modelo do Conhecimento da Sociedade Multiagente</td></tr>
<tr><td colspan="2">Extração e Representação de Padrões</td><td colspan="3">Padrões de Software e Sistemas de Padrões</td></tr>
<tr><td colspan="2">Implementação de Domínio</td><td colspan="3">Agentes de Software</td></tr>
</table>

Tabela 4.1: Fases e produtos da Metodologia MADEM

A MADEM é uma Metodologia para a Engenharia de Domínio Multiagente surgida a partir da integração de outros trabalhos de pesquisa do grupo GESEC: a GRAMO – *Generic Requirement Analysis Method based on Ontologies* (FARIA, 2004), que é um Método baseado em Ontologias para a Análise de Domínio, o qual guia a captura e especificação de requisitos de uma família de sistemas em um domínio de aplicação; e a DDEMAS – *Domain Design for Multi-agent Systems* (FERREIRA, 2004), uma técnica para o Projeto de Domínio de Sistemas Multiagente, a qual guia a captura e especificação do projeto reutilizável de uma família de sistemas em um domínio de aplicação.

Os produtos da MADEM são reutilizados como insumos para compor uma aplicação multiagente específica, uma vez que tenham sido selecionados e adaptados a partir de um conjunto de modelos orientados a agentes representando uma família de aplicações em um domínio. Entretanto, deve-se levar também em consideração que, eventualmente, esse conjunto pode ser vazio, não havendo artefatos de software disponíveis. Nessa hipótese, a reutilização é substituída por um desenvolvimento sistemático, resultando em modelos específicos, mas que podem ser generalizados para o reuso posterior. Em ambos os casos, entretanto, procede-se conforme as fases, subfases, passos, subprodutos e produtos da MAAEM, sintetizados na Tabela 4.2 e detalhados na Seção 4.3 deste capítulo.

<table>
<tr><th colspan="2">Fases</th><th>Passos</th><th colspan="2">Produtos</th><th></th></tr>
<tr><td colspan="2" rowspan="4">Análise
da
Aplicação</td><td>Modelagem de Conceitos</td><td>Modelo de Conceitos</td><td colspan="2" rowspan="4">Especificação
da
Aplicação</td></tr>
<tr><td>Modelagem de Objetivos</td><td>Modelo de Objetivos</td></tr>
<tr><td>Modelagem de Papéis</td><td>Modelo de Papéis</td></tr>
<tr><td>Modelagem de Interações entre Papéis</td><td>Modelo de Interações entre Papéis</td></tr>
<tr><td rowspan="6">Projeto
da
Aplicação</td><td rowspan="3">Projeto
da
Arquitetura</td><td>Modelagem da Sociedade Multiagente</td><td>Modelo da Sociedade Multiagente</td><td rowspan="3">Modelo
da
Arquitetura</td><td rowspan="6">Arquitetura
da
Aplicação</td></tr>
<tr><td>Modelagem das Interações entre Agentes</td><td>Modelo das Interações entre Agentes</td></tr>
<tr><td>Modelagem dos Mecanismos de Cooperação e Coordenação</td><td>Modelo dos Mecanismos de Cooperação e Coordenação</td></tr>
<tr><td rowspan="2">Projetos
de
Agente</td><td>Modelagem do Conhecimento e das Atividades de Agente</td><td>Modelo do Conhecimento e das Atividades de Agente</td><td rowspan="2">Modelos
de
Agente</td></tr>
<tr><td>Modelagem dos Estados de Agente</td><td>Modelo dos Estados de Agente</td></tr>
<tr><td colspan="2">Modelagem do Conhecimento da Sociedade Multiagente</td><td colspan="2">Modelo do Conhecimento da Sociedade Multiagente</td></tr>
<tr><td colspan="2" rowspan="2">Implementação
da
Aplicação</td><td>Mapeamento de Agentes do Projeto à Implementação e de Responsabilidades em Comportamentos</td><td>Modelo de Agentes e Comportamentos</td><td colspan="2" rowspan="2">Modelo de Implementação da Aplicação</td></tr>
<tr><td>Mapeamento de Interações entre Agentes em Atos de Comunicação</td><td>Modelo de Atos de Comunicação entre Agentes</td></tr>
</table>

Tabela 4.2: Fases, subfases, passos, subprodutos e produtos da Metodologia MAEEM

No sentido de promover a integração entre a MADEM e a MAAEM, ambas contam com os benefícios oferecidos pela Engenharia do Conhecimento à Engenharia de Software (FALBO *et al.*, 2002), tendo seu conhecimento representado através de uma ontologia, a ONTORMAS – *Ontology for Reusing Multi-agent Software*, cuja instanciação permite representar os artefatos de software resultantes do processo de desenvolvimento de sistemas multiagente.

A adoção de ontologias decorre, em primeiro lugar, do fato de serem estruturas de representação de conhecimento bastante interessantes para a especificação de abstrações de software de alto nível, em virtude de características como não ambigüidade, formalidade, reusabilidade e adaptabilidade. Em segundo lugar, essa escolha viabiliza a reutilização dos produtos da MADEM como insumos

para a MAAEM, posto que aqueles também são representados através dessa ontologia (GIRARDI, LINDOSO, 2005b), tornando-os compatíveis entre si.

De fato, a metodologia MADEM utilizava a ontologia ONTOMADEM, enquanto, por consequência, a MAAEM utilizaria a ONTOMAAEM. Porém, em vez de se construir esta independentemente daquela, e por razões práticas relativos ao próprio reuso, optou-se por integrar ambas no que se denominou de ONTORMAS, segundo descrito na seção a seguir.

4.2 ONTORMAS – Uma Ontologia para o Reuso de Software Multiagente

A ontologia ONTORMAS funciona como uma ferramenta para a modelagem e um repositório para o armazenamento dos produtos da Engenharia de Domínio e de Aplicações Multiagente, representados na forma de conceitos semanticamente relacionados, sobre os quais é possível realizar inferências, facilitando o seu entendimento e a sua consequente reutilização.

Nela, a construção de artefatos de software orientados a agentes tanto para o reuso – segundo a MADEM – quanto com o reuso – conforme a MAAEM – é feito através da instanciação das correspondentes subclasses de *Tarefas de Modelagem*, *Produtos de Modelagem*, *Conceitos de Modelagem* e *Relacionamentos de Modelagem*, seguindo a semântica estabelecida na própria ontologia.

Tarefas e produtos de modelagem estão bastante relacionados no contexto da ONTORMAS, posto que a realização de uma tarefa sempre origina um respectivo produto. Além disso, ambos são classificados em simples ou compostos, sendo os produtos deste tipo quando agrupam subprodutos – que podem ser, por sua vez, de quaisquer tipos – e sendo de tipo simples quando contêm simplesmente conceitos e relacionamentos de modelagem. Já as tarefas são simples se resultam apenas em produtos também simples e são compostas se originam produtos compostos e, por conseguinte, possuem subtarefas – que igualmente podem ter qualquer tipo – relacionadas com os componentes de tais produtos.

Tais relações estabelecidas na ONTORMAS são vistas na Figura 4.1, onde em uma rede semântica feita através da *Ontoviz Tab* do *Protégé* (2006) aparecem o produto *Modelo de Objetivos*, a tarefa *Modelagem de Objetivos*, os conceitos *Objetivo Geral* e *Específico*, *Responsabilidade* e *Entidade Externa* e o relacionamento *Obtém Informação*, dentre outros elementos intermediários.

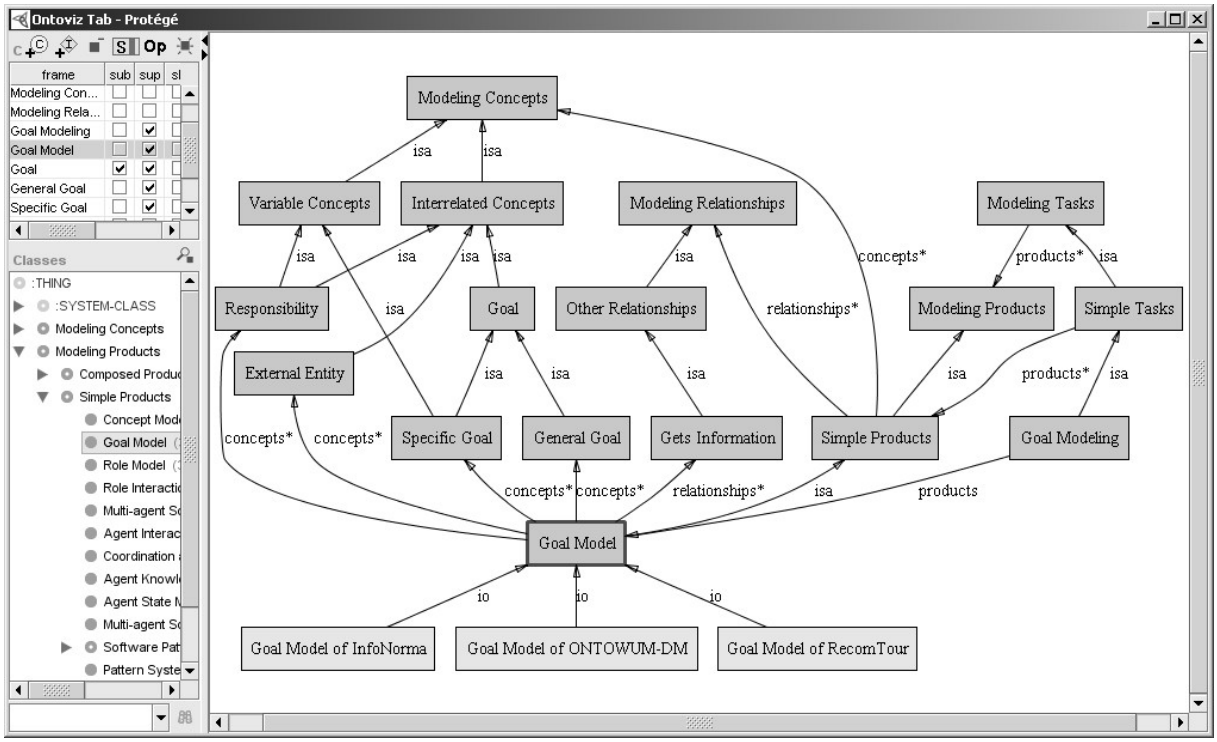


Figura 4.1: Rede semântica relacionando os elementos-chave de modelagem da ONTORMAS, construída através da *Ontoviz Tab* do *Protégé*

No tocante à reutilização, destaca-se a importância da variabilidade, que diferencia as aplicações de uma família (GIMENES, TRAVASSOS, 2005). Tal noção influencia diretamente na seleção e adaptação dos conceitos de modelagem para composição dos produtos, sendo contemplada pela ONTORMAS através do atributo *tipo de variabilidade* da classe *Conceito Variável*, ilustrado na Figura 4.2.

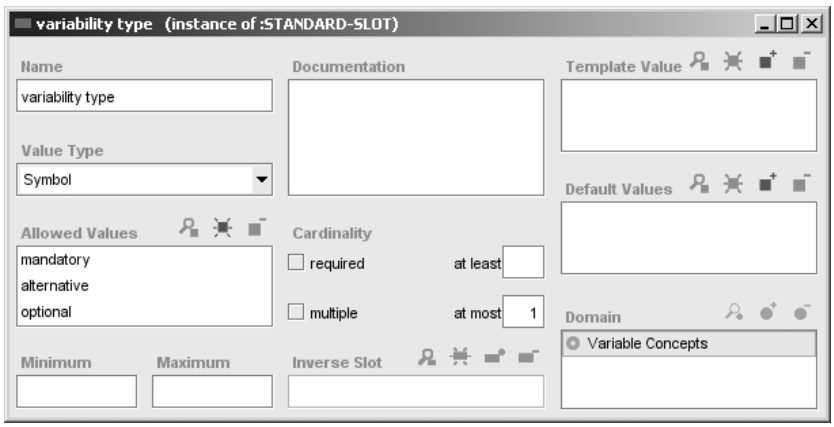


Figura 4.2: Atributo *tipo de variabilidade* da classe *Conceito Variável* da ONTORMAS

A Figura 4.3, a Figura 4.4, a Figura 4.5 e a Figura 4.6 mostram as partes da hierarquia de classes da ONTORMAS, construída com o auxílio do editor de ontologias Protégé (GENNARI *et al.*, 2002).

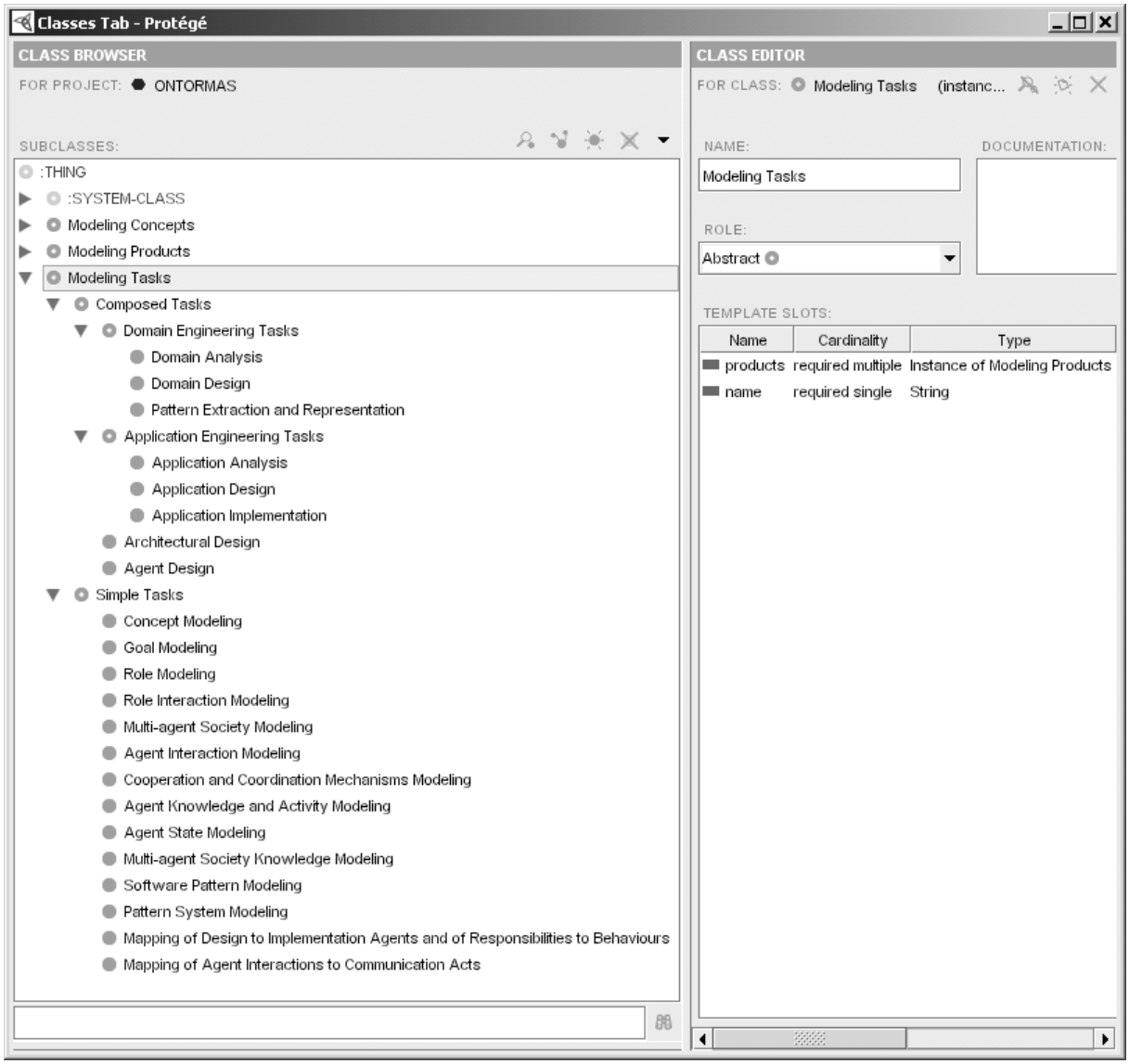


Figura 4.3: Parte da hierarquia de classes da ONTORMAS, destacando a classe das *Tarefas de Modelagem* e suas subclasses

A notação gráfica de todos os produtos e subprodutos de modelagem da ONTORMAS é mostrada ao longo das seções e subseções – correspondentes às tarefas e subtarefas da ontologia e, consequentemente, às fases, subfases e passos da metodologia – dos Capítulos 5 e 6, nos quais são apresentados os estudos de caso da *RecomTour* – Uma Aplicação Multiagente para a Recomendação de Pacotes Turísticos através da Mineração de Uso na Web e da Filtragem Colaborativa e da *InfoNorma* – Uma Aplicação Multiagente para a Entrega Personalizada de Instrumentos Jurídico-Normativos através da Filtragem de Informação baseada no Conteúdo, respectivamente, elaborados com o intuído de avaliar a metodologia ora descrita.

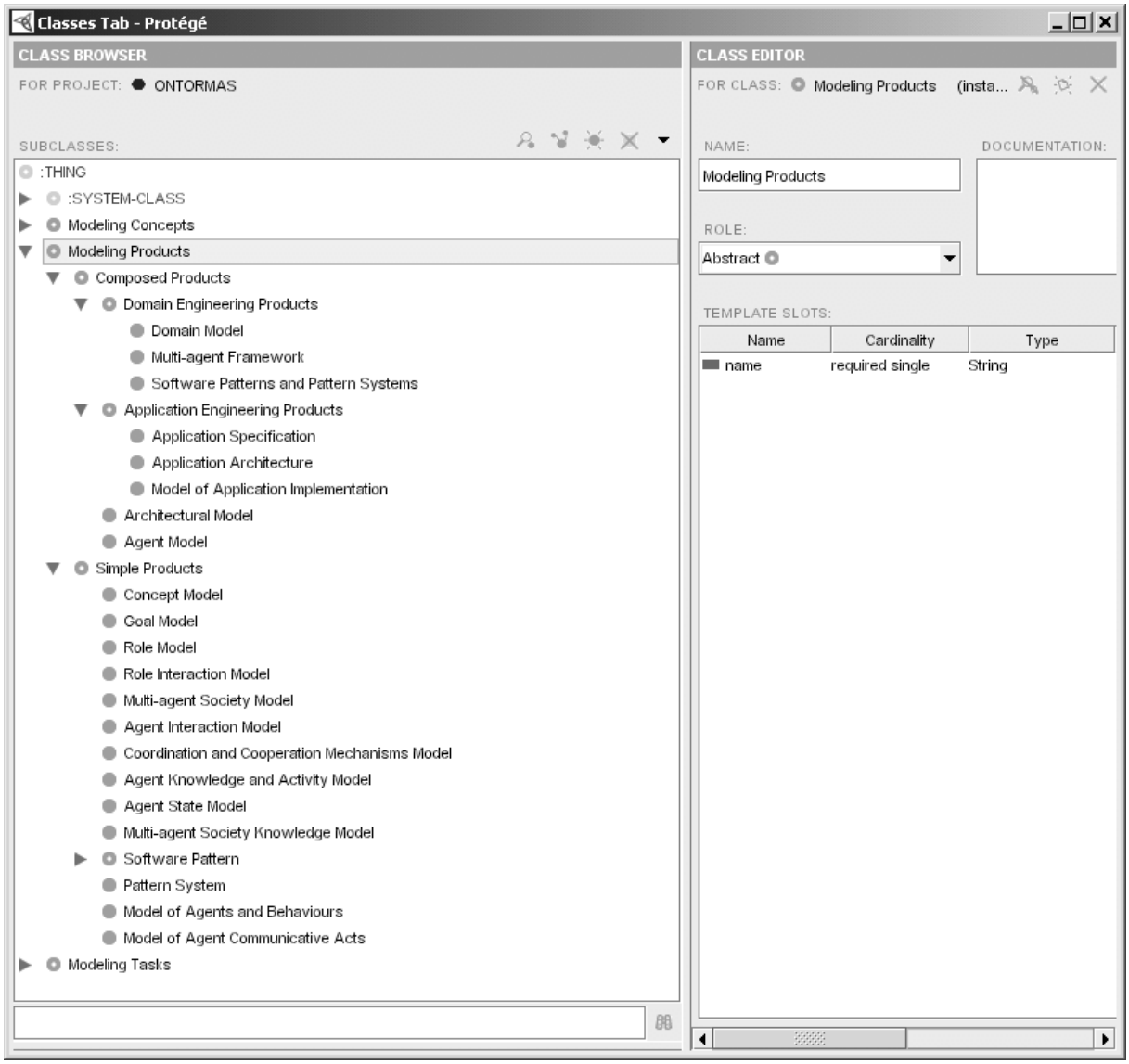


Figura 4.4: Parte da hierarquia de classes da ONTORMAS, destacando a classe dos *Produtos de Modelagem* e suas subclasses

Por outro lado, conceitos e relacionamentos de modelagem são também elementos muito próximos uns dos outros na ONTORMAS. Isso acontece porque são os primeiros que demandam a existência dos segundos, considerando que nenhum modelo fica pronto contendo tão somente conceitos, sendo necessários os relacionamentos entre conceitos para completar a semântica de tais modelos.

Essa noção deriva diretamente das redes semânticas como meios formais para representação de conhecimento (RUSSELL, NORVIG, 2004), já que, a rigor, todos os modelos produzidos com suporte da ONTORMAS guardam alguma semelhança com tais estruturas. É importante notar que os conceitos de modelagem apresentam quatro classificações, sendo elas:

- básicos: não possuem nenhum atributo além do nome e da descrição;
- inter-relacionados: podem estabelecer relações de *sinonímia*, *especialização* ou *generalização* com outros do mesmo tipo;
- variáveis: têm a variabilidade do tipo *mandatória*, se forem exigíveis em todas as aplicações de uma família, *alternativa*, se a escolha de um dentre vários for obrigatória, ou *opcional*, se forem dispensáveis em alguns casos;
- implementacionais: usados no mapeamento dos produtos da análise e do projeto para a plataforma multiagente de implementação JADE – *Java Agent DEvelopment Framework* (BELLIFEMINE *et al.*, 2005a).

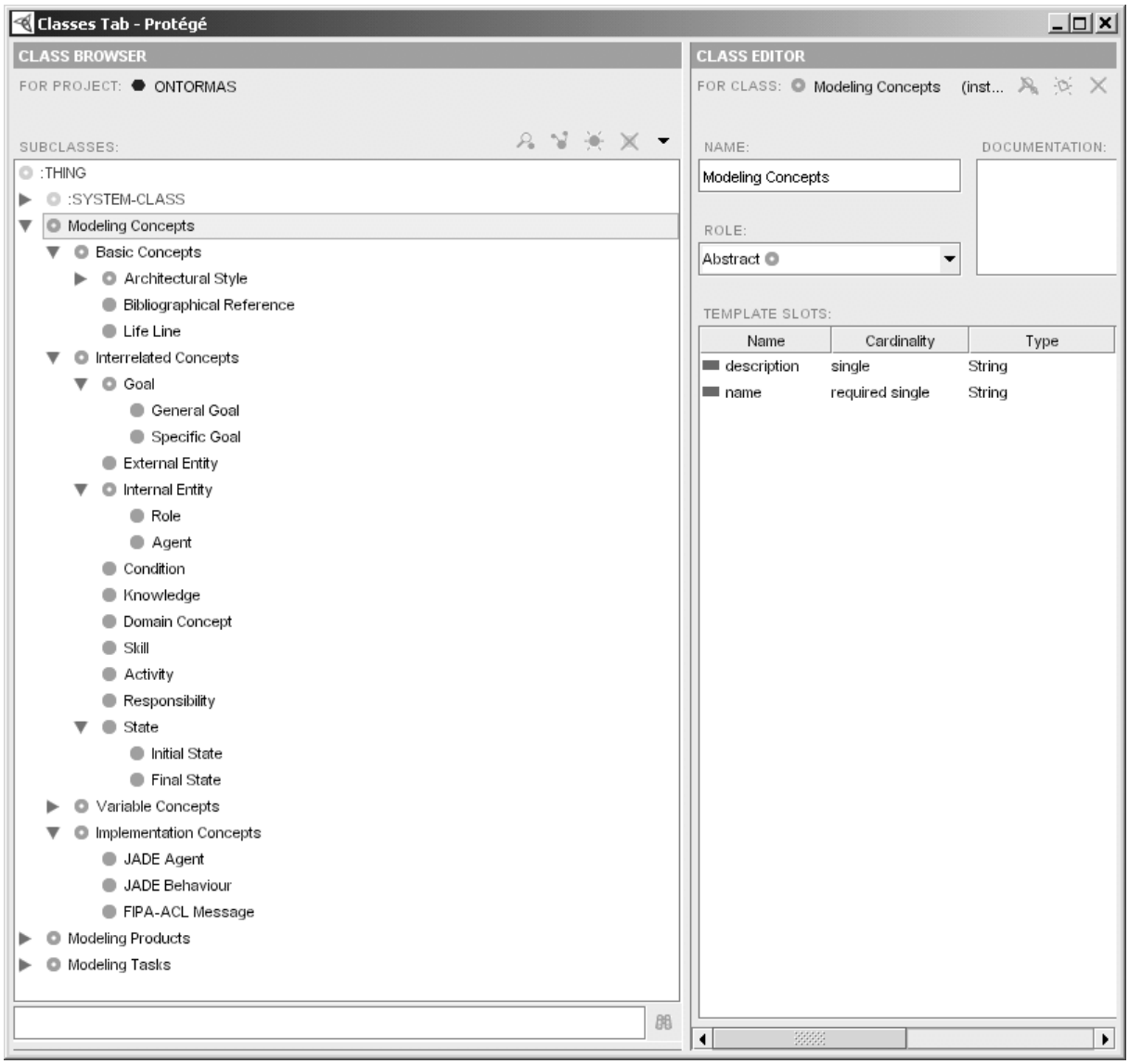


Figura 4.5: Parte da hierarquia de classes da ONTORMAS, destacando a classe dos *Conceitos de Modelagem* e suas subclasses

O significado de cada conceito e relacionamento para os fins das metodologias MADEM e MAAEM é explicitado a seguir, mas também no decorrer do presente capítulo, quando serão indicados quais conceitos são envolvidos em cada tarefa do desenvolvimento, sendo também dito que relacionamentos se estabelecem entre eles, de modo que os respectivos produtos sejam corretamente elaborados, isso durante o passo a passo de desenvolvimento de uma aplicação multiagente específica, seja com ou sem o reuso.

Assim, tais passos envolvem as seguintes noções: os *conceitos* identificam as entidades do domínio e os relacionamentos existentes entre elas, refletindo o conhecimento dos especialistas e permitindo particularizar um problema para certa área; os *objetivos* são o alvo que uma organização busca atingir, atribuindo *responsabilidades* aos indivíduos que a integram, os quais desempenham *papéis* podendo executar *atividades*; para isso, usam e produzem *conhecimentos*, satisfazem *pré* e *pós-condições* e exigem *destrezas*; apesar de autônomos, os papéis, que são *entidades internas*, estabelecem *interações* entre si e também com *entidades externas* para executar atividades mais complexas, além de suas capacidades.

Por outro lado, os *agentes* são as entidades básicas do sistema, que assumem *responsabilidades*, executam *atividades*, usam e produzem *conhecimentos*, satisfazem *pré* e *pós-condições* e exigem *destrezas*. Eles se reúnem em *sociedades multiagente* e, apesar de serem autônomos, estabelecem *interações* entre si para executar atividades mais complexas. Os *mecanismos de cooperação e coordenação* determinam a forma como os agentes se auxiliam e se organizam para atingir o fim da sociedade. O *conhecimento da sociedade multiagente* permite que os agentes se compreendam e se comuniquem. Tudo isso integra a *arquitetura* geral do sistema. Detalhando cada *agente*, há o *conhecimento* e as *atividades*, que informam o seu tipo – se *reativo* ou *deliberativo* – e seus aspectos estruturais, e os *estados*, que dizem o que ele faz, traduzindo aspectos comportamentais.

Por exemplo, em uma universidade, o objetivo geral é a sólida formação acadêmica e profissional, e os objetivos específicos são, em razão disso, a oferta de ensino de qualidade; o incentivo a projetos de pesquisa; e a promoção de atividades de extensão. No mesmo contexto, professores em geral são responsáveis pelo ensino; mestres e doutores em particular, pela pesquisa; e todos estes juntamente com estudantes e técnicos, pela extensão. Sendo ainda que o papel do pesquisador

é favorecer o progresso da ciência, usando seu conhecimento acumulado para produzir novos conhecimentos científicos.

Para isso, são necessárias habilidades como metodologias didáticas e científicas e recursos tais como laboratórios, equipamentos, bibliografia etc. Por fim, no caso da extensão acadêmica, o papel dos estudantes, que são entidades internas, é levar o conhecimento gerado na universidade para a comunidade que a rodeia através da interação com os habitantes locais, líderes comunitários, autoridades públicas, profissionais liberais etc, que são entidades externas.

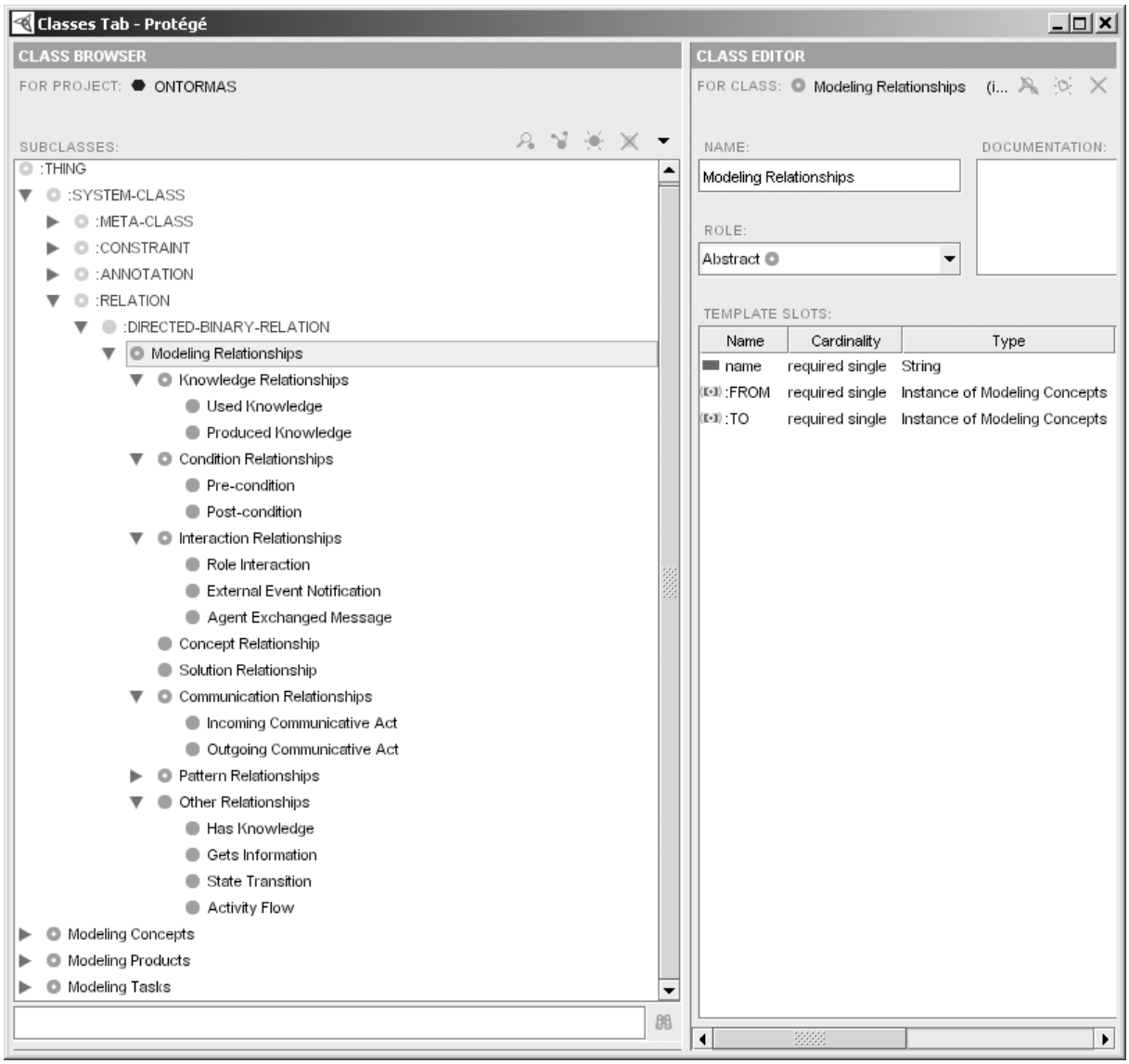


Figura 4.6: Parte da hierarquia de classes da ONTORMAS, destacando a classe dos *Relacionamentos de Modelagem* e suas subclasses

De maneira complementar à descrição acima realizada, no ANEXO 1 deste trabalho se encontra a hierarquia de classes completa da ontologia

ONTORMAS, além da classe *Objetivo Específico* detalhada – a título de exemplo, já que as demais a ela equivalem – da qual se pode verificar uma série de características, dentre tais:

- se é concreta ou abstrata;
- as superclasses que ela estende;
- as subclasses que dela herdam;
- as suas instâncias diretas;
- os seus atributos, contendo:
  - nome,
  - documentação,
  - tipo (por exemplo: *texto*, *instância*, *número*, *símbolo* etc),
  - permissões (valores para *texto* e classes para *instância*),
  - cardinalidade (por exemplo: *um para um*, *zero para muitos* etc),
  - outras facetas (conteúdo padrão, por exemplo).

Outro ponto que merece destaque é quanto à forma como se promove a reutilização através da ONTORMAS. Assim, a Figura 4.7 mostra uma consulta simples feita através da *Queries Tab* do *Protégé* (THE PROTÉGÉ PROJECT, 2005) para a seleção do objeto geral que norteará a modelagem de objetivos da aplicação *RecomTour*, a qual é feita a partir da comparação do atributo *nome*, da classe *Objetivo Geral*, com os termos “recomendação”, “mineração” e “colaborativa”, resultando na instância contida no respectivo modelo da ONTOWUM (MARINHO, 2005).

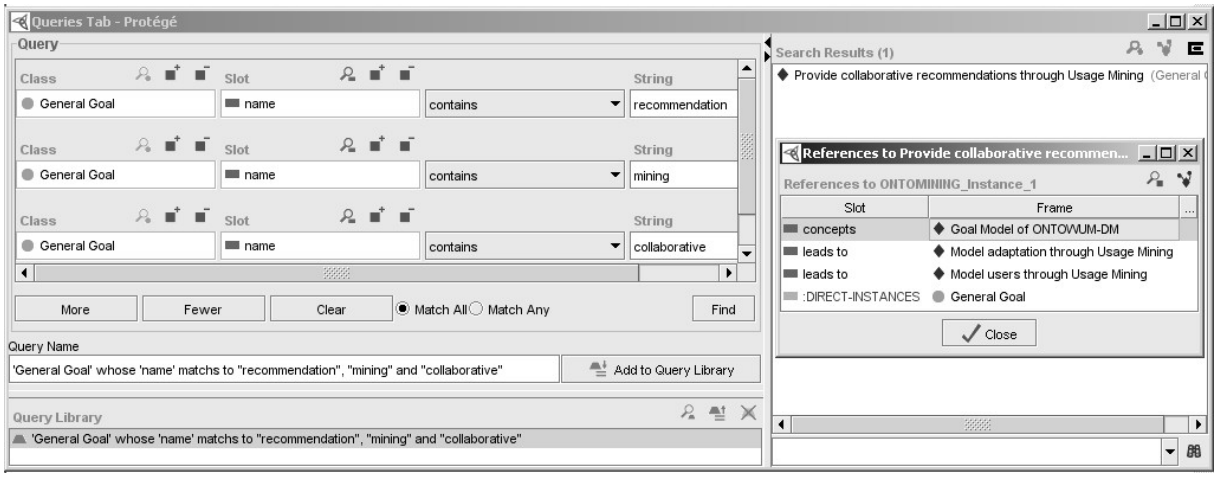


Figura 4.7: Consulta simples através da *Queries Tab* do *Protégé* para seleção de um objetivo geral no repositório da ONTORMAS

Já a Figura 4.8 exibe uma consulta complexa feita através da *Algernon Tab* do *Protégé* (2006) para a seleção dos padrões arquiteturais que orientarão a modelagem dos mecanismos de cooperação e coordenação da aplicação *RecomTour*, por meio da qual se busca instâncias da classe *Padrão Arquitetural* cujos atributos *contexto* e *problema* contenham, respectivamente, as expressões “multiagente” e “nível de abstração”, resultando no padrão de arquitetura em camadas *Multi-agent Layer* (GIRARDI *et al.*, 2005b).

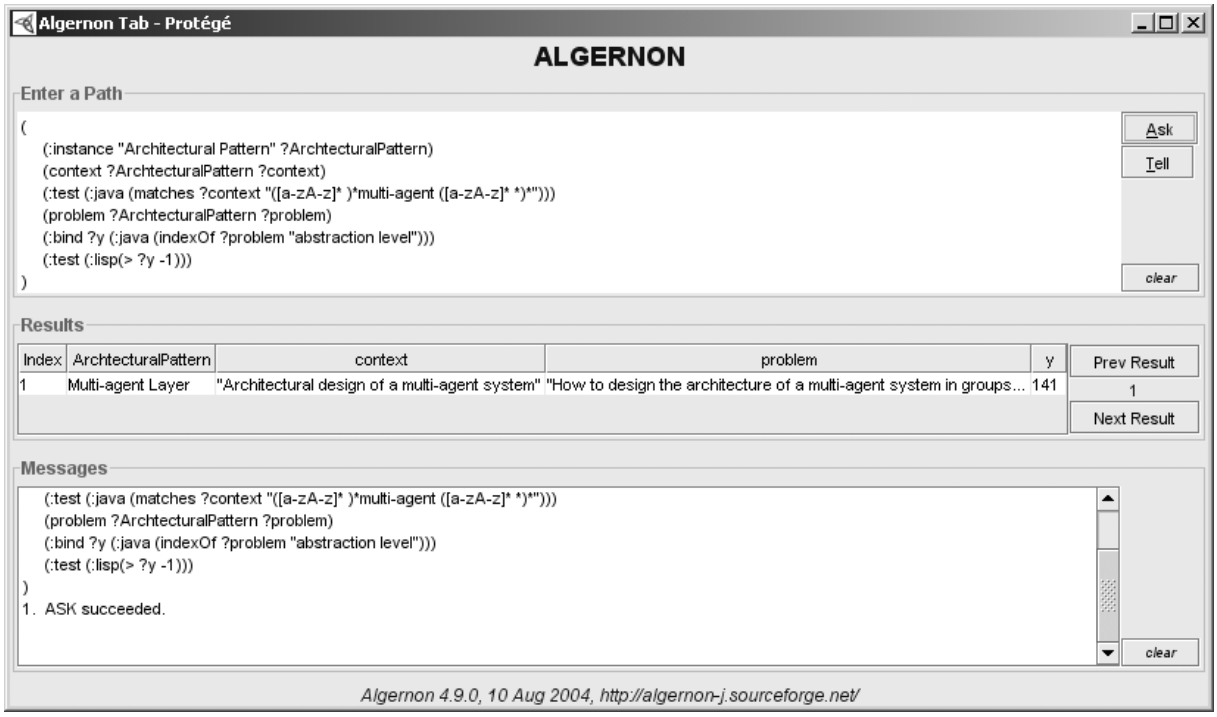


Figura 4.8: Consulta complexa através da *Algernon Tab* do *Protégé* para seleção de padrões arquiteturais no repositório da ONTORMAS

Além da seleção, acima exemplificada, muitas vezes o reuso de artefatos de software orientados a agentes, nos termos da MAAEM, envolve a sua prévia adaptação para adequá-los ao domínio que esteja sendo abordado. Na ONTORMAS, isso é feito quanto às subclasses concretas da classe abstrata *Conceito Inter-relacionado*, nos casos em que alguma instância de uma delas até seja pertinente em certo contexto de desenvolvimento, mas não esteja adequadamente denominada e/ou descrita.

Assim, adaptar um determinado conceito de modelagem consiste na criação de um novo conceito do mesmo tipo, sendo que este indicará em si próprio se é *sinônimo*, *especialização* ou *generalização* daquele que o inspirou, conforme se

tenha adaptado, respectivamente: apenas sua denominação, sem alterar o significado; ou também sua descrição, para lhe conferir um sentido mais estrito ou mais amplo.

A Figura 4.9 mostra um exemplo no qual a instância “Modelar adaptação através de mineração de uso” de *Objetivo Específico*, encontrada no modelo de objetivos da ONTOWUM, foi especializada em “Modelar recomendações” para se ajustar à questão da recomendação de pacotes turísticos, enfocada pela aplicação *RecomTour*. Antes voltado para a personalização de interfaces da Web, depois de adaptado, tal objetivo passou a cuidar especificamente do oferecimento de sugestões de destinos de viagem a turistas.

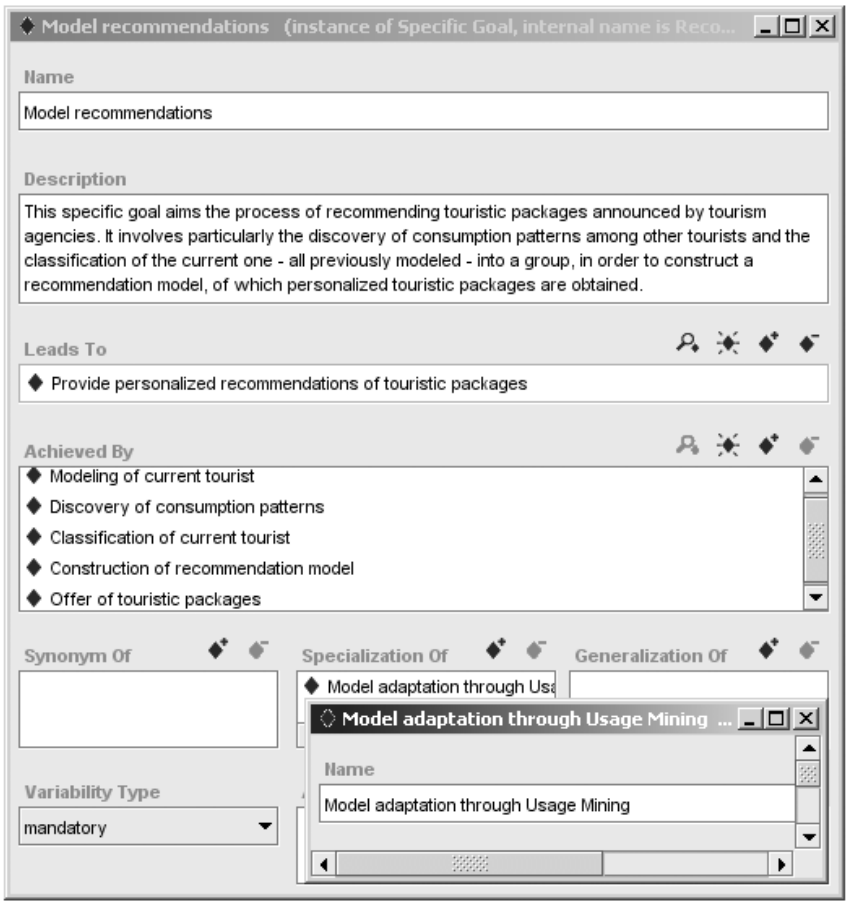


Figura 4.9: Exemplo de adaptação por especialização de uma instância de *Objetivo Específico*, que estende a classe *Conceito Inter-relacionado* da ONTORMAS

Por fim, a composição resulta da própria reunião dos conceitos, através das tarefas de modelagem, estabelecendo-se relacionamentos semânticos entre eles, para formação dos produtos, que são os modelos, seguindo-se as diretrizes da metodologia MAAEM, conforme detalhado na seção a seguir.

4.3 Fases do desenvolvimento através da metodologia MAAEM

Nesta seção, as fases de Análise da Aplicação, Projeto da Aplicação e de Implementação da Aplicação da metodologia MAAEM são descritas em termos dos passos que envolvem e dos produtos que originam.

4.3.1 Análise da Aplicação

Na fase de análise da Engenharia de Aplicações Multiagente, pretende-se elaborar a especificação dos requisitos de uma determinada aplicação, partindo de modelos de domínio, que são artefatos de software de elevado grau de abstração elaborados na fase correspondente da Engenharia de Domínio Multiagente.

Desse modo, a tarefa central do desenvolvedor é reusar um conjunto de requisitos compartilhados por todas as aplicações de uma família em um domínio – ou seja, comuns a elas – e os complementar com requisitos pertencentes somente à aplicação em desenvolvimento – isto é, particulares dela. Tais requisitos específicos da aplicação, quando provenientes da família, são levantados a partir da seleção dentre os requisitos fixos ou variáveis no domínio, sendo que, enquanto aqueles são sempre reutilizados, no caso destes o reuso depende da variabilidade estabelecida. E, de qualquer forma, ambos os tipos de requisitos podem passar por adaptações antes da composição final da especificação que guiará o projeto e a implementação.

Nas subseções subsequentes, a descrição do passo a passo dessa fase – que envolve a *Modelagem de Conceitos*, a *Modelagem de Objetivos*, a *Modelagem de Papéis* e a *Modelagem de Interações entre Papéis* – é exemplificada com a *Especificação da Aplicação* multiagente *RecomTour*, já mencionada na seção precedente, a qual promove a reutilização do modelo de domínio da ONTOWUM-DM, constante do APÊNDICE A.

4.3.1.1 Modelagem de Conceitos

No paradigma multiagente, a definição de conceitos – os quais identificam as entidades do domínio conforme o conhecimento dos especialistas na área – é fundamental porque baseia a futura construção da ontologia que representará o conhecimento compartilhado pelos agentes de uma aplicação. Isso é feito a partir de

modelos de conceitos, representados em redes semânticas nas quais os nós são os conceitos e os arcos são as relações entre conceitos. O produto desse passo é o *Modelo de Conceitos*, tendo também forma de uma rede semântica.

Assim, a modelagem de conceitos segundo a MAAEM serve como uma "tormenta de idéias" e começa a partir da análise informal dos requisitos da aplicação, com a reunião de um conjunto inicial de conceitos, estendendo-se em seguida na forma de um refinamento progressivo.

Uma vez identificados alguns conceitos-chave da aplicação, inicia-se a seleção de conceitos disponíveis no repositório da ONTORMAS de forma similar às consultas ilustradas na Figura 4.7 e na Figura 4.8, tomando-se tais conceitos-chave como parâmetros para o reuso e se levando em conta a variabilidade de cada um, o que permite ir enriquecendo a rede semântica.

A composição do modelo decorre do relacionamento final de todos os conceitos presentes no modelo, sejam os reutilizados sejam os novos, de maneira que resulte uma representação uniforme e coerente do conhecimento básico do problema enfocado pela aplicação em questão.

A Figura 4.10 exibe o *Modelo de Conceitos* da *RecomTour*, que reúne os principais conceitos relativos à recomendação personalizada de pacotes turísticos, alguns obtidos a partir do reuso do respectivo modelo da ONTOWUM-DM.

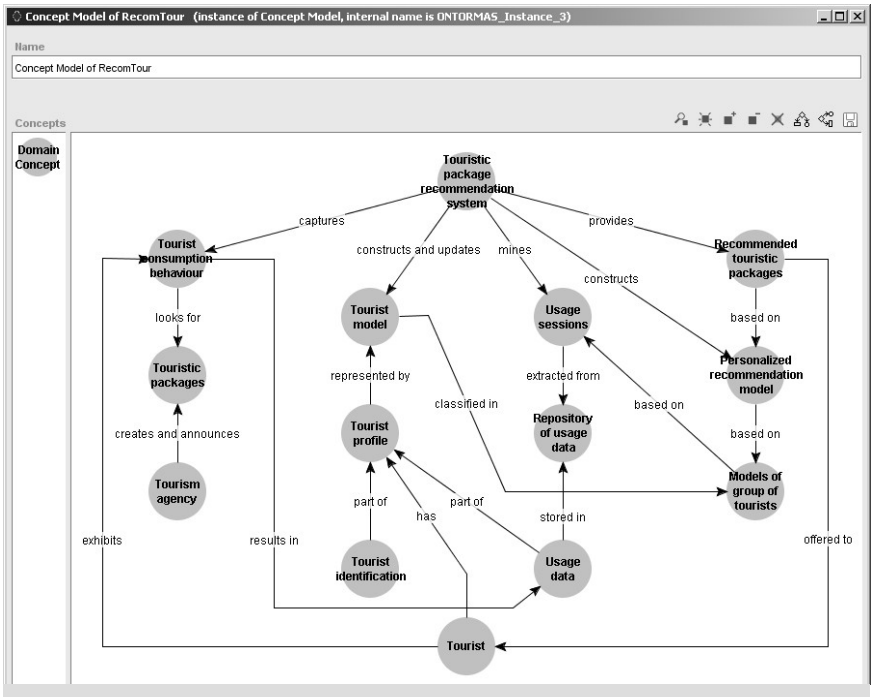


Figura 4.10: Modelo de Conceitos da *RecomTour*

Não havendo modelos de conceitos para serem reutilizados, o *Modelo de Conceitos* de uma aplicação deve ser desenvolvido para o caso específico. Assim, primeiramente, através da consulta a especialistas e a outras fontes de informação no domínio, são identificados os principais conceitos e relacionamentos. Em seguida, aplicações existentes no domínio também são examinadas. Feito isso, é construída uma rede semântica representando tais conceitos e relacionamentos.

Padrões de análise podem ser reutilizados na construção de um *Modelo de Conceitos*, já que eles representam grupos de conceitos e relacionamentos que descrevem modelos de negócio resultantes de experiências de desenvolvimento repetidas com sucesso.

4.3.1.2 Modelagem de Objetivos

A definição de uma hierarquia de objetivos, de um conjunto de responsabilidades e das entidades externas que trocam informações com a aplicação corresponde a outra parte da especificação de requisitos guiada pela MAAEM. O produto desse passo é o *Modelo de Objetivos*, que embasa fortemente o restante dos passos, tendo a forma de um diagrama organizacional de vários níveis.

A seleção se inicia com o estabelecimento do objetivo principal da aplicação. Em seguida, partindo de modelos de domínio que contemplem o problema abordado pela aplicação, procede-se a uma operação similar mostrada na Figura 4.7 e na Figura 4.8, através da qual se selecionarão objetivos que casem, parcial ou totalmente, com o objetivo da aplicação.

Os objetivos são organizados, segundo a hierarquia estabelecida entre eles, como geral e específicos, sendo aquele fixo e estes variáveis. Assim, quando o objetivo da aplicação corresponder ao objetivo geral de um dado modelo de objetivos no repositório da ONTORMAS, então este e todos os objetivos específicos cuja variabilidade seja mandatória serão selecionados. Do contrário, quando a correspondência ocorrer com um objetivo específico, então a reutilização abrangerá somente este e os objetivos específicos mandatórios relacionados. Em ambos os casos, quanto aos demais objetivos específicos, selecionam-se os opcionais que contribuam para o alcance do objetivo da aplicação e, dentre os alternativos, aqueles que melhor se adequam a esse fim.

Após isso, é feita também a seleção de responsabilidades dentre aquelas contidas nos modelos de objetivos reutilizados, considerando-se para tanto, a variabilidade das que permitem alcançar algum dos objetivos específicos selecionados. O mesmo ocorre em relação às entidades externas, porém, sem levar em conta variabilidades.

Uma vez que tenham sido selecionados todos objetivos, responsabilidades e entidades externas, eles são então eventualmente adaptados para compor o *Modelo de Objetivos*, que conterá: no seu nível superior, o próprio objetivo da aplicação como objetivo geral; nos níveis intermediários, os objetivos específicos, hierarquizados segundo o estavam nos modelos de objetivos originários; e no nível inferior, as responsabilidades identificadas.

Na Figura 4.11 se observa parte do *Modelo de Objetivos* da *RecomTour*. Para sua produção, recorreu-se ao modelo de objetivos da ONTOWUM-DM, de onde foi reutilizada a parte relativa à modelagem de usuários e à adaptação de sistemas através da mineração de uso.

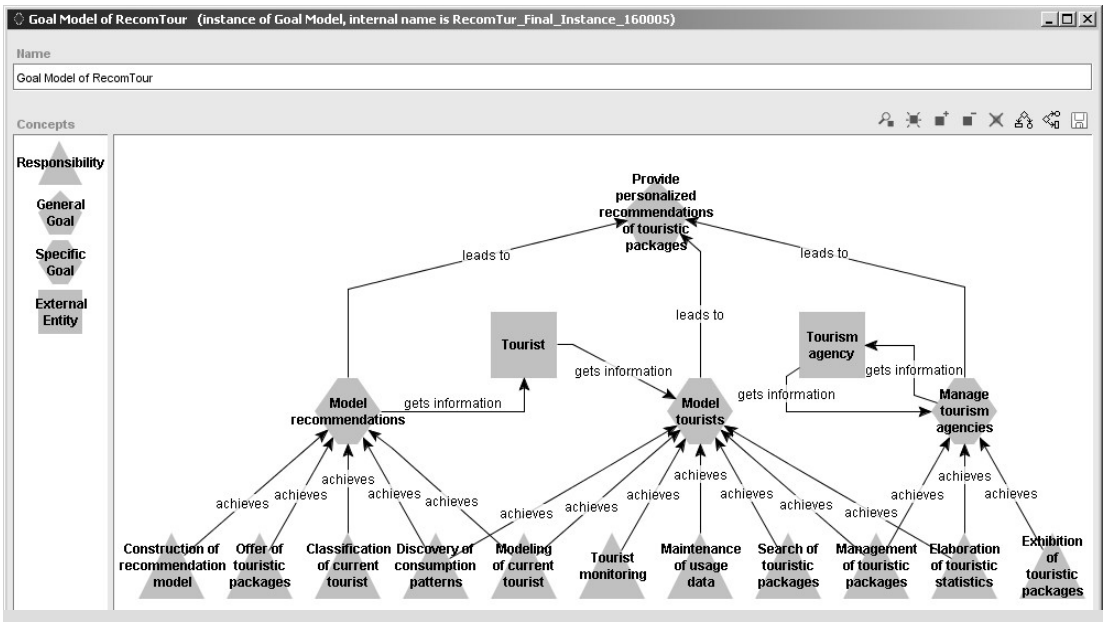


Figura 4.11: Modelo de Objetivos da *RecomTour*

Não havendo possibilidade de reuso por falta de modelos de objetivos apropriados ou sendo eles incompletos para tanto, o *Modelo de Objetivos* da aplicação é elaborado da forma como segue. Em primeiro lugar, é definido um objetivo geral para a aplicação, correspondendo ao problema que ela pretende resolver. Então, a partir da decomposição deste em subproblemas menores, são

obtidos os objetivos específicos que conduzem àquele objetivo geral e que podem ainda apresentar subdivisões, mas sendo, em todos os casos, abordagens mandatórias, alternativas ou opcionais para as questões focadas. Logo após, responsabilidades são identificadas, as quais permitirão atingir os objetivos específicos. São também definidas as entidades externas que obtêm informações da aplicação ou vice-versa. Por fim, o *Modelo de Objetivos* é construído do modo já descrito acima.

4.3.1.3 Modelagem de Papéis

Nos termos da MAAEM, identificadas as responsabilidades, são então a elas associados os papéis, bem como os conhecimentos usados e produzidos, as pré e pós-condições impostas e as destrezas requeridas. Além disso, são introduzidas as entidades externas que contribuem para a execução de tais responsabilidades, que podem ser exercidas através de atividades de igual natureza mas de maior especificidade, as quais estabelecem, portanto, as mesmas associações acima mencionadas. O produto desse passo é o *Modelo de Papéis*, representado graficamente por um diagrama organizacional de três níveis.

A seleção dos papéis e das entidades externas a serem reutilizados é feita com base nos modelos de domínio previamente selecionados para o reuso. Neles, por meio de um procedimento com aquele visto na Figura 4.7 e na Figura 4.8, buscam-se os papéis relacionados com cada responsabilidade reutilizada, assim como as entidades externas decorrentes da modelagem de objetivos. Feito isso, deve-se também selecionar e, se necessário, adaptar as atividades que por ventura decomponham as respectivas responsabilidades. Tudo de acordo com a variabilidade que cada atividade possuir.

Em seguida, ocorre a seleção e adaptação dos conhecimentos de entrada e de saída de cada responsabilidade ou atividade, de modo que ao uso de uns equivalha à produção dos outros. Já as destrezas, por apresentarem variabilidade, são tratadas de maneira geral na fase de análise, como uma série de possíveis modos de os papéis agirem no desempenho de suas atribuições, pois serão especializadas na fase de projeto.

Quanto às condições, existem dois tipos, conforme sejam verificadas antes ou depois das correspondentes responsabilidades e atividades, sendo que,

em ambos os casos, aquelas determinam as circunstâncias para a execução destas. Assim, a efetivação de certa pré-condição é a situação suficiente para que uma responsabilidade ou atividade se suceda, isto é, tais condições são prévias e suspensivas, posto que nada acontece até sua satisfação. Por outro lado, a conclusão de dada responsabilidade ou atividade é a situação necessária para que uma pós-condição tenha lugar, ou seja, tais condições são posteriores e resolutivas, já que nenhuma ação ocorre após seu adimplemento.

Portanto, a suficiência e a necessidade dizem respeito à implicação lógica havida entre um antecedente – a pré-condição no primeiro caso e a responsabilidade ou atividade no segundo caso – e um consequente – a responsabilidade ou atividade na primeira hipótese e a pós-condição na segunda hipótese – quer dizer, se aquele é atendido, então este é realizado, e vice-versa.

Desse modo, o *Modelo de Papéis* resultará de uma composição de todos os componentes selecionados e adaptados com base nos modelos de papéis reutilizados, ficando no seu primeiro nível, os papéis; no segundo, as responsabilidades e as atividades e os conhecimentos e as condições; e no terceiro, as habilidades. Na Figura 4.12 é vista parte do *Modelo de Papéis* da *RecomTour*, proveniente do reuso do modelo correspondente na ONTOWUM-DM.

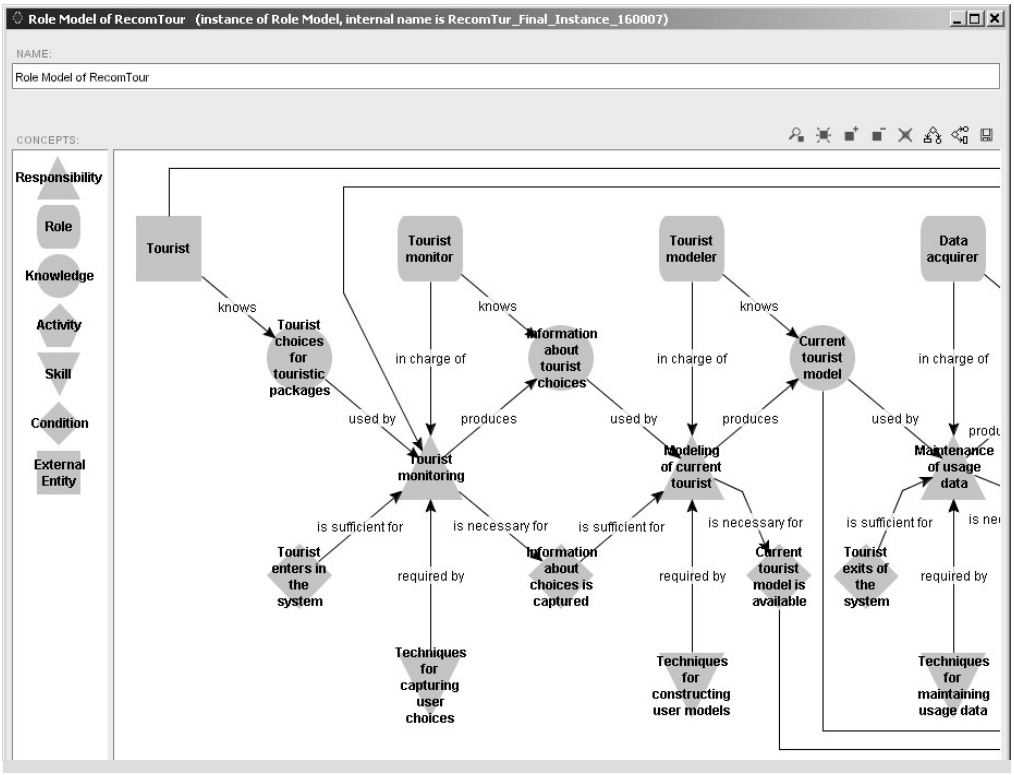


Figura 4.12: Parte do Modelo de Papéis da *RecomTour*

No entanto, se não existir possibilidade de reutilização na construção do *Modelo de Papéis* de uma determinada aplicação, procede-se da seguinte maneira. São acrescidas as já referidas entidades externas constantes do modelo de objetivos. A cada responsabilidade identificada no passo anterior é associado um papel. Essas responsabilidades podem ser decompostas em um conjunto de atividades. Os respectivos conhecimentos usados e produzidos e as condições anteriores e posteriores são também identificados. Finalizando, cada uma daquelas responsabilidades ou atividades demandará uma destreza própria, que pode ser considerada como uma técnica necessária para sua execução.

4.3.1.4 Modelagem de Interações entre Papéis

Concluindo a fase de análise de requisitos de uma aplicação conforme a MAAEM, depois de selecionados, adaptados e compostos os papéis que nela estarão presentes, precisa-se determinar como eles interagem tanto entre si quanto com entidades externas. O produto desse passo é um conjunto – mas que pode ser unitário, dependendo da simplicidade da aplicação – de *Modelos de Interações entre Papéis*, um para cada objetivo específico de mais baixo nível na hierarquia. A sua representação gráfica é similar à de um diagrama de colaboração da AUML (ODELL *et al.*, 2000).

Como as interações dependem fundamentalmente dos papéis e entidades externas envolvidas, então, em função da diversidade de modelos de domínios reutilizados nos passos anteriores, poderá restar prejudicado o reuso neste passo, o que levará a um maior esforço de adaptação e composição em detrimento da mera seleção das interações.

Cada interação entre papéis tem a forma de invocação de um método pelo papel de origem, identificado por uma responsabilidade ou atividade do papel de destino e tendo como parâmetro o conhecimento passado daquele para este. Há também a indicação, por meio de numeração cardinal, da ordem em que ocorrem as interações.

Já quando entidades externas estão envolvidas nas interações, seja como originárias seja como destinatárias, visto que elas não possuem responsabilidades nem atividades, e que também não têm conhecimento das possuídas pelos papéis,

então as mensagens são identificadas por termos que designam ações tais como *exibir*, *obter* ou *fornecer* determinados conhecimentos.

Assim, deve-se analisar em conjunto as responsabilidades e atividades relacionadas com cada papel, bem como os conhecimentos usados e produzidos por elas, e também os conhecidos por entidades externas, para se adaptar as interações antes estabelecidas nos modelos de interações entre papéis reutilizados. A Figura 4.13 mostra uma dos *Modelos de Interações entre Papéis* da *RecomTour*, decorrente em alguma medida do reuso da ONTOWUM-DM.

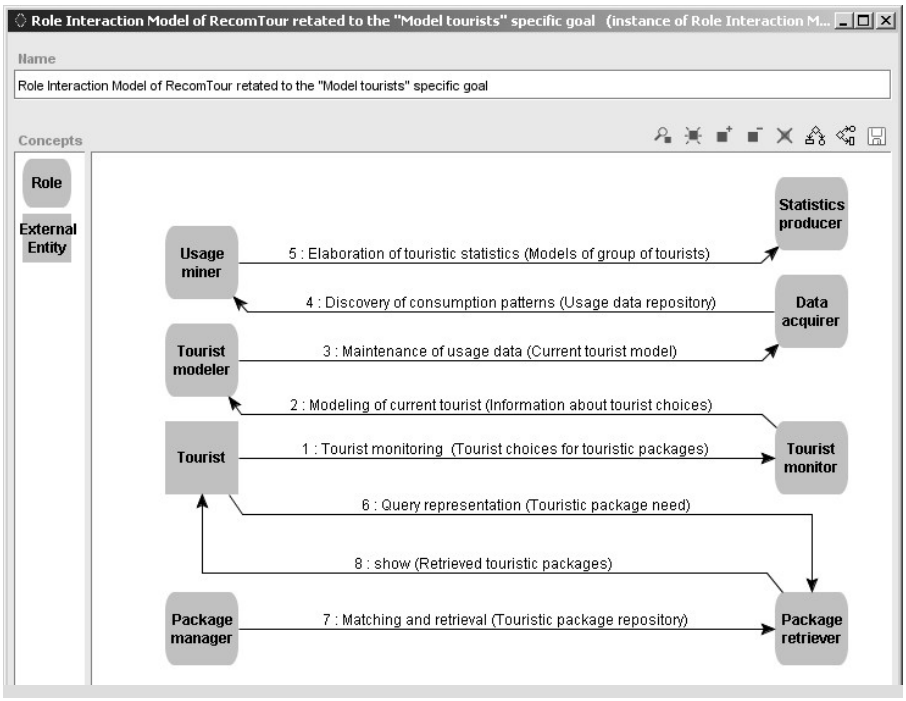


Figura 4.13: Modelo de Interações entre Papéis da *RecomTour* referente ao objetivo específico *modelar turistas*

A ausência de modelos de interações entre papéis aptos à reutilização impõe a elaboração dos *Modelos de Interações entre Papéis* de uma aplicação a partir do zero, mas como já dito, isso não acrescenta muita dificuldade nesse passo, posto que mesmo sendo possível o reuso, este é sempre reduzido em virtude de particularidades inerentes ao modelo.

4.3.2 Projeto da Aplicação

A fase de projeto da Engenharia de Aplicações Multiagente consiste de três subfases, que são: o *Projeto da Arquitetura*, que define a visão global e externa

da sociedade multiagente, sobretudo, passando pela escolha de seus mecanismos de cooperação e coordenação; o *Projeto do Agente*, que estabelece a visão detalhada e interna de cada agente, modelando sua estrutura e seu comportamento; e a *Modelagem do Conhecimento da Sociedade Multiagente*, que identifica os conceitos compartilhados por todos os agentes em sua comunicação.

Desse jeito, a tarefa principal do desenvolvedor é reusar soluções de projeto relativas a uma família de aplicações em um domínio, complementando-as com outras referentes apenas à aplicação em construção, gerando a arquitetura específica e seus respectivos detalhamentos que orientarão a futura implementação.

A reutilização tem como base frameworks multiagente, que são artefatos de software de elevado grau de abstração resultantes da fase correspondente da Engenharia de Domínio Multiagente, além da especificação de requisitos da aplicação, produzida na fase precedente da Engenharia de Aplicações Multiagente. Portanto, o esforço de desenvolvimento se concentra na seleção, adaptação e composição dos elementos daqueles arcabouços, de acordo com os requisitos especificados para a aplicação. Padrões arquiteturais e de projeto detalhado também podem influenciar na resolução do problema.

Nas subseções seguintes, a descrição dos passos dessa fase é ilustrada com a *Arquitetura da Aplicação* multiagente *RecomTour*, já indicada anteriormente, a qual reusa o framework multiagente da ONTOWUM-DD, constante do APÊNDICE A.

4.3.2.1 Projeto da Arquitetura

Na subfase inicial do Projeto da Aplicação conforme a MAAEM, o objetivo é desenvolver o *Modelo da Arquitetura*, representando uma solução orientada a agentes para os requisitos da aplicação especificada na fase de análise e que compreende a *Modelagem da Sociedade Multiagente*, *Modelagem das Interações entre Agentes* e a *Modelagem dos Mecanismos de Cooperação e Coordenação*.

a) Modelagem da Sociedade Multiagente

Nesse passo, o ponto de partida são os papéis identificados na *Especificação da Aplicação*, que servem como critério para seleção de agentes nos frameworks multiagente reutilizados, os quais devem contemplar o problema

abordado. O produto desse passo é o *Modelo da Sociedade Multiagente*, representado graficamente por um diagrama organizacional de três níveis.

Assim, feita a seleção de agentes que casem com os papéis previamente reutilizados, a adaptação consistirá em refazer o mapeamento dos agentes em face das responsabilidades, já que os papéis estabeleciam relações de um para um, mas agora se pode ter um daqueles desempenhando mais de um destes, pois regras de coesão funcional, semelhança de responsabilidades ou interações excessivas podem indicar a conveniência em se fundir vários papéis em um único agente.

Quanto ao reuso dos demais elementos associados aos agentes, tais como conhecimentos, condições e destrezas, nenhuma grande empreitada é requerida, senão por estas últimas, que lá na fase de análise apenas comportavam possíveis técnicas aptas a auxiliar no cumprimento da responsabilidade ou atividade correspondente e aqui na fase de projeto são tratados com mais cuidado, sendo caracterizadas por um algoritmo, procedimento ou tecnologia própria que tenha sido escolhida.

Na Figura 4.14 é vista parte do *Modelo da Sociedade Multiagente* da *RecomTour*, composto exatamente como o modelo de papéis, mediante a reutilização do respectivo modelo da ONTOWUM-DD.

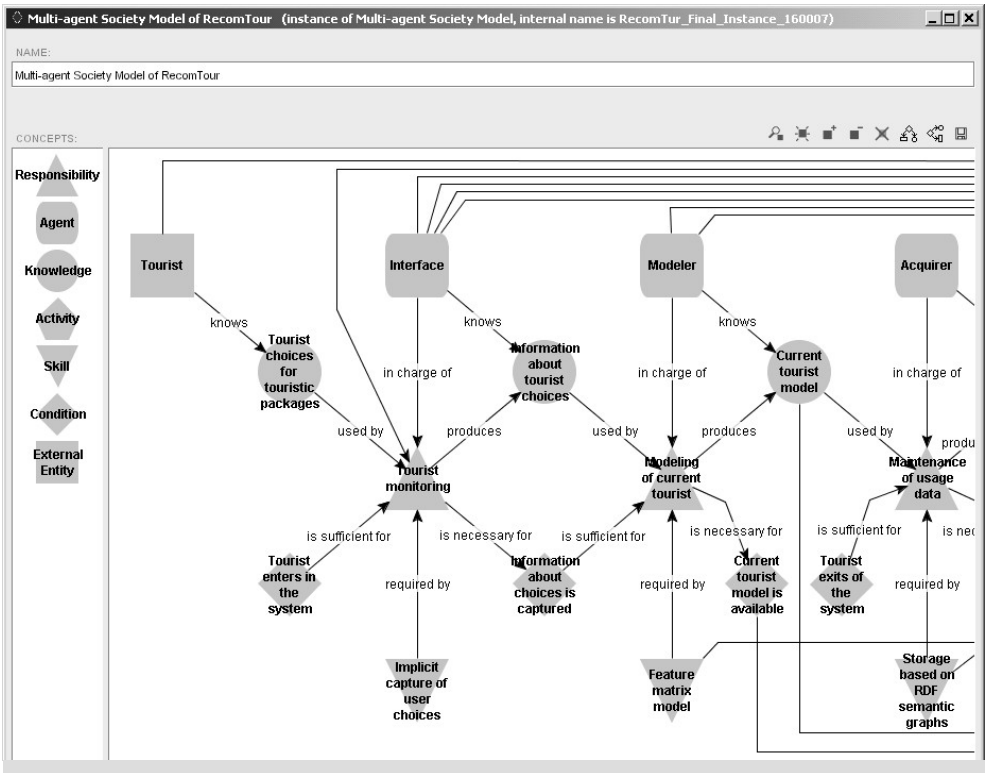


Figura 4.14: Parte do Modelo da Sociedade Multiagente da *RecomTour*

b) Modelagem das Interações entre Agentes

Como consequência da reorgaização das responsabilidades e atividades, com a passagem de papéis a agentes, as interações havidas entre eles também passam a se dar de maneira diversa. O produto desse passo é o *Modelo das Interações entre Agentes*, cuja notação gráfica é similar à de um diagrama de sequência da AUML (ODELL *et al.*, 2000).

Diante de tais mudanças, assim como ocorre entre papéis, nesse passo se têm mais adaptações que apenas seleções na composição do modelo, em função das alterações que podem acontecer até mesmo com a inclusão ou exclusão de um único agente em relação ao modelo de interações entre agentes reutilizados.

De acordo com as linhas de vida, ordena-se as interações, que podem ser do tipo *mensagem trocada entre agentes* ou *notificação de evento externo*, sendo que enquanto aquelas têm agentes como remetentes e destinatários e são compostas por uma performativa da linguagem de comunicação – cujos tipos estão descritos na Seção 4.3.3.2 – e por um conhecimento passado de um para o outro, estas partem de entidades externas para agentes e contêm uma pré ou pós-condição satisfeita. A Figura 4.15 mostra o *Modelo das Interações entre Agentes* da *RecomTour*, que faz reuso do correspondente modelo da ONTOWUM-DD.

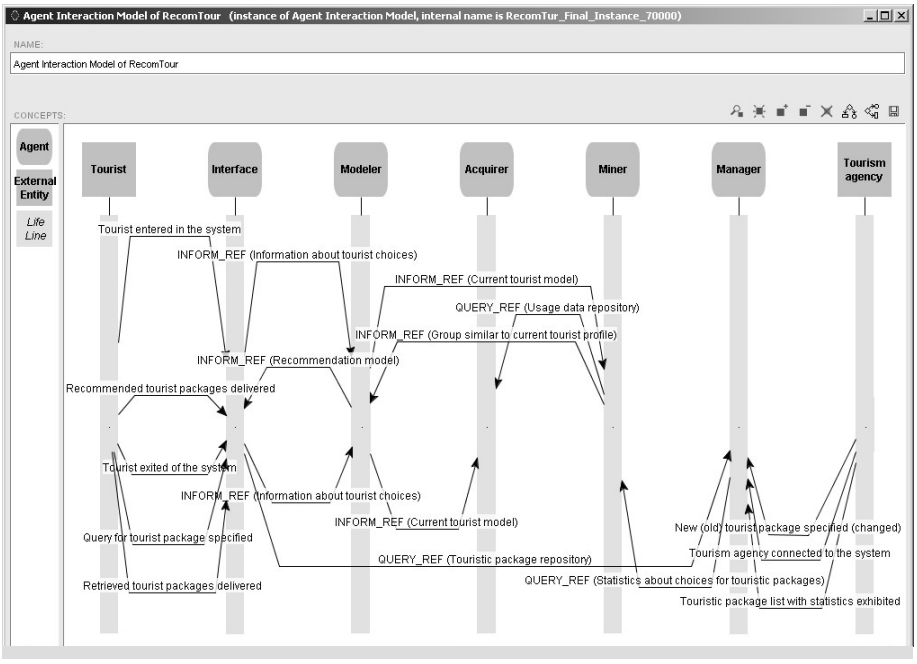


Figura 4.15: Modelo das Interações entre Agentes da *RecomTour*

c) Modelagem dos Mecanismos de Cooperação e Coordenação

Partindo do *Modelo da Sociedade Multiagente* e do *Modelo das Interações entre Agentes*, considerados até então apenas como um esboço arquitetural da aplicação, e segundo requisitos funcionais e não-funcionais previamente analisados, deve-se agora organizá-la através da seleção e adaptação de mecanismos de cooperação e coordenação entre agentes (HUHNS, STEPHENS, 1999) (JENNINGS, 1993). O produto desse passo é o *Modelo* dos *Mecanismos de Cooperação e Coordenação*, representando a arquitetura adotada.

Padrões arquiteturais (GIRARDI *et al.*, 2005a) (GIRARDI *et al.*, 2005b) (GIRARDI *et al.*, 2003) (SILVA JUNIOR, 2003) e regras de projeto como coesão funcional também são considerados.

Os padrões arquiteturais são selecionados no repositório da ONTORMAS (GIRARDI, LINDOSO, 2005c) de maneira análoga à ilustrada na Figura 4.7 e na Figura 4.8. Construída no Protégé, e com o suporte à criação, execução e gravação que ele provê, torna-se possível realizar facilmente a seleção de padrões em tal ontologia.

Para tanto, são comparadas expressões textuais com determinados atributos de classes da ontologia. Assim, a descrição do objetivo da aplicação é confrontada com o problema descrito em cada padrão e com seu contexto de aplicação, no caso, o dos sistemas multiagente.

De acordo com a solução definida, a adaptação decorrente de sua adoção pode resultar no agrupamento – em camadas ou federações, por exemplo – de agentes, o que exigirá que se adaptem também os outros elementos de modelagem associados. Também se escolhe um protocolo de comunicação de agentes – como o FIPA-ACL – para a descrição das interações.

A Figura 4.16 exibe o *Modelo dos Mecanismos de Cooperação e Coordenação* da *RecomTour*, que segue o padrão arquitetural em camadas para sistemas multiagente denominado *Multi-agent Layer* (GIRARDI *et al.*, 2005b), exatamente como foi feito na modelagem dos mecanismos de cooperação e coordenação da ONTOWUM-DD, reutilizada por esta aplicação.

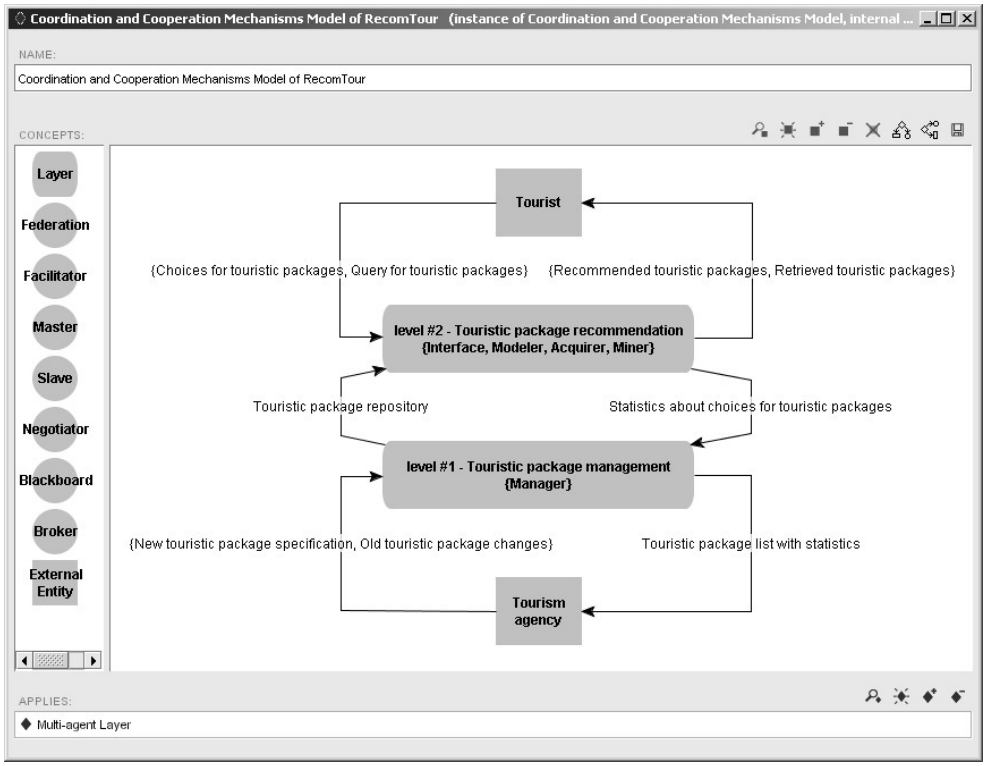


Figura 4.16: Modelo dos Mecanismos de Cooperação e Coordenação da *RecomTour*

4.3.2.2 Projeto de Agente

Na subfase intermediária do Projeto da Aplicação segundo a MAAEM, o alvo é estabelecer o detalhamento de cada agente da sociedade, representado no *Modelo de Agente*, o que envolve a *Modelagem do Conhecimento e das Atividades de Agente* e a *Modelagem dos Estados de Agente*. Nesse sentido, é feito o reuso – de forma parecida com o realizado nas modelagens de conceitos, de objetivos e de papéis – do dos correspondentes modelos que façam parte dos modelos de domínio anteriomente reutilizados. Também se tenta aplicar padrões de projeto detalhado, mediante procedimento similar ao empregado no projeto arquitetural, os quais ajudam a definir o tipo do agente, se reativo ou se deliberativo.

a) Modelagem do Conhecimento e das Atividades de Agente

Nesse passo, busca-se representar isoladamente cada responsabilidade de cada agente da arquitetura, de modo a facilitar a visualização de suas particularidades por parte do implementador, além de introduzir detalhes relativos às técnicas adotadas. O produto desse passo é um *Modelo do Conhecimento e das*

Atividades de Agente por responsabilidade do agente, representado graficamente por um diagrama organizacional em quatro partes.

Nele, da esquerda para a direita aparecem, nesta ordem: as pré e pós-condições a que a responsabilidade – ou as respectivas atividades – se submetam; essas, então, vêm logo a seguir, acompanhadas das correspondentes destrezas requeridas por cada uma; adiante se encontram os conhecimentos usados e produzidos nesse processo; por fim, estão os agentes portados de tais conhecimentos.

A Figura 4.17 ilustra o *Modelo do Conhecimento e das Atividades do Agente Minerador* da *RecomTour* para *descoberta de padrões de uso*, composto, com algumas adaptações, pelos elementos selecionados do respectivo modelo da ONTOWUM-DD.

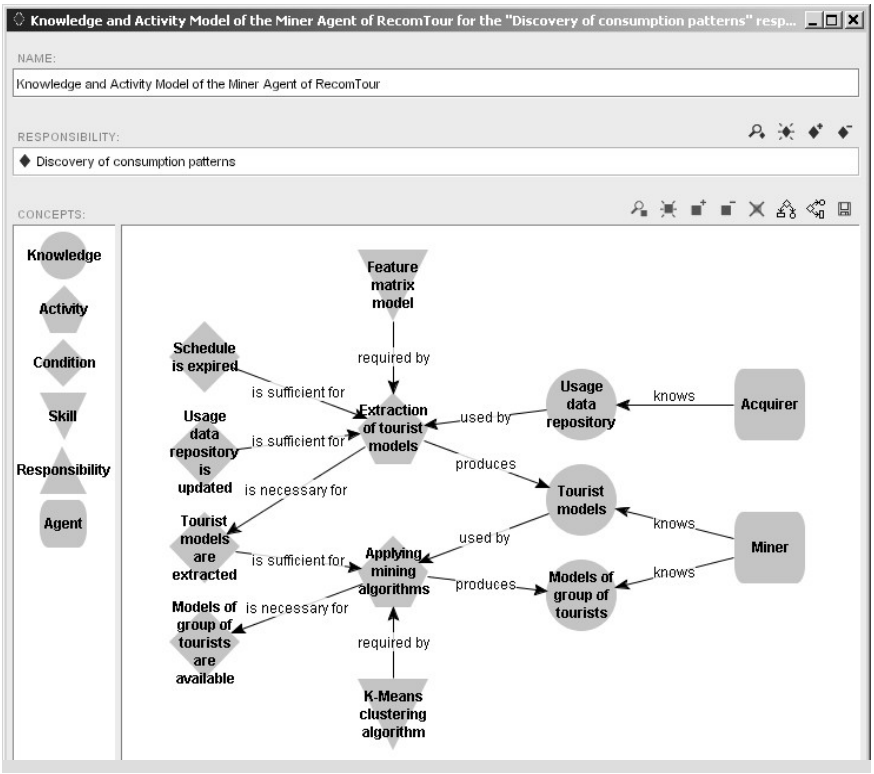


Figura 4.17: Modelo do Conhecimento e das Atividades do Agente *Minerador* para a responsabilidade *descoberta de padrões de consumo*

b) Modelagem dos Estados de Agente

Aqui, procuram-se identicar os estados e transições por que passa cada agente, com base na estrutura da sociedade multiagente, nas interações entre os

agentes e na organização dos mecanismos de cooperação e cordenação. O produto desse passo é o *Modelo dos Estados de Agente*, cuja notação gráfica é similar à de um diagrama de estados da UML (BOOCH *et al.*, 1999).

Assim, de acordo com as responsabilidades assumidas pelo agente e com as possíveis atividades que ele exerça, são estabelecidos os estados nos quais o agente possa se encontrar durante o seu ciclo de vida. Essa tarefa é facilitada pela observação das condições referentes ao agente, sendo elas próprias que determinam as transições de estados.

A Figura 4.18 mostra o *Modelo dos Estados do Agente Minerador* da *RecomTour*, que repete as definições do respectivo modelo da ONTOWUM-DD, mediante as devidas seleções, adaptações e composições.

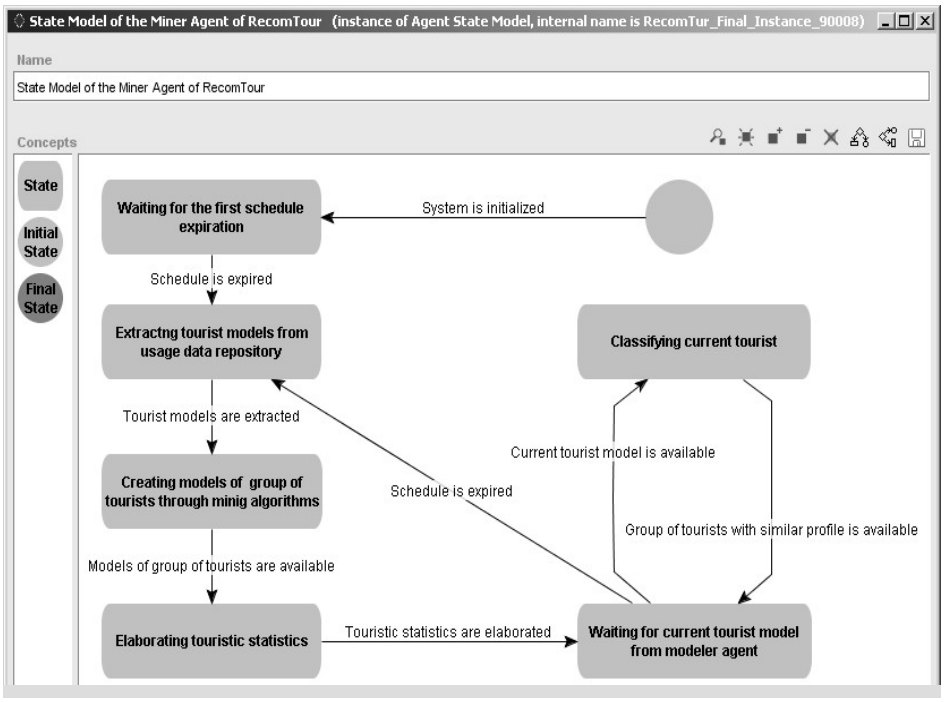


Figura 4.18: Modelo dos Estados do Agente *Minerador* da *RecomTour*

4.3.2.3 Modelagem do Conhecimento da Sociedade Multiagente

Na subfase final do Projeto da Aplicação de acordo com a MAAEM, o foco é representar o significado dos conceitos compartilhados e compreendidos por todos os agentes em sua comunicação. O produto desse passo é o *Modelo do Conhecimento da Sociedade Multiagente*, representado como uma rede semântica na qual os nós são os conceitos e os arcos são os relacionamentos entre eles.

Isso é feito através seleção, dentro dos modelos de conhecimento de sociedade multiagente dos frameworks multiagente reutilizados, de conceitos – bem como de seus subconceitos – que se adequem à comunicação estabelecida entre os agentes do problema em questão, os quais têm seus relacionamentos adaptados em razão das soluções de projeto específicas empregadas, disso restando composta a rede semântica clara e concisamente.

Normalmente, além da reutilização acima descrita, envolvendo modelos da fase de projeto da Engenharia de Domínio, constam também como conhecimentos da sociedade de agentes os principais conceitos do correspondente modelo produzido da fase de análise da própria Engenharia de Aplicações e alguns novos, mas sempre se indicando se estes conceitos equivalem, especializam ou generalizam aqueles.

A Figura 4.19 exibe o *Modelo do Conhecimento da Sociedade Multiagente* da *RecomTour*, construído com o reuso do respectivo modelo da ONTOWUM-DD.

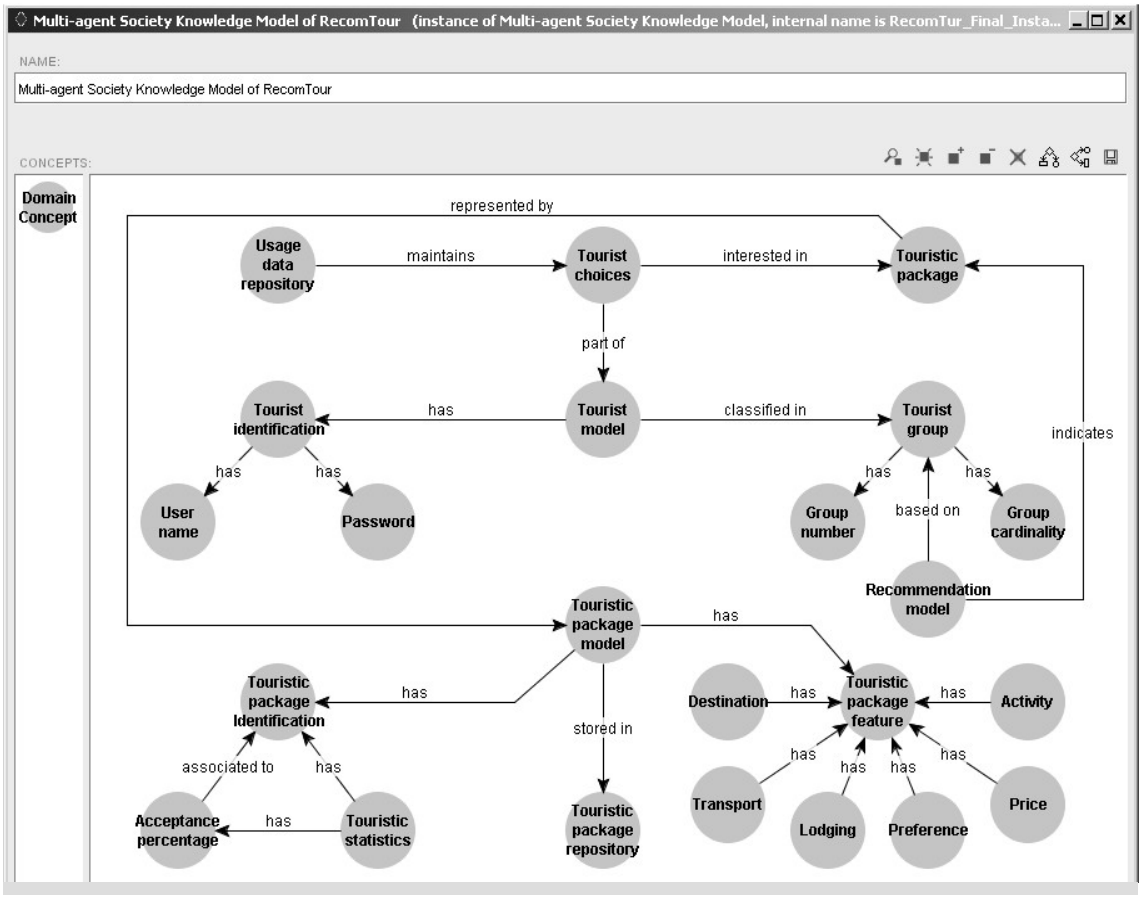


Figura 4.19: Modelo do Conhecimento da Sociedade Multiagente da *RecomTour*

4.3.3 Implementação da Aplicação

Na fase de implementação da Engenharia de Aplicações Multiagente, almeja-se produzir modelos cujos conceitos representados equivalham às principais classes utilizadas na plataforma JADE (2005) para escrever uma aplicação multiagente, o que é feito por meio da linguagem orientada a objetos Java (2005), a qual dá suporte àquela plataforna. Tal modelo, desse modo elaborado, poderia até ser submetido a um processo automático para geração de um esqueleto do código-fonte da aplicação, que deveria apenas ser complementado pelo programador.

Dessa maneira, a tarefa fundamental do desenvolvedor é mapear os conceitos presentes nos modelos do projeto da aplicação em termos de agentes, comportamentos e atos de comunicação, que são os conceitos envolvidos na tecnologia de implementação adotada.

O reuso, por conseguinte, situa-se não propriamente na elaboração dos modelos de implementação, que resultam do simples mapeamento de conceitos, mas sim na complementação do código gerado, para a qual o programador pode se valer de agentes de software anteriormente implementados, cuja seleção ocorre conforme a semelhança de comportamentos e comunicações entre si que apresentem, sendo a adaptação necessária em função dos detalhes que diferem uma aplicação de outra, as quais sempre procedem da composição das implementações de toda sua sociedade multiagente.

Nas subseções subsequentes, a descrição do *Mapeamento de Agentes do Projeto à Implementação e de Responsabilidades em Comportamentos* e do *Mapeamento de Interações entre Agentes em Atos de Comunicação* é exemplificada com o *Modelo de Implementação da Aplicação* multiagente *RecomTour*.

4.3.3.1 Mapeamento de Agentes do Projeto à Implementação e de Responsabilidades em Comportamentos

Nesse momento, conforme a MAAEM, tem-se o intuito de realizar o mapeamento de conceitos de projeto – agentes e responsabilidades, ou atividades que decomponham as últimas – para conceitos de implementação – agentes e comportamentos – correspondentes a classes do JADE. O produto desse passo é o

Modelo de Agentes e Comportamentos, representado graficamente por um diagrama organizacional de dois níveis.

Nele, tanto os agentes do JADE quanto os comportamentos do JADE possuem cada qual um atributo que indica os respectivos agentes e responsabilidades ou atividades a eles mapeados de um para um. Assim, no nível superior se encontram os agentes e no nível inferior ficam os comportamentos, sendo que há um relacionamento entre ambos indicando quais daqueles executam cada uma destas.

Além disso, existem vários tipos de comportamentos – sendo cada um apropriado a uma tarefa em particular – de acordo com a hierarquia listada a seguir (BELLIFEMINE *et al.*, 2005b):

- Comportamento: tarefa genérica,
  - Comportamento Simples: tarefa que não possui subtarefas,
    - Comportamento Atômico: tarefa simples que deve ser executada uma única vez,
      - Comportamento Despertador: tarefa simples que deve ser executada uma vez após um intervalo de tempo,
    - Comportamento Cíclico: tarefa simples que deve ser executada para sempre,
      - Comportamento Temporizado: tarefa simples que deve ser executada sempre a cada período estabelecido,
  - Comportamento Composto: tarefa que possui subtarefas,
    - Comportamento Seqüencial: tarefa complexa cujas subtarefas são desempenhadas concatenadamente até todas estarem concluídas,
    - Comportamento Paralelo: tarefa complexa cujas subtarefas são desempenhadas concorrentemente até as condições de cada uma serem encontradas,
    - Comportamento FSM: tarefa complexa cujas subtarefas correspondem às ações executadas em cada estado de uma máquina de estado finito, sendo também definidas as transições entre eles.

Em suma, durante o mapeamento, um comportamento será do tipo *simples* se estiver relacionado com uma atividade ou se a correspondente responsabilidade não for dividida em atividades, do contrário, será do tipo *composto*.

Caso seja necessário um maior detalhamento, então se deve recorrer às descrições dos subtipos de comportamento acima postas, adotando-se aquele que mais se adequar à tarefa em questão. A Figura 4.20 ilustra o *Modelo de Agentes e Comportamentos* da *RecomTour*, onde se encontram diversos exemplos de mapeamento tanto de agentes quanto de comportamentos.

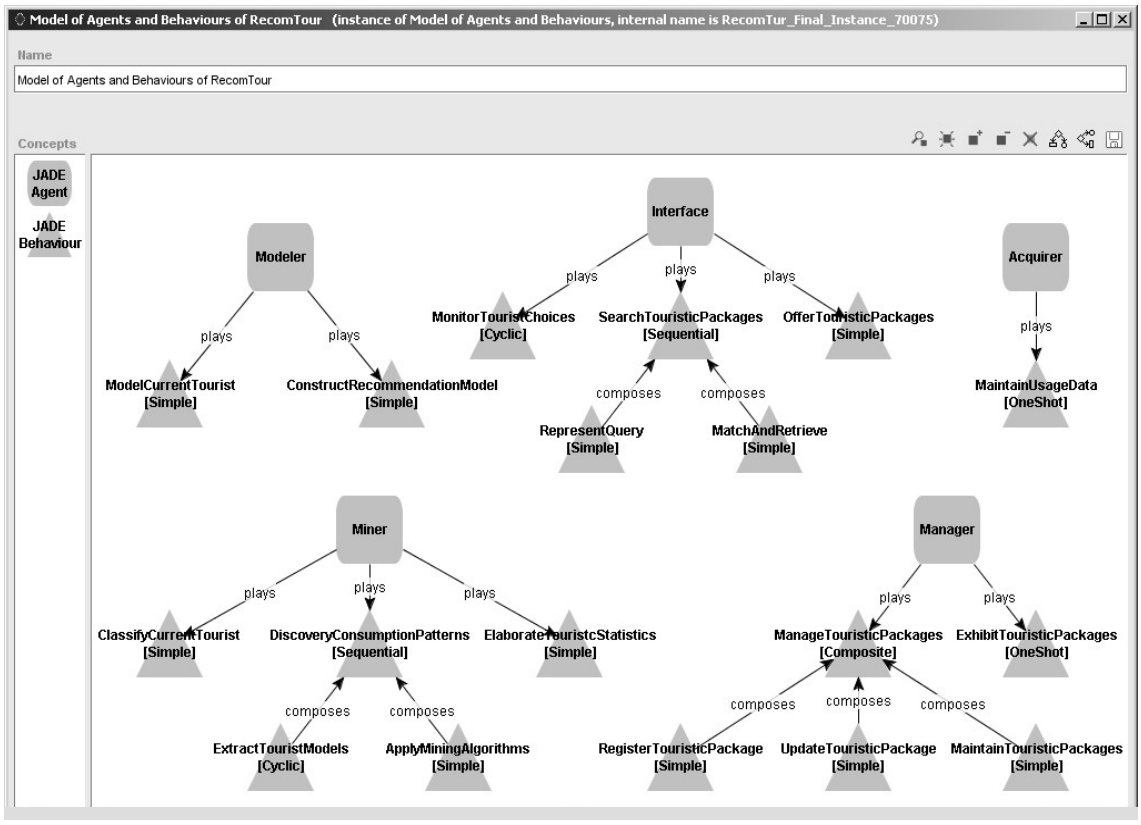


Figura 4.20: Modelo de Agentes e Comportamentos da *RecomTour*

4.3.3.2 Mapeamento de Interações entre Agentes em Atos de Comunicação

Nesse ponto, segundo a MAAEM, tem-se a meta de efetuar o mapeamento das interações entre agentes em atos de comunicação representados por mensagens da linguagem de comunicação de agentes da FIPA – *Foundation for Intelligent Physical Agents* (2005a). O produto desse passo é o *Modelo de Atos de Comunicação entre Agentes*, representado graficamente por um diagrama de vários níveis.

Inicialmente, no nível central são enfileirados os agentes anteriormente definidos. Em seguida, são estabelecidas e intercaladas nos níveis acima ou abaixo as mensagens trocadas entre tais agentes, havendo uma ordem indicada para elas. Após isso, cada mensagem é relacionada com um par de agentes, sendo um deles o remetente – através de um ato de comunicação de saída, ou seja, de onde a mensagem vem – e o outro, o destinatário – por meio de um ato de comunicação de entrada, isto é, para quem a mensagem vai.

As mensagens da FIPA-ACL – *Agent Communication Language*, de acordo com a estrutura especificada (2005b), são compostas – além de outros atributos, tais como: *remetente*, *destinatário*, *conteúdo*, *linguagem*, *ontologia*, *protocolo* etc – por uma *performativa*, definida em uma biblioteca de atos comunicativos (2005c), sendo as principais as seguintes:

- ACCEPT_PROPOSAL: aceitação de uma proposta para execução de uma ação previamente submetida;
- AGREE: concordância em executar alguma ação, possível no futuro;
- CANCEL: notificação da desistência de executar alguma ação;
- CONFIRM: confirmação de que uma proposição é verdadeira, feita por um emissor a um receptor que certamente não sabia disso;
- DISCONFIRM: confirmação de que uma proposição é falsa, feita por um emissor a um receptor que certamente pensava o contrário disso;
- FAILURE: notificação de um agente a outro acerca de uma ação tentada, mas falha;
- INFORM: informação de que uma proposição é verdadeira;
- INFORM_IF: informação – através de uma ação macro – de um agente a outro de que uma proposição é verdadeira ou falsa;
- INFORM_REF: informação – através de uma ação macro – do emissor ao receptor do objeto correspondente a um descritor;
- PROPOSE: submissão de uma proposta para a execução de certa ação, dadas certas condições;
- QUERY_IF: indagação de um agente a outro se uma proposição é verdadeira ou falsa;
- QUERY_REF: solicitação de um agente a outro pelo objeto referenciado pela expressão;

- REFUSE: negação em executar uma dada ação, sendo explicada a razão da recusa;
- REJECT_PROPOSAL: rejeição de uma proposta para execução de uma ação durante a negociação;
- REQUEST: requisição do emissor ao receptor para executar alguma ação, podendo essa ser um próprio ato de comunicação;
- REQUEST_WHEN: vontade do emissor de que o receptor execute alguma ação quando dada proposição seja verdadeira;
- REQUEST_WHENEVER: vontade do emissor de que o receptor execute alguma ação tão logo dada proposição seja verdadeira e assim por diante sempre que a proposição não seja falsa.

A escolha da performativa de cada mensagem deve levar em conta, primeiramente, a natureza e a finalidade desta, que devem ser comparadas com as descrições acima elencadas. A Figura 4.21 ilustra o *Modelo de Atos de Comunicação entre Agentes* da *RecomTour*.

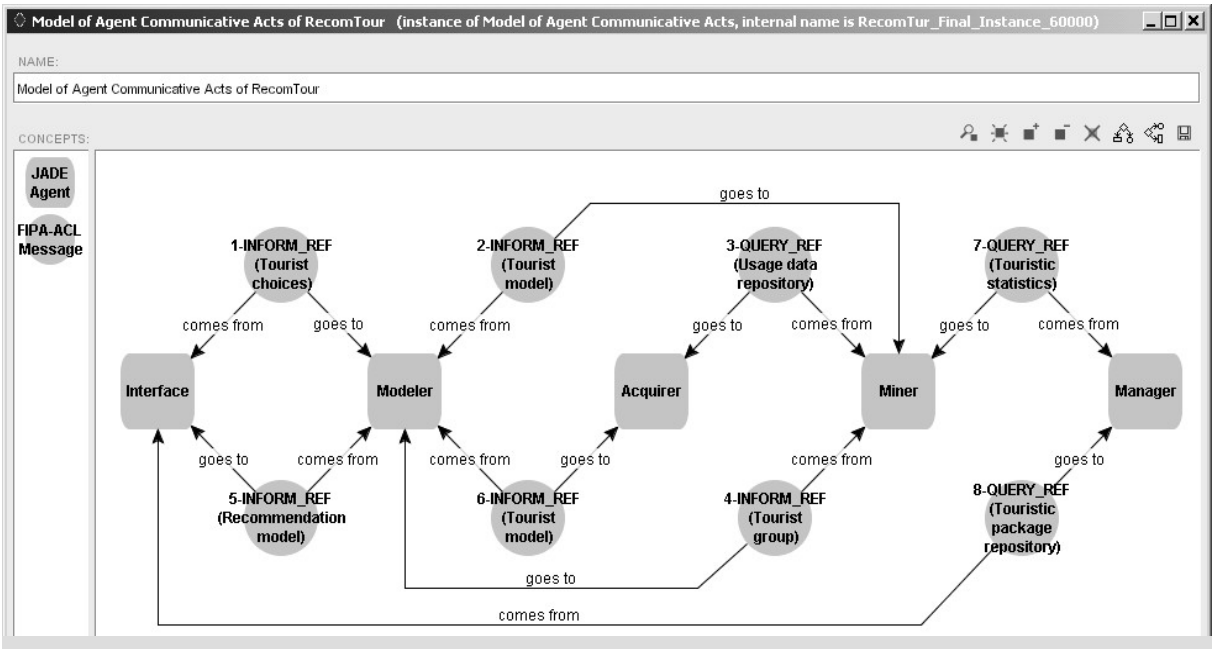


Figura 4.21: Modelo de Atos de Comunicação entre Agentes da *RecomTour*

4.4 Considerações finais

Este capítulo contemplou a parte principal da presente dissertação, posto que nele foi descrita a metodologia MAAEM para o desenvolvimento de aplicações

orientadas a agentes com base no reuso, bem como a ontologia ONTORMAS, que representa o conhecimento da metodologia e serve como repositório para os componentes reutilizáveis e também como ferramenta de suporte ao desenvolvimento.

Assim, foram descritos e ilustrados todas as fases, subfases, passos, produtos e subprodutos da metodologia, cada qual nas hipóteses tanto com quanto sem reutilização.

Algo que merece especial destaque é a estreita relação que as metodologias MAAEM e MADEM guardam entre si, sendo tal expressa principalmente por meio da unificação do conhecimento de ambas em uma única ontologia, a ONTORMAS, como já dito anteriormente.

Isso propiciou a contemplação integral do ciclo de desenvolvimento de software orientado a agentes e baseado no reuso, sendo esta, por sinal, a maior contribuição do presente trabalho.

Os próximos capítulos trazem duas experiências na construção de aplicações multiagente específicas que têm o objetivo de avaliar a MAAEM e a ONTORMAS nos casos sem e com reuso, respectivamente.

5 ESTUDO DE CASO COM REUSO: *RecomTour* – Uma Aplicação Multiagente para a Recomendação de Pacotes Turísticos através da Mineração de Uso na Web e da Filtragem Colaborativa

5.1 Considerações iniciais

Para demonstrar a aplicabilidade da metodologia MAAEM e da ontologia ONTORMAS ao desenvolvimento de aplicações multiagente quando é possível a reutilização de artefatos de software provenientes da Engenharia de Domínio Multiagente, foi idealizado o presente estudo de caso, o qual consiste na construção da *RecomTour* – Uma Aplicação Multiagente para a Recomendação de Pacotes Turísticos através da Mineração de Uso na Web e da Filtragem Colaborativa.

Essa experiência aborda todas as fases do desenvolvimento, iniciando-se pela análise e especificação de requisitos da aplicação, passando pelo projeto arquitetural da sociedade multiagente e pelo projeto detalhado de cada agente, cujo produto é a arquitetura da aplicação, a qual, encerrando o processo, é convertida em um projeto dependente de uma determinada tecnologia de implementação orientada a agentes.

5.1.1 Contexto da aplicação: o domínio turístico

É notório o crescimento cada vez mais acelerado da atividade turística. E devido a ele, a oferta de produtos e serviços tem se diversificado de forma a dificultar a escolha pelo turista de pacotes turísticos (BENI, 1998). Por outro lado, como o leque de opções se torna cada vez mais extenso, o turista tem se tornado mais exigente, forçando as agências de turismo a oferecerem serviços cada vez mais personalizados e a buscarem meios eficazes de divulgação.

Apoiando-se no exposto acima, busca-se especificar uma aplicação que satisfaça os turistas quanto às suas necessidades de serviços turísticos personalizados e que possa auxiliá-los no processo de tomada de decisão sobre o que visitar, onde, quando, como, porque etc, também devendo servir às agências de turismo no tocante à divulgação dos serviços oferecidos e de sua oferta personalizada. Deste modo, é previsto que a aplicação proposta disponha de duas interfaces distintas: uma para aqueles e outra para estas.

Assim, a interface do turista possibilitará a criação e a manutenção do perfis dos usuários, com base nos quais se realiza a filtragem de pacotes turísticos para uma possível recomendação aos turistas, de acordo com o perfil de cada um, sendo tais operações efetuadas através da mineração de uso e da filtragem colaborativa, respectivamente.

Já a interface das agências de turismo será a forma de alimentação da aplicação com pacotes turísticos que potencialmente serão recomendados aos turistas, uma vez que é através dela que as agências os cadastrarão, servindo ela para a administração de pacotes turísticos.

5.1.2 Requisitos da aplicação

A aplicação *RecomTour* objetiva a recomendação personalizada de pacotes turísticos previamente cadastrados no sistema pelas agências de turismo. Em razão disso, tem-se um auxílio ao turista no processo de tomada de decisão sobre que pacote turístico adquirir dentro de um conjunto de opções e às agências de turismo como um meio de divulgação de seus pacotes.

Desse modo, a *RecomTour* tem três requisitos principais. O primeiro diz respeito aos turistas e cuida da forma como são adquiridas, implicitamente, suas preferências e como é modelado, com base nessas preferências, o seu perfil. Isso para que se possa oferecer recomendações personalizadas de pacotes turísticos.

O segundo requisito é relacionado à atividade de modelagem da recomendação e envolve a utilização das informações capturadas e o modelo de usuário para a classificação dos turistas em grupos de turistas com perfis semelhantes e, baseado nesses grupos, a composição de uma interface personalizada com as recomendações dos pacotes turísticos.

O terceiro requisito está relacionado às agências de turismo e engloba a administração de pacotes turísticos, através das ações de cadastro e alteração, além da exibição de estatísticas relativas a aceitação dos pacotes turísticos.

5.1.3 Fundamentos da aplicação

Como já esboçado, a *RecomTour* tem por finalidade efetuar a recomendação de pacotes turísticos selecionados de acordo com as necessidades

de cada usuário, por meio de uma interface personalizada. Portanto, ela pertence a uma família de aplicações para a recomendação personalizada de informações. E, representando uma solução para essa classe de problemas, dispõe da *ONTOWUM* – um Modelo de Domínio, um Framework Multiagente e Agentes de Software para a Mineração de Uso na Web (MARINHO, 2005).

Assim, é na *ONTOWUM* que se buscará a devida fundamentação para a modelagem da *RecomTour*, sendo também a partir dela que se procederá à seleção, adaptação e composição dos modelos a serem reutilizados, quando possível, nas fases e passos do desenvolvimento da aplicação em questão, no sentido de contemplar todas as funcionalidades desejadas.

Os tópicos que constituem a base da especificação da *ONTOWUM*, e conseqüentemente da *RecomTour*, são a mineração de uso na Web e a filtragem colaborativa, nos termos brevemente descritos a seguir. Demais considerações sobre ambos os tópicos poderão ser encontradas ao longo do capítulo.

5.1.3.1 Mineração de uso na Web

A mineração de uso da Web (MUW) é a área que se dedica à atividade de investigação de seqüências de visitas feitas por usuários a páginas da Web, visando não só reconstituir os passos seguidos pelos usuários, mas principalmente descobrir quais padrões podem ser interessantes para o domínio da aplicação, através da aplicação de técnicas de mineração de dados aos dados de uso à Web (BECKER, VANZIN, 2004) (CHEN *et al.*, 1998).

Os dados de uso estão concentrados principalmente nos *logs* dos servidores Web, que armazenam as interações dos usuários com as páginas visitadas, mas também podem ser encontrados nas próprias estruturas dos sites Web – informações sobre as referências e vínculos entre as páginas – ou obtidos dos usuários a partir do uso de programas *CGI*, *cookies* e outros mecanismos. Desse modo, a mineração de uso da Web é apropriada para analisar sistematicamente o comportamento passado dos usuários. (BOULLOSA, 2002).

Um outro aspecto da mineração de uso da Web é que ela serve como apoio na tomada de decisões. A saída de um processo é geralmente um conjunto de modelos de dados que representam conhecimento implícito sobre padrões de uso dos usuários da Web. Esses modelos são então analisados por especialistas, tais

como analistas de mercado que buscam novas formas de aumentar os lucros ou administradores de sites que buscam aperfeiçoar a estrutura do site de forma a melhorar a experiência de navegação dos visitantes.

A mineração de uso da Web também fornece conhecimento operacional que pode servir como insumo para sistemas de personalização da Web. Os dados de uso representam as interações dos usuários com os sites da Web. A mineração de uso da Web fornece uma abordagem para a coleta e pré-processamento desses dados, gerando modelos que representam, dentre outras coisas, o comportamento e interesses dos usuários. Esses modelos podem então ser utilizados por um sistema para a personalização automática da Web, ou seja, sem a intervenção de especialistas humanos (PIERRAKOS *et al.*, 2003).

O processo da mineração de uso obedece as seguintes etapas:

- aquisição de dados: os dados de uso, que podem ser provenientes de várias fontes, são reunidos e o seu conteúdo e estrutura identificados. Esses dados podem ser coletados tanto de servidores Web, de máquinas clientes ou de fontes intermediárias tais como servidores *proxy*;
- pré-processamento dos dados: os dados são limpos de ruídos e inconsistências e são integrados de forma a serem utilizados como entrada para a próxima etapa, correspondente à descoberta de padrões. Isso envolve basicamente as tarefas de filtragem dos dados, identificação de usuários e identificação de sessões dos usuários;
- descoberta de padrões: os padrões são descobertos através da aplicação de técnicas estatísticas e de aprendizagem de máquina aos dados, tais como: agrupamento, classificação, descoberta de regras de associação e descoberta de padrões seqüenciais. Geralmente o conhecimento requerido pelos sistemas de personalização corresponde aos padrões de navegação ou interesses dos usuários inferidos nessa etapa;
- pós-processamento do conhecimento: o conhecimento extraído é avaliado e apresentado em um formato inteligível pelos humanos, como por exemplo, através de relatórios ou ferramentas de visualização. Para fins de personalização automática, o conhecimento descoberto é diretamente incorporado a um módulo de personalização.

5.1.3.2 Filtragem colaborativa

A filtragem colaborativa é um processo que se baseia na similaridade entre usuários para gerar recomendação de produtos, ou seja, utiliza a avaliação dada por outros usuários para predizer se um determinado item será ou não relevante para um usuário particular (QUEIROZ, 2002) (TORRES, 2004) (ADOMAVICUS, TUZHILIN, 2005). O âmago dos sistemas colaborativos é troca de experiências entre indivíduos que possuem interesses comuns, visto que os itens são filtrados baseado nas avaliações feitas pelos usuários (CAZELA, REATEGUI, 2004).

A técnica da filtragem colaborativa pode ser descrita através dos seguintes passos (QUEIROZ, 2002) (CAZELA, REATEGUI, 2004):

- representação dos dados de entrada: o usuário exprime suas preferências, a principio preenchendo um questionário e, depois, atribuindo notas a itens propostos pelo sistema, o que indica os interesses do usuário e são armazenados em seu perfil, e, caso o sistema não peça ao usuário que atribua notas aos itens consumidos, pode-se, de maneira alternativa, considerar ‘1’ caso o usuário tenha consumido o produto e ‘0’, caso contrário;
- mensuração da similaridade entre os usuários: o sistema compara as avaliações dos usuários para itens consumidos, a fim de descobrir quão similares são essas avaliações e, assim, encontrar usuários que estejam mais “próximos”;
- formação da vizinhança: uma vez que a similaridade entre os usuários foi calculada, é necessário que se agrupem os usuários similares (“vizinhos”), devendo-se notar que cada usuário terá uma vizinhança diferente, o que garante a personalização das recomendações;
- geração da recomendação: o sistema utiliza as informações da vizinhança para fazer a recomendação de itens ao usuário, sendo que tal é feita com base numa predição calculada, a qual representa o valor que o usuário supostamente daria ao item, sendo recomendados aqueles que apresentem os maiores valores de predição.

As técnicas de filtragem colaborativa possuem as seguintes vantagens (QUEIROZ, 2002):

- independência de conteúdo: como a filtragem colaborativa baseia-se na avaliação dada pelos usuários, qualquer tipo de item pode ser analisado – um filme, um livro, uma música etc – e as recomendações são enviadas para os usuários com base apenas nas avaliações dos outros usuários, sem que haja a necessidade da análise do conteúdo dos itens;
- habilidade de recomendar produtos segundo características como qualidade e gostos: na filtragem colaborativa conta-se com a capacidade de se recomendar um item utilizando-se análises subjetivas que os usuários tiveram;
- possibilidade de produzir recomendações inesperadas e boas: tendo em vista a forma como são geradas as recomendações, a filtragem colaborativa pode gerar recomendações inesperadas, porém boas, isso ocorrendo devido ao fato dela representar os hábitos de consumo dos usuários.

Por outro lado, os sistemas baseados em filtragem colaborativa também apresentam algumas limitações (ADOMAVICUS, TUZHILIN, 2005) (CAZELA, REATEGUI, 2004):

- recomendação de novos itens: um item não pode ser recomendado até que tenha sido avaliado por um número mínimo de usuários, uma vez que o sistema não possui dados suficientes para atribuir um peso a ele, caracterizando o chamado *cold start*;
- usuário “ovelha negra”: neste caso, o sistema padece de desempenho, pois não há vizinhos próximos o suficiente para que a recomendação alcance os interesses do usuário, uma vez que as recomendações são baseadas em usuários que apresentam interesses similares aos dele;
- esparcialidade: quando o número de usuários é demasiadamente pequeno em relação ao número de informações no sistema, o número de avaliações é muito pequeno e há um grande risco de tornarem-se muito esparsas, isso acontecendo também com usuários cujas

preferências sejam incomuns em relação ao restante da população de usuários, tornando-se muito pobres suas recomendações.

5.2 Análise da aplicação *RecomTour*

Na fase de Análise da Aplicação, conforme estabelecido pela metodologia MAAEM, a preocupação do desenvolvedor é produzir a *Especificação da Aplicação* (Figura 5.1), a qual é composta por quatro subprodutos: o *Modelo de Conceitos*; o *Modelo de Objetivos*; o *Modelo de Papéis*; e os *Modelos de Interações entre Papéis*. Nas próximas subseções, é compassadamente descrita a elaboração desses modelos na análise da RecomTour.

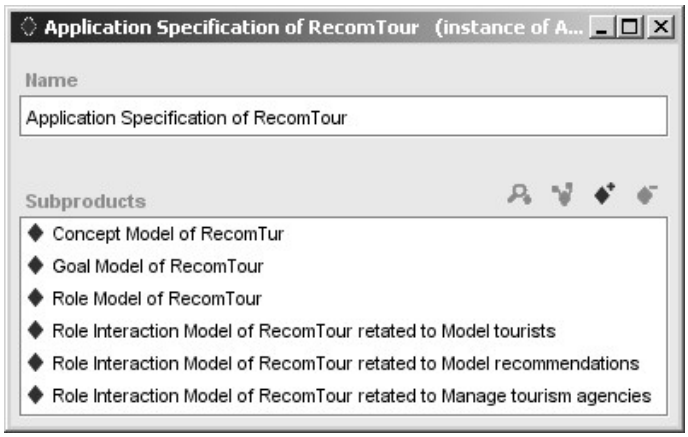


Figura 5.1: Especificação da Aplicação *RecomTour*

A modelagem nessa fase é realizada com o reuso do modelo de domínio da ONTOWUM-DM (APÊNDICE A), que apresenta os requisitos correspondentes à família de aplicações à qual pertence a *RecomTour*.

5.2.1 Modelagem de conceitos

A Figura 5.2 adiante ilustra o *Modelo de Conceitos* da RecomTour, cuja principal utilidade no processo de desenvolvimento da aplicação é servir como um primeiro momento para se levantar as idéias mais importantes relativas à aplicação, o que é expresso na forma de conceitos semanticamente relacionados.

Em razão da reutilização acima anunciada, na modelagem da RecomTour aparecem tanto conceitos originários do respectivo modelo da ONTOWUM-DM quanto outros particulares da recomendação de pacotes turísticos.

Assim, por apresentarem variabilidade do tipo *mandatória*, os seguintes conceitos existentes na ONTOWUM-DM foram obrigatoriamente reutilizados: *sistema recomendador colaborativo baseado em mineração de uso*, *comportamento de navegação do usuário*, *páginas da Web*, *modelo de usuário*, *perfil de usuário*, *recomendações personalizadas*, *modelo de adaptação*, *modelo de grupo de usuários com perfis similares*.

De tal reuso, em função da especialização para o domínio turístico, surgiram então, nessa ordem de correspondência, os conceitos a seguir: *sistema de recomendação de pacotes turísticos*, *comportamento de consumo do turista*, *pacote turístico*, *modelo do turista*, *perfil do turista*, *pacotes turísticos recomendados*, *modelo de recomendação personalizada*, *modelos de grupo de turistas*.

Duas exceções são relativas aos conceitos *usuário*, que foi especializado como *turista* e como *agência de turismo*, e *repositório de dados de uso*, reusado da forma que estava. Todos os demais conceitos, não relacionados acima, são novos: *identificação do turista*, *sessão de uso* e *dados de uso*.

Já quanto aos relacionamentos, tem-se como conceito de partida o sistema de recomendação de pacotes turísticos, o qual visa *prover* pacotes turísticos recomendados, o que faz *capturando* o comportamento de consumo do turista, *construindo e atualizando* os modelos do turista, *minerando* as sessões de uso e *construindo* os modelos de recomendação personalizada.

O comportamento de consumo do turista, por sua vez, *tem como alvo* pacotes turísticos, os quais *são criados e anunciados* por agências de turismo. Por outro lado, o perfil do turista, que *é representado pelo* modelo do turista, *tem comos partes* a identificação de turista e os dados de uso, os quais *são armazenados em* um repositório de dados de uso, *de onde são extraídas as* sessões de uso, *em que se baseiam* modelos de grupo de turistas, *em um dos quais é classificado* cada modelo do turista. Por fim, o turista, *tendo* um perfil, *apresenta* um comportamento de consumo e *a ele são oferecidos* pacotes turísticos recomendados.

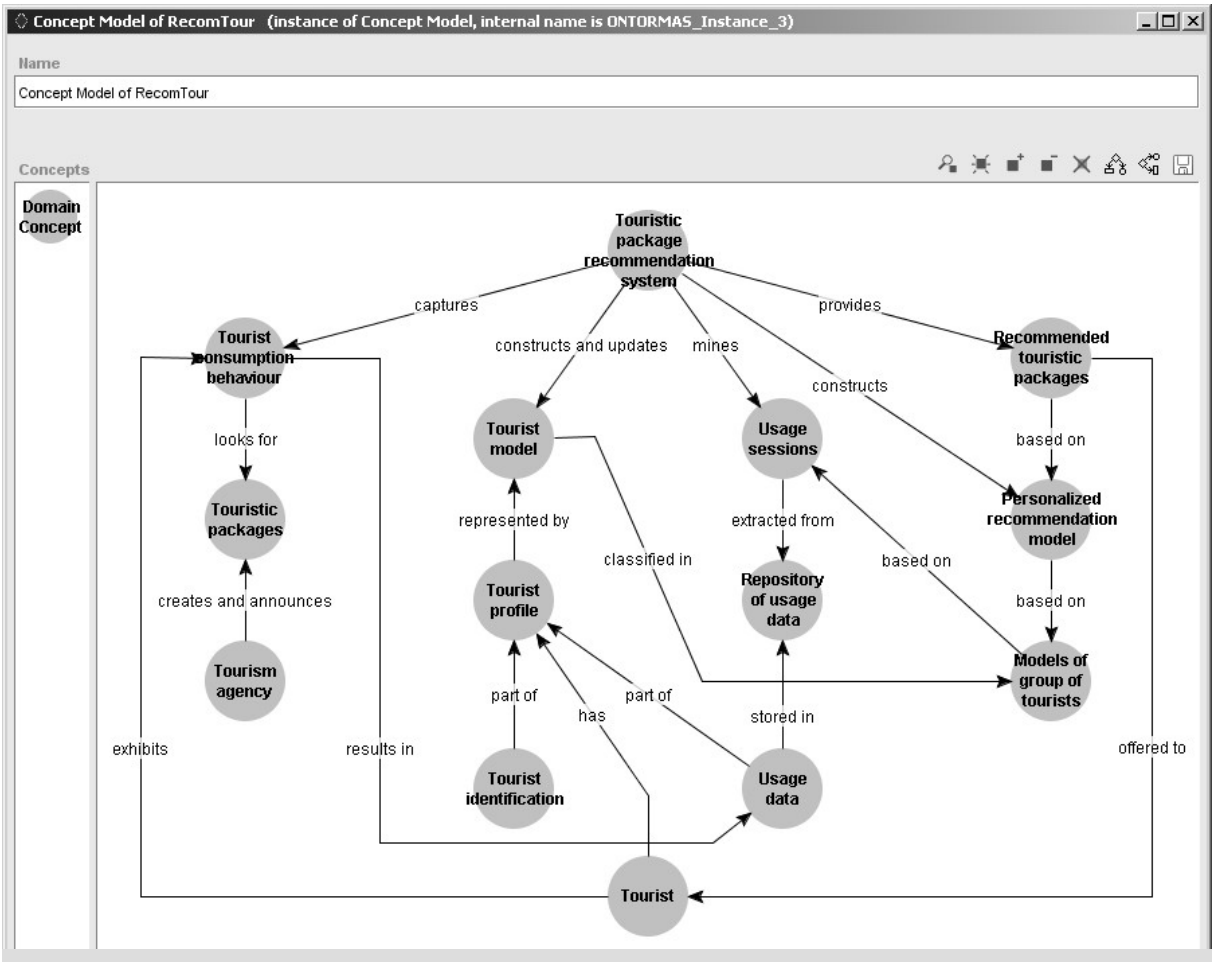


Figura 5.2: Modelo de Conceitos da *RecomTour*

É o entendimento inicial de todos esses conceitos e relacionamentos – os quais, diga-se, não constituem um rol exaustivo, mas apenas exemplificativo – que embasa todo o restante da modelagem, como se poderá constatar nos demais passos a seguir.

5.2.2 Modelagem de objetivos

A Figura 5.3 mostra o *Modelo de Objetivos* da RecomTour, trazendo uma hierarquia de objetivos – geral e específicos – e de responsabilidades, que correspondem às funcionalidades esperadas da aplicação, sendo que uns permitem o sucesso do outros. Constam também desse modelo entidades externas que interagem destinatárias da aplicação ou com ela envolvidas.

Uma vez que a aplicação RecomTour faz reuso de elementos da ONTOWUM-DM, nesta modelagem, da mesma forma que na anterior, partiu-se da

seleção, adaptação e composição de objetivos, responsabilidades e entidades externas presentes no correspondente modelo.

Assim, antes de qualquer coisa, procedeu-se ao estudo das variabilidades dos conceitos de modelagem reutilizáveis. O objetivo geral de *prover recomendações colaborativas através de mineração de uso* e a entidade externa *usuário* – por serem ambos fixos – foram reusados mediante especialização em termos do Turismo, resultando, quanto àquele, em *prover recomendações personalizadas de pacotes turísticos* e, quanto a esta, em *turista* e *agência de turismo*.

Por outro lado, dos três objetivos específicos, que conduzem ao objeto geral e trocam informações com as entidades externas, dois deles – por serem variáveis, porém mandatórios – provieram da reutilização de *modelar adaptação através de mineração de uso* e *modelar usuários através de mineração de uso*, sendo especializados, respectivamente, em *modelar recomendações* e em *modelar turistas*. Já o outro, *gerenciar agências de turismo*, é novo e específico do presente desenvolvimento.

No tocante às responsabilidades, as quais permitem alcançar os correspondentes objetivos, todas aquelas encontradas no modelo de objetivos reutilizado foram aproveitadas, visto possuírem variabilidade mandatória, sendo elas: *construção do modelo de adaptação*, *adaptação da interface da Web*, *classificação do usuário corrente*, *descoberta de padrões de uso*, *modelagem do usuário corrente*, *monitoração do usuário* e *manutenção de dados de uso*.

Dessa forma, à execeção da última, as restantes sofreram especializações em decorrência do domínio enfocado, originando, respectivamente, as seguintes: *construção do modelo de recomendação*, *oferta de pacotes turísticos*, *classificação do turista corrente*, *descoberta de padrões de consumo*, *modelagem do turista corrente*, *monitoração de turistas*. E, além delas, algumas novas e particulares foram introduzidas: *busca de pacotes turísticos*, *gerenciamento de pacotes turísticos*, *elaboração de estatísticas turísticas* e *exibição de pacotes turísticos*.

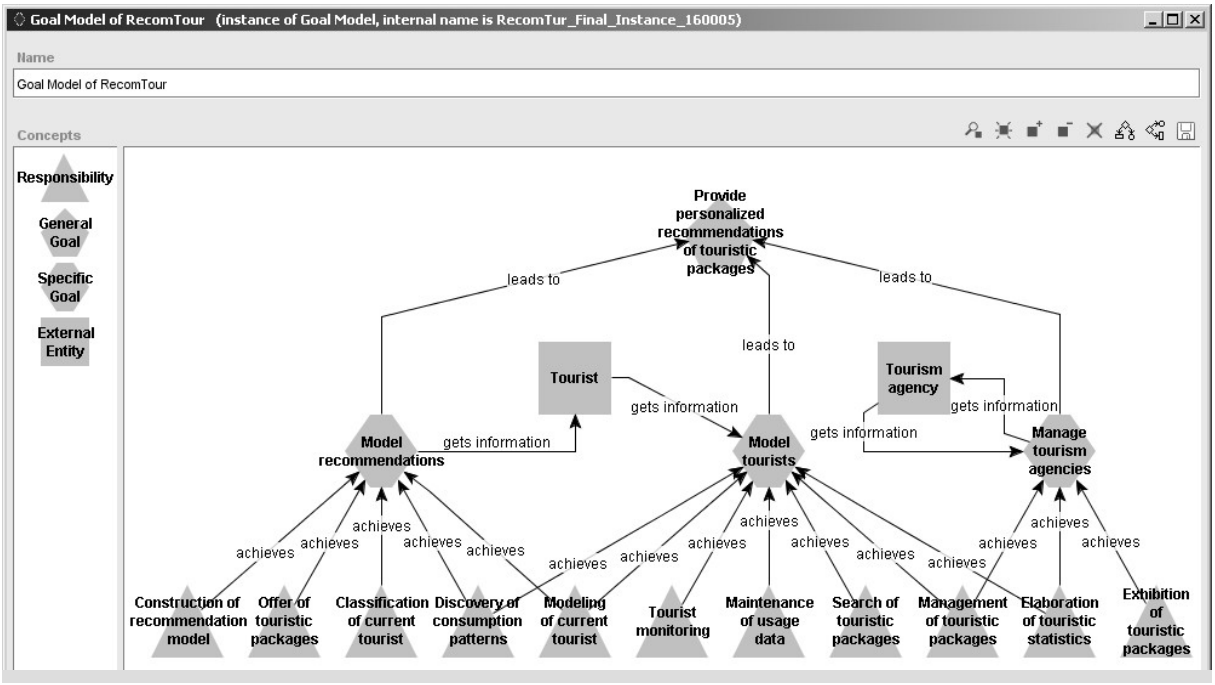


Figura 5.3: Modelo de Objetivos da *RecomTour*

Depois de estabelecidos os objetivos da aplicação, bem como as correspondentes responsabilidades, pode-se, então, prosseguir com o desenvolvimento, posto que é a partir destas que se derivará os respectivos papéis no passo posterior.

5.2.3 Modelagem de papéis

A Figura 5.4, a Figura 5.5, a Figura 5.6, a Figura 5.7, a Figura 5.8 e a Figura 5.9 exibem as partes do *Modelo de Papéis* da RecomTour, o qual apresenta como principal função a atribuição das responsabilidades anteriormente identificadas a papéis, sendo que estes, no seu efetivo desempenho, envolvem o uso e a produção de determinados conhecimentos, o atendimento de pré-condições e de pós-condições, além de requererem destrezas específicas para esse fim. Pode haver casos em que uma responsabilidade seja abrangente demais, devendo ela ser repartida em duas ou mais atividades, ressaltando-se que estas têm a mesma natureza daquela.

Preliminarmente, empreendeu-se a tentativa de reutilização dos papéis, entidades externas, conhecimentos e condições presentes na ONTOWUM-DM, os quais, posto que não apresentam variabilidades, sendo, portanto, conceitos fixos, são reutilizados acompanhando o reuso das respectivas responsabilidades. Já as

atividades e destrezas, por serem variáveis, são reusadas conforme sejam mandatórias, alternativas ou opcionais, da forma anteriormente ensinada.

Assim, quanto aos papéis, – a título de exemplo, visto que em relação aos demais conceitos de modelagem se procedeu de maneira bastante similar – selecionou-se os seguintes, para serem em seguida compostos no modelo: *monitor de usuário*, *modelador de usuário*, *classificador*, *personalizador*, *interface do usuário*, *aquisitor de dados* e *minerador de uso*. E, ao serem adaptados para o domínio em questão por meio da especialização, com execeção dos dois últimos, resultaram, nesta ordem, os papéis: *monitor de turista*, *modelador de turista*, *classificador de turistas*, *recomendador de pacotes* e *interface do turista*. Ademais, foram definidos exclusivamente para a RecomTour o *elaborador de estatísticas*, o *gerenciador de pacotes*, o *recuperador de pacotes* e a *interface da agência de turismo*.

A primeira responsabilidade é a *monitoração de turistas*, sendo encargo do papel *monitor de turista*. Este papel usa como conhecimento as *opções por pacotes turísticos* feitas pelo *turista* e também os *pacotes turísticos recuperados* para ele, produzindo *informações sobre escolhas do turista*, sendo requeridas para tal, *técnicas para captura de informações sobre escolhas de usuários*. A pré-condição para isto é que um *turista tenha entrado no sistema* e a pós-condição é que as *informações sobre escolhas do turista tenham sido capturadas*.

Já a segunda responsabilidade, que é a *modelagem do turista corrente*, atribuída ao papel *modelador de turista*, que utiliza o conhecimento gerado pelo papel anterior, bem como tem a pós-condição deste como sua pré-condição, e produz o *modelo do turista corrente*, resultando em que o *modelo do turista corrente esteja disponível* como pós-condição, através de *técnicas para construção de modelos de usuário*.

A terceira responsabilidade, consistente na *manutenção de dados de uso*, tem como responsável o papel *aquisitor de dados*, o qual usa o produto de seu precedente, atendida a pré-condição de que o *turista tenha saído do sistema*, para originar um *repositório de dados de uso*, com base em *técnicas para manutenção de dados de uso*, o que da margem à pós-condição de que o *repositório de dados de uso esteja atualizado*.

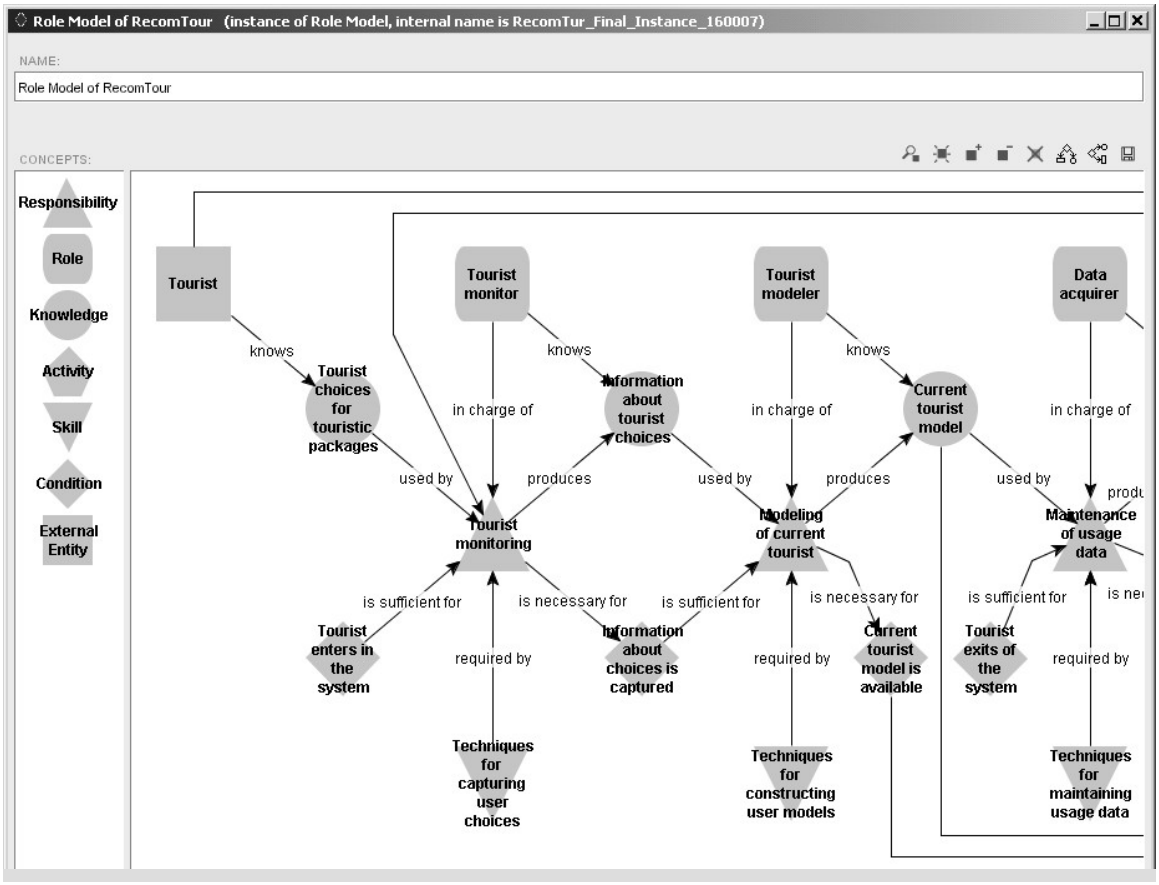


Figura 5.4: Primeira parte do Modelo de Papéis da *RecomTour*

A quarta responsabilidade é a *descoberta de padrões de consumo*, cujo encarregado é o papel *minerador de uso* e que, devido a sua complexidade, é exercida através de duas atividades. Uma delas é a *extração de modelos de turista*, que faz uso do conhecimento produzido pelo papel precedente, assim como tem uma pré-condição coincidente com a pós-condição do papel antecessor e outra de que um *tempo de espera tenha transcorrido*, produzindo os *modelos de turista* baseado em *técnicas para extração de modelos de turista* e tendo que os *modelos de turista estejam extraídos* como pós-condição.

A outra atividade, seqüencial em relação àquela, que é a *aplicação de algoritmos de mineração*, guiada por *técnicas para mineração de dados de uso*, usa tais modelos para produzir *modelos de grupo de turistas*, quando ocorrida sua pré-condição que é a pós-condição da atividade antecessora desta, o que repercute na pós-condição de que *modelos de grupo de turistas estejam diponíveis*.

A quinta responsabilidade, que se refere à *classificação do turista corrente*, é incumbência do papel *classificador de turistas*, que identifica o *grupo similar ao perfil do turista corrente* a partir do produto advindo do papel antecedente

e do *modelo do turista corrente*, o que faz por meio de *técnicas para classificação de usuários em grupos*, se atendida a pré-condição de que o *modelo do turista corrente esteja disponível*, dando lugar à pós-condição de que o *grupo com perfil similar ao do turista corrente esteja disponível*.

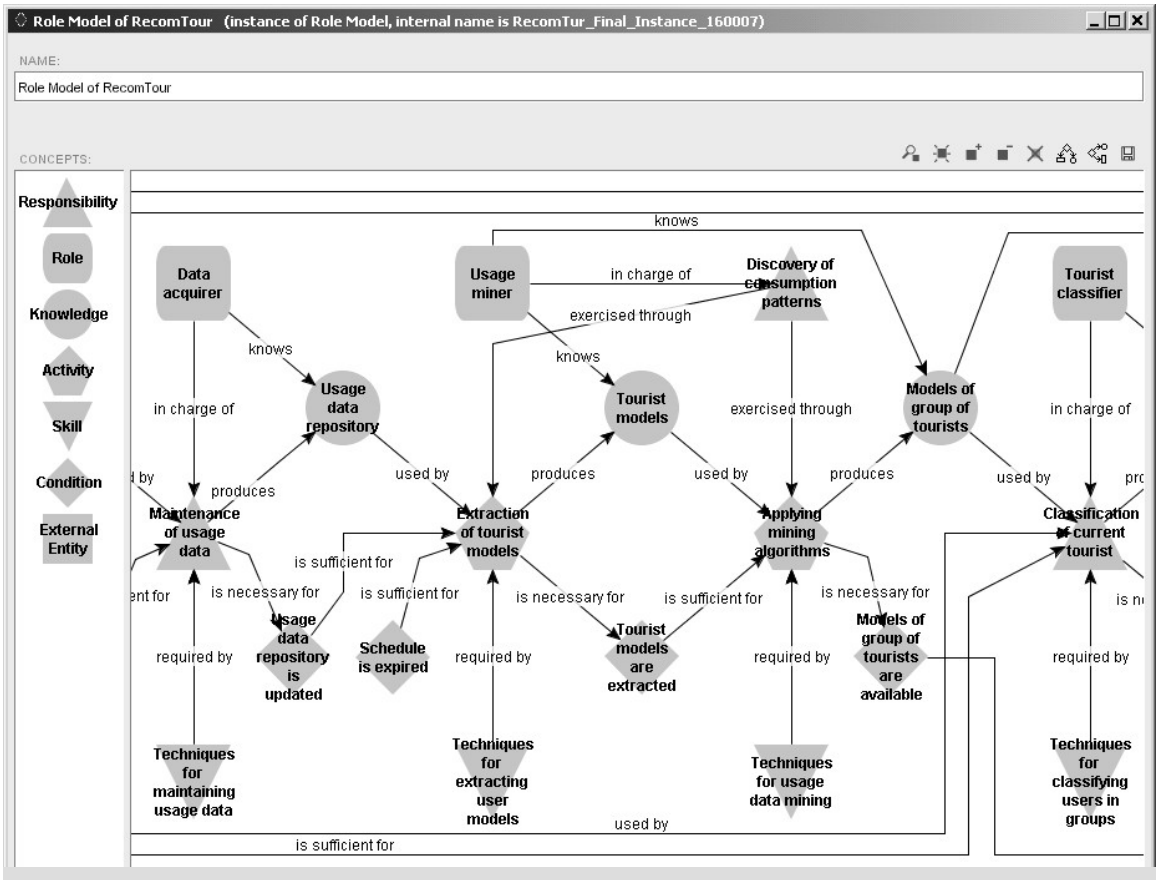


Figura 5.5: Segunda parte do Modelo de Papéis da *RecomTour*

A sexta responsabilidade, que diz respeito à *construção do modelo de recomendação*, tem o papel *recomendador de pacotes* como encarregado, que produz o *modelo de recomendação* usando o produto resultante da responsabilidade anterior, nos termos de *técnicas para construção de modelos de adaptação*, sendo a pré-condição para tanto a pós-condição precedente, e tendo sua pós-condição que o *modelo de recomendação esteja pronto*.

A sétima responsabilidade, relativa à *oferta de pacotes turísticos*, incumbe ao papel *interface do turista*, que usa os resultados da recomendação e produz *pacotes turísticos recomendados*, tendo como pré-condição a pós-condição que a antecede e como pós-condição que as *recomendações tenham sido entregues*, e sendo necessárias, no caso, *técnicas para entrega de informações personalizadas*.

Cabe destacar que todas as responsabilidades e demais conceitos de modelagem acima expostos foram selecionados na ONTOWUM-DM visando a reutilização, sendo que alguns deles precisaram passar por adaptações antes da composição final.

A oitava responsabilidade é a *elaboração de estatísticas turísticas*, sendo encargo do papel *elaborador de estatísticas*. Ele usa os *modelos de grupo de turistas*, produzindo *estatísticas sobre escolhas de pacotes turísticos*, sendo para isso necessárias *técnicas para coleta e processamento de dados estatísticos*. A pré-condição para isto é que *modelos de grupo de turistas estejam diponíveis* e a pós-condição é que as *estatísticas turísticas tenham sido elaboradas*.

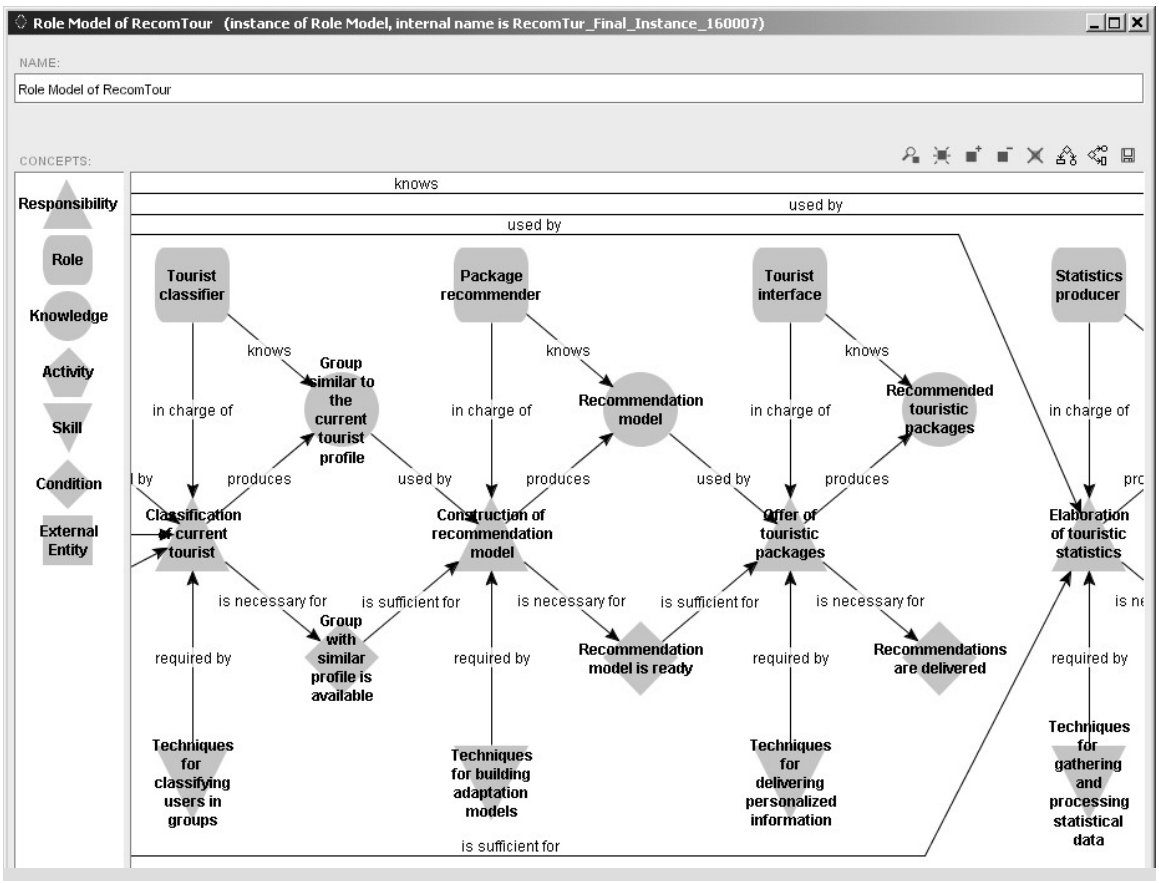


Figura 5.6: Terceira parte do Modelo de Papéis da *RecomTour*

A nona responsabilidade é o *gerenciamento de pacotes turísticos*, pela qual responde o papel *gerenciador de pacotes*, sendo exercida através de três atividades em razão de sua complexidade. A primeira é o *cadastro de pacotes turísticos*, que usa *informações sobre pacotes turísticos* fornecidas pelas *agências de turismo* e, sob pré-condição de que um *novo pacote turístico tenha sido*

especificado, produz o respectivo *modelo de pacote turístico*, valendo-se de *técnicas para representação de pacotes turísticos*, tendo a pós-condição de que um *modelo de pacote turístico esteja disponível*.

A segunda atividade, alternativa em relação à anterior, que é a *atualização de pacotes turísticos*, também guiada por *técnicas para representação de pacotes turísticos* e igualmente produzindo *modelos de pacotes turísticos*, usando tanto as *informações sobre um certo pacote turístico* quanto as referentes a ele armazenadas em um *repositório de pacotes turísticos*, por ocasião da pré-condição de que um *antigo pacote turístico tenha sido alterado*, o que também causa na pós-condição e que um *modelo de pacote turístico esteja disponível*.

Já a terceira atividade, seqüencial em relação a ambas, que é a *manutenção de pacotes turísticos*, orientada por *técnicas para manutenção de pacotes turísticos*, toma *modelos de pacotes turísticos* e os armazena em um *repositório de pacotes turísticos*, tendo a pós-condição das demais como pré-condição e que um *pacote turístico esteja armazenado* como pós-condição.

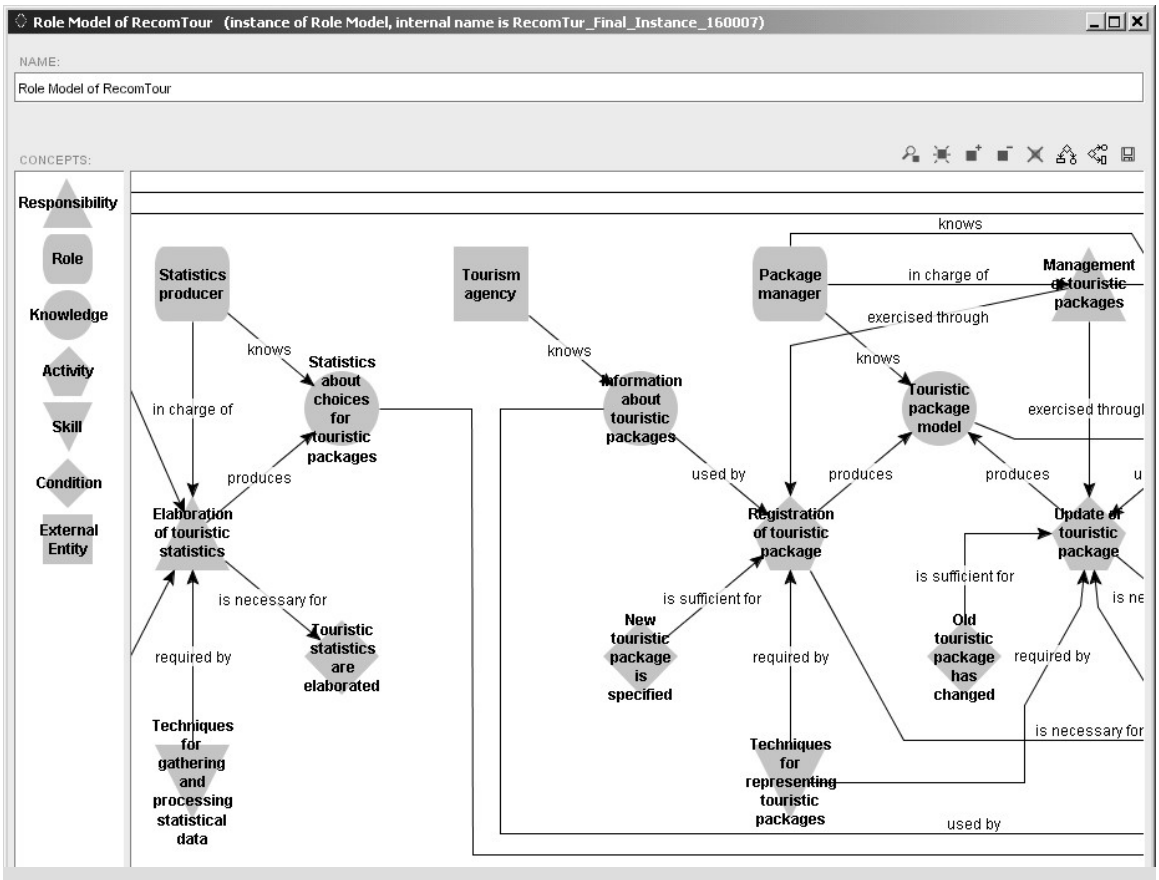


Figura 5.7: Quarta parte do Modelo de Papéis da *RecomTour*

A décima responsabilidade é a *busca de pacotes turísticos*, tendo como responsável o papel *recuperador de pacotes*, e sendo exercida através de duas atividades por conta de sua complexidade. Uma delas é a *representação da consulta*, a qual faz uso de uma *necessidade de pacote turístico* apresentada por um *turista* e produz a *representação da consulta* por intermédio de *técnicas para o recebimento de consultas*, diante da pré-condição de que uma *consulta por pacotes turísticos tenha sido especificada* e se verificando a pós-condição de que a *consulta esteja representada*.

A outra atividade, seqüencial em relação àquela, que é a *recuperação de pacotes turísticos*, mediante *técnicas para recuperação de informações*, usa tais representações cruzadas com o conteúdo do *repositório de pacotes turísticos* para produzir *pacotes turísticos recuperados*, quando ocorrida sua pré-condição que é a pós-condição da atividade que precede esta, proporcionando a pós-condição de que *pacotes turísticos tenham sido recuperados*.

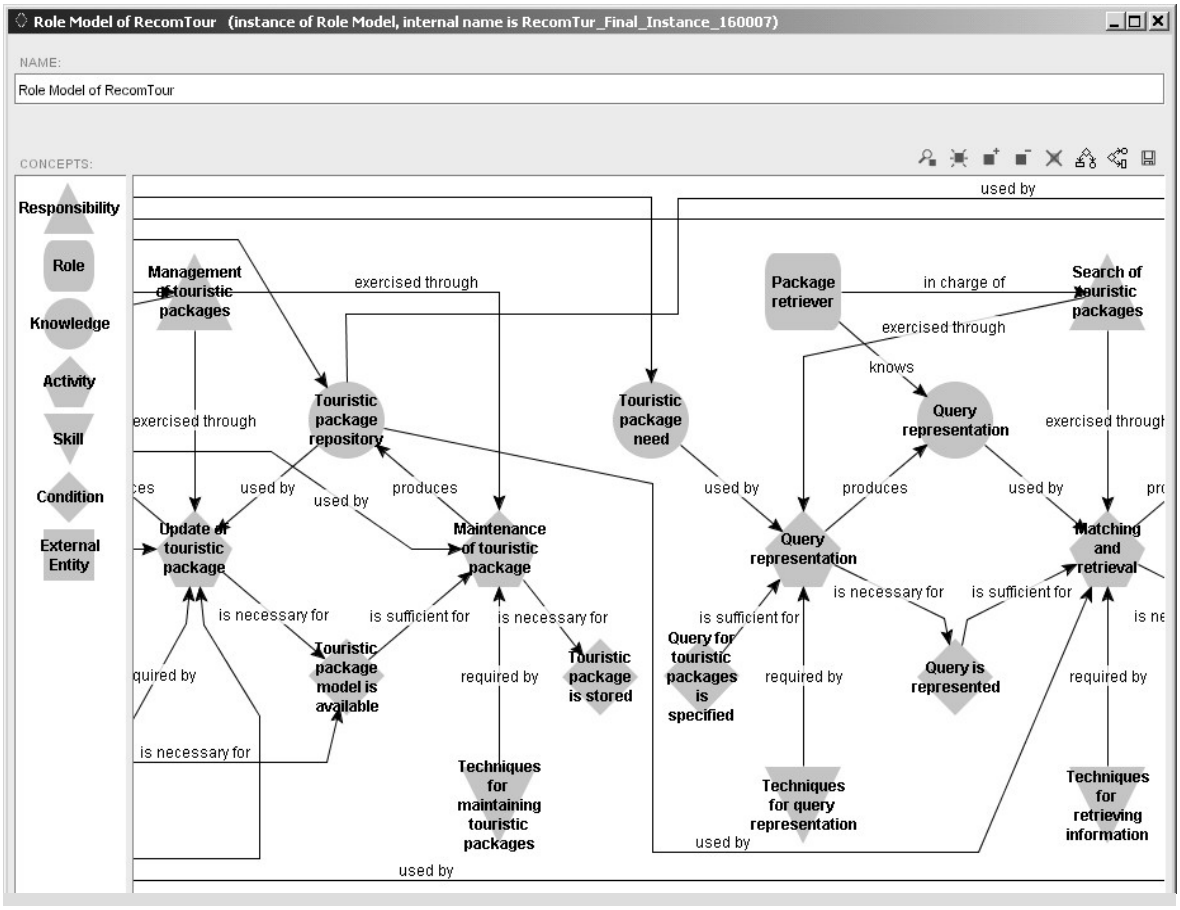


Figura 5.8: Quinta parte do Modelo de Papéis da *RecomTour*

A última responsabilidade, relativa à *exibição de pacotes turísticos*, sendo incumbência do papel *interface da agência de turismo*, a qual usa tanto as *estatísticas sobre escolhas de pacotes turísticos* quanto as informações constantes do *repositório de pacotes turísticos* para produzir a *lista de pacotes turísticos com estatísticas* para cada *agência de turismo*, tendo como pré-condição que a *agência de turismo esteja conectada* ao sistema e como pós-condição que os *pacotes turísticos tenham sido listados*, isso através de *técnicas para listagem de informações e estatísticas*.

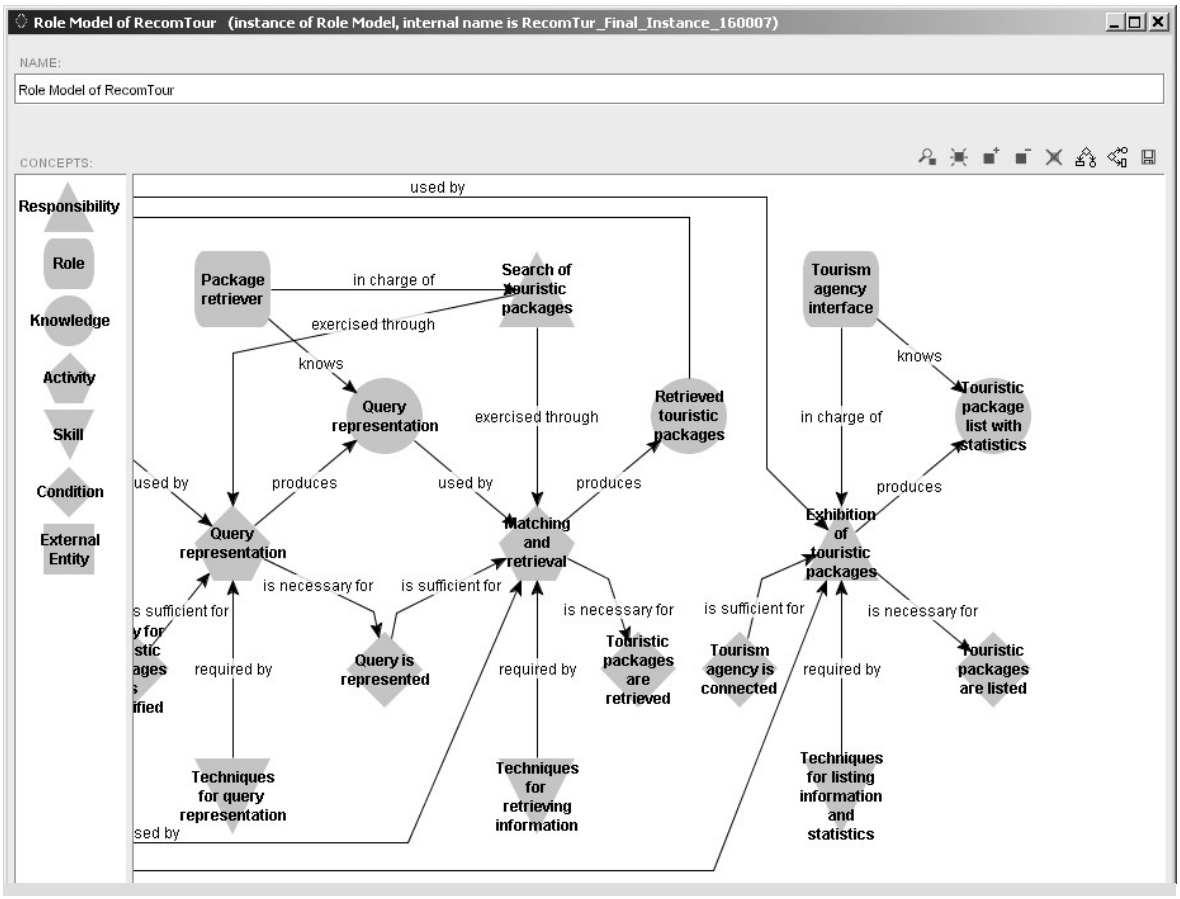


Figura 5.9: Sexta parte do Modelo de Papéis da *RecomTour*

Os papéis, juntamente com os demais conceitos de modelagem que aparecem acima, representam uma visão estática da aplicação, sendo esse apenas um de seus aspectos. Assim, a sua dinâmica é definida através de interações entre papéis como se verá a seguir.

5.2.4 Modelagem de interações entre papéis

A Figura 5.10, a Figura 5.11 e a Figura 5.12 trazem os *Modelos de Interações entre Papéis* da RecomTour, relativos aos objetivos específicos *modelar turistas*, *modelar recomendações* e *gerenciar agências de turismo*, respectivamente, sendo um para cada devido à complexidade do problema, os quais ilustram de maneira seqüencial como os papéis e as entidades externas da aplicação interagem, através da troca de mensagens invocando responsabilidades ou atividades, bem como indicando que conhecimentos são passados como parâmetros.

De início, quanto ao reuso, tal foi possível no tocante a dois dos três modelos de interações entre papéis pertinentes à RecomTour, tendo sido um de maneira total e outro parcial, mas em ambas as situações com as especializações devidas às que se fez nos respectivos papéis e entidades externas, de acordo com as peculiaridades do domínio turístico, e somente em virtude disso e do que já se descreveu anteriormente, sendo desnecessário mostrar as interações reutilizadas.

Assim, em relação à modelagem de turistas, de oito interações, as quatro iniciais foram selecionadas, adaptadas e compostas a partir da ONTOWUM-DM, enquanto que não pôde haver a reutilização quanto à outra parte delas, visto que acontecerem entre participantes inéditos naquele modelo de domínio.

A primeira interação é "*1 : monitoração de turistas (opções do turista por pacotes turísticos)*", ocorrendo entre o turista e monitor de turista, e consistindo no início do funcionamento da aplicação, já que para haver a recomendações é necessário que um turista esteja utilizando a aplicação.

Em seguida, a segunda interação é "*2 : modelagem do turista corrente (informações sobre as escolhas de pacotes turísticos)*", sendo através da qual o papel de modelador de turista solicita ao modelador de turista que elabore os devidos modelos, a serem usados posteriormente.

Logo após, a terceira interação é "*3 : manutenção de dados de uso (modelo do turista corrente)*", a qual parte do papel modelador de turista para o aquisitor de dados, que armazena tais modelos para oportuno emprego.

Depois disso, a quarta interação é "4 *: descoberta de padrões de consumo (repositório de dados de uso)*", acontecendo entre os papéis aquisitor de dados e minerador de uso, quando este pede àquele que forneça os dados de uso que serão minerados.

Prosseguindo, a quinta interação é "*5 : elaboração de estatísticas turísticas (modelos de grupo de turistas)*", ocorrendo quando o papel elaborador de estatísticas recebe do minerador de uso as informações envolvidas em seu trabalho.

Por outro lado, a sexta interação é "*6 : representação da consulta (necessidade de pacote turístico)*", consistindo justamente na solicitação que o turista faz ao papel recuperador de pacotes para que recupere certos pacotes.

Como resultado, a sétima interação é "*7 : comparação e recuperação (repositório de pacotes turísticos)*", dando-se entre o papel gerenciador de pacotes e o recuperador de pacotes, e compondo os requisitos para a recuperação, juntamente com a representação da consulta formulada.

Finalmente, a oitava interação é "8 *: visualização (pacotes turísticos recuperados)*", sendo concluída com o recebimento pelo turista dos pacotes turísticos buscados na aplicação.

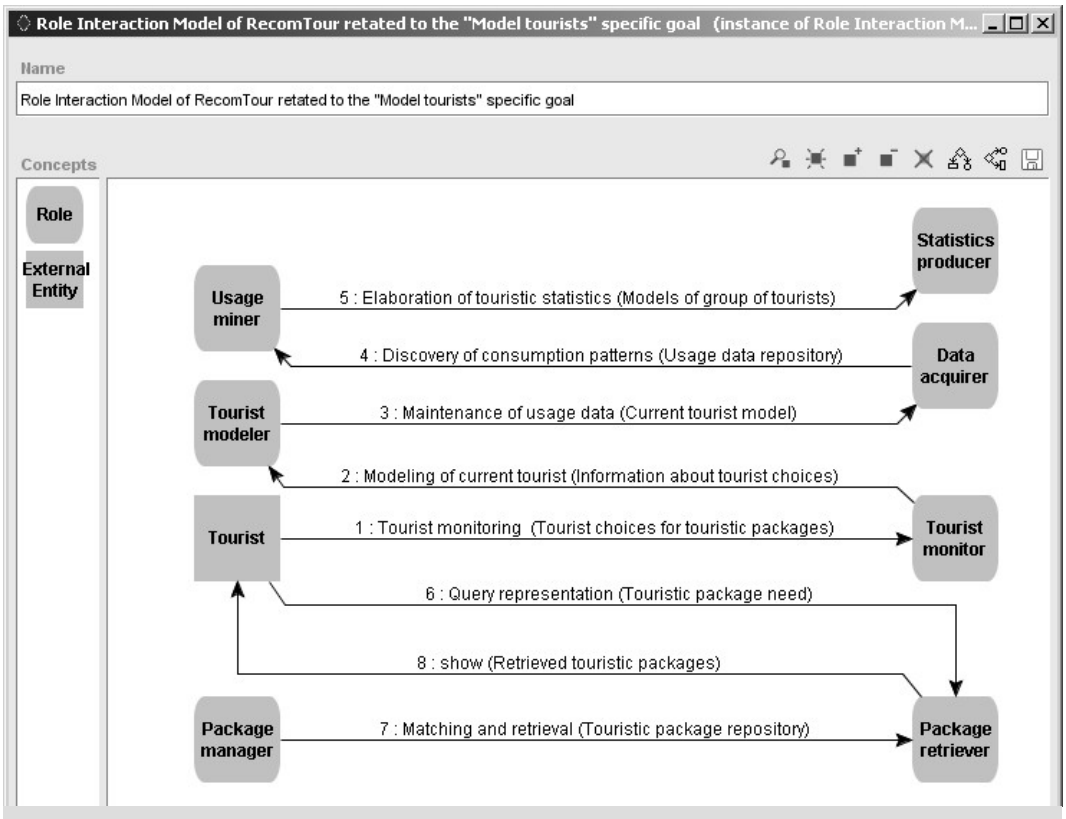


Figura 5.10: Modelo de Interações entre Papéis da *RecomTour* referente ao objetivo específico *modelar turistas*

Já com referência à modelagem de recomendações, existiu o reuso de todas as interações presentes no correspondente modelo da ONTOWUM-DM, tendo havido apenas a seleção, adaptação e composição necessárias.

A primeira interação é "*1 : classificação do turista corrente (modelo do turista corrente)*", ocorrendo entre o modelador de turista e classificador de turista, diz respeito ao começo do processo de recomendação, posto que é com base em seu modelo que pacotes turísticos podem ser recomendados a um dado turista.

Em seguida, a segunda interação, "*2 : classificação do turista corrente (modelos de grupo de turistas)*", sendo através da qual o papel de minerador de uso complementa o material usado pelo classificador de turista na sua atividade fim.

Logo após, a terceira interação, "*3 : construção do modelo de recomendação (grupo similar ao perfil do turista corrente)*", parte do papel classificador de turista para o recomendador de pacotes, possibilitando a definição dos pacotes turísticos recomendáveis a cada perfil de turista.

Como resultado, a quarta interação, "4 *: oferta de pacotes turísticos (modelo de recomendação)*", acontece entre os papéis recomendador de pacotes e interface do turista, que procede à recomendação de pacotes ao turista corrente.

Por fim, a quinta interação, "5 *: visualização (pacotes turísticos recomendados)*", encerra-se com a entrega ao turista dos pacotes turísticos sugeridos pela aplicação.

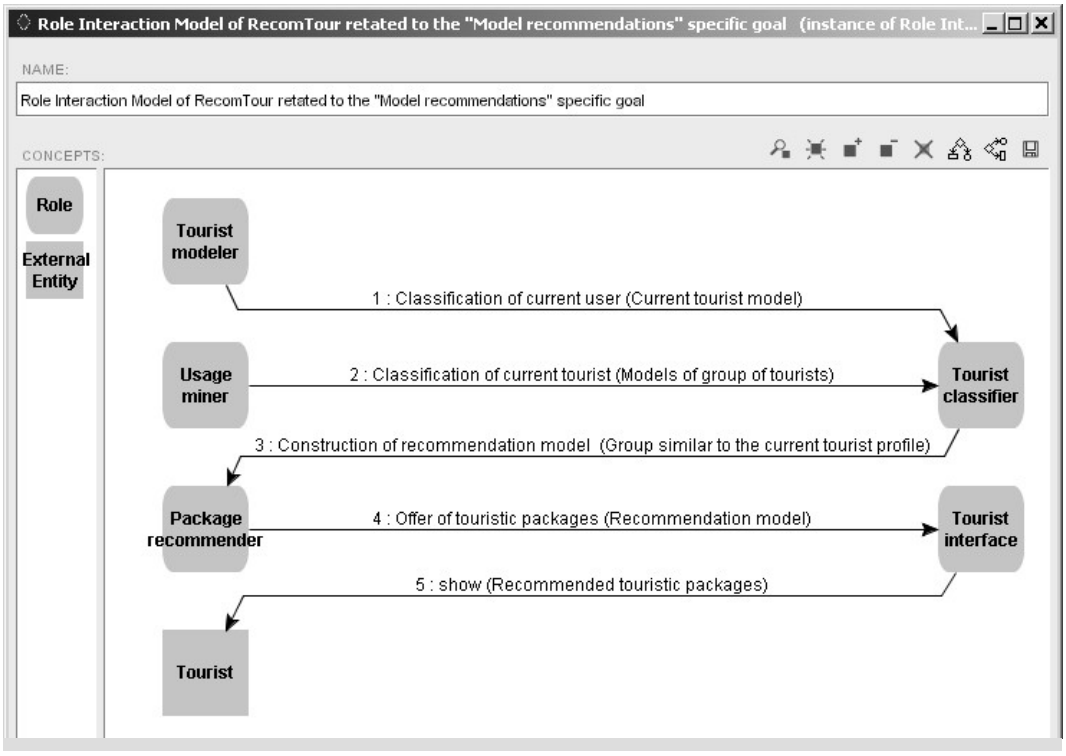


Figura 5.11: Modelo de Interações entre Papéis da *RecomTour* referente ao objetivo específico *modelar recomendações*

Atinente ao gerenciamento de agências de turismo, nenhuma reutilização foi possível, posto que a entidade externa e os três papéis envolvidos não se encontram descritos na ONTOWU-DM, o que motivou um desenvolvimento específico para o caso.

A primeira interação, "*1 : cadastro de pacotes turísticos (informações sobre pacotes turísticos)*", ocorrendo entre a agência de turismo e o gerenciador de pacotes, consiste na alimentação da aplicação de pacotes dos quais os turistas poderão dispor através de escolhas, recuperações e recomendações.

Em seguida, a segunda interação, "*2 : exibição de pacotes turísticos (repositório de pacotes turísticos)*", indo do gerenciador de pacotes para a interface da agência de turismo, que fica aguardando as estatísticas turísticas.

Logo após, a terceira interação, "*3 : exibição de pacotes turísticos (estatísticas sobre escolhas de pacotes turísticos)*", parte do elaborador de estatísticas também para a interface da agência de turismo, a fim de agregar-se ao conteúdo previamente detido por ela.

Como resultado, a quarta interação, "4 *: visualização (lista de pacotes turísticos com estatísticas)*", acaba com a listagem para a agência de turismo de seus pacotes turísticos as respectivas estatísticas sobre eles.

Enfim, a quinta interação, "5 *: atualização de pacotes turísticos (informações sobre pacotes turísticos)*", existe apenas em complemento à primeira, sendo havida entres as mesmas entidades.

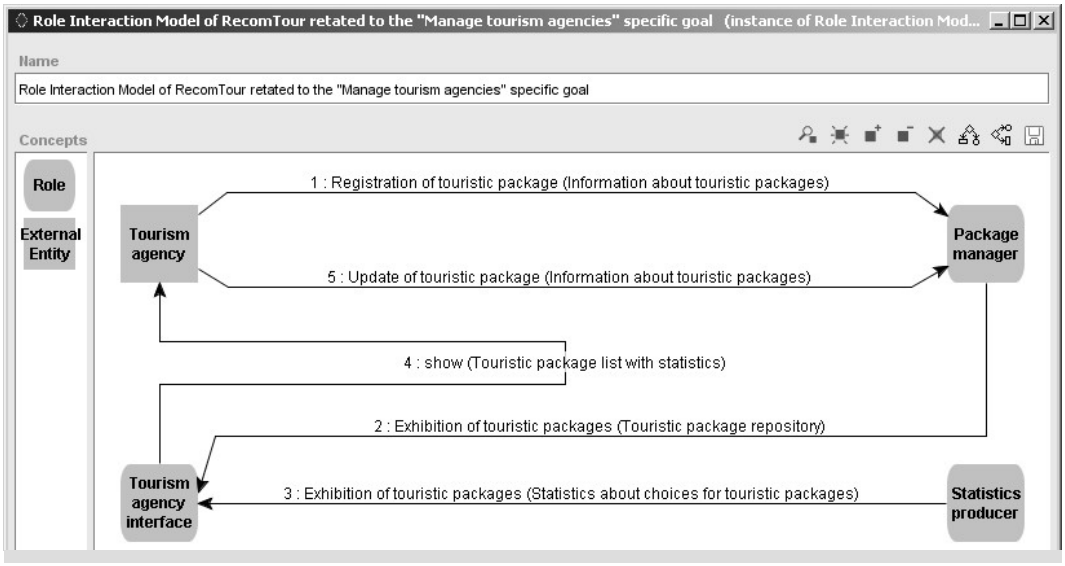


Figura 5.12: Modelo de Interações entre Papéis da *RecomTour* referente ao objetivo específico *gerenciar agências de turismo*

Estabelecidas as interações entre papéis, encerra-se a fase de análise da aplicação. Os próximos passos, pertencentes ao projeto, são todos baseados no refinamento dos modelos aqui elaborados.

5.3 Projeto da aplicação *RecomTour*

Na fase de Projeto da Aplicação, segundo o passo a passo prescrito pela metodologia MAAEM, o desenvolvedor busca construir a *Arquitetura da Aplicação* (Figura 5.13), a qual é composta por três tipos de subprodutos: o *Modelo da Arquitetura*; o *Modelo do Agente*, sendo um para cada agente; e o *Modelo do Conhecimento da Sociedade Multiagente*. Nas próximas subseções, é cuidadosamente mostrada a elaboração desses modelos no projeto da RecomTour.

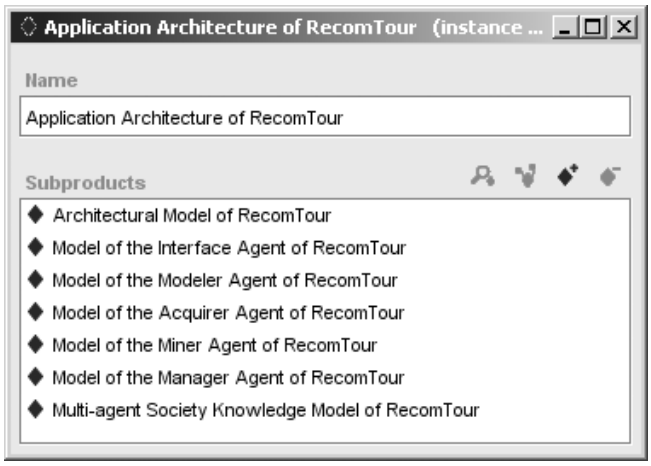


Figura 5.13: Arquitetura da Aplicação *RecomTour*

A modelagem nessa fase é realizada com o reuso do framework multiagente da ONTOWUM-DD (APÊNDICE A), que apresenta soluções de projeto arquitetural e detalhado para a família de aplicações à qual pertence a *RecomTour*.

5.3.1 Projeto da arquitetura

Nesta primeira subfase, é produzido o *Modelo da Arquitetura* da RecomTour (Figura 5.14), que inclui o *Modelo da Sociedade Multiagente*, o *Modelo das Interações entre Agentes* e o *Modelo dos Mecanismos de Cooperação e Coordenação*.

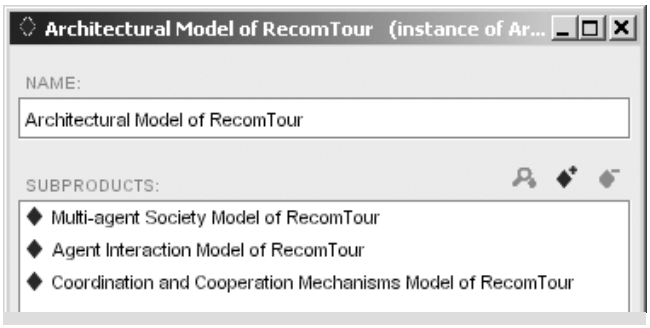


Figura 5.14: Modelo da Arquitetura da *RecomTour*

5.3.1.1 Modelagem da sociedade multiagente

A Figura 5.15, a Figura 5.16, a Figura 5.17, a Figura 5.18, a Figura 5.19 e a Figura 5.20 ilustram as partes do *Modelo da Sociedade Multiagente* da RecomTour, que se diferencia do Modelo de Papéis por adotar como abstração principal os agentes, que são capazes de assumir vários papéis durante sua execução. Assim, faze-se o mapeamento de um ou mais destes para um daqueles, de acordo com a afinidade que haja entre as respectivas responsabilidades. Além disso, as destrezas requeridas são refinadas, tornando-se soluções concretas para os problemas dos quais cada agente é encarregado.

Assim, os papéis de *interface do turista, monitor de turista*, *recuperador de pacotes* e *recomendador de pacotes* passam a ser desempenhados pelo agente *Interfaceador*, visto que interagem com a mesma entidade externa, o *Turista*. De forma semelhante, os papéis *minerador de uso*, *classificador de turistas* e *elaborador de estatísticas* serão desempenhados pelo agente *Minerador* e os papéis *gerenciador de pacotes* e *interface da agência de turismo* pelo agente *Gerenciador*. Quanto aos papéis *modelador de turista* e *aquisitor de dados*, ficam convertidos, respectivamente, nos agentes *Modelador* e *Aquisitor*.

No projeto, para cada responsabilidade ou atividade, é selecionada uma destreza específica dentre aquelas que foram levantadas genericamente na fase de análise. Assim, em vez de várias técnicas que se prestam em potencial ao fim esperado, é definido um modo particular de se realizar a ação atribuída ao agente.

Com isso, para a *monitoração de turistas*, foi adotada a *captura implícita de escolhas de turistas*. Esta captura é feita no momento em que o turista interage com o sistema, selecionando ou não um pacote turístico. Neste levantamento, colhe-

se também a seqüência em que os pacotes são escolhidos pelo turista. Ao final, tem-se uma lista com a seqüência das escolhas do turista corrente, o que representa seu comportamento de consumo.

Com base nesse comportamento de consumo, parte-se para a *modelagem do turista corrente*. A forma escolhida para representar os modelos de turista foi o *modelo de matrizes de características* (SHAHABI, BANAEI-KASHANI, 2003). Neste modelo, características indicam as informações contidas numa sessão do turista. Para este caso, as características consideradas são a seleção de pacotes e a seqüência dessa seleção. Assim, para cada característica da sessão do turista é criada uma matriz, as quais, em conjunto, constituem o modelo do turista corrente.

Para a *manutenção de dados de uso*, são usados *grafos semânticos em RDF* (LASSILA, SWICK, 1999). Quando uma sessão de turista é encerrada, o agente *Aquisitor* atribui a ela um identificador e a grava em um arquivo *RDF*. Este formato de arquivo é usado de forma a facilitar a análise desses dados no futuro.

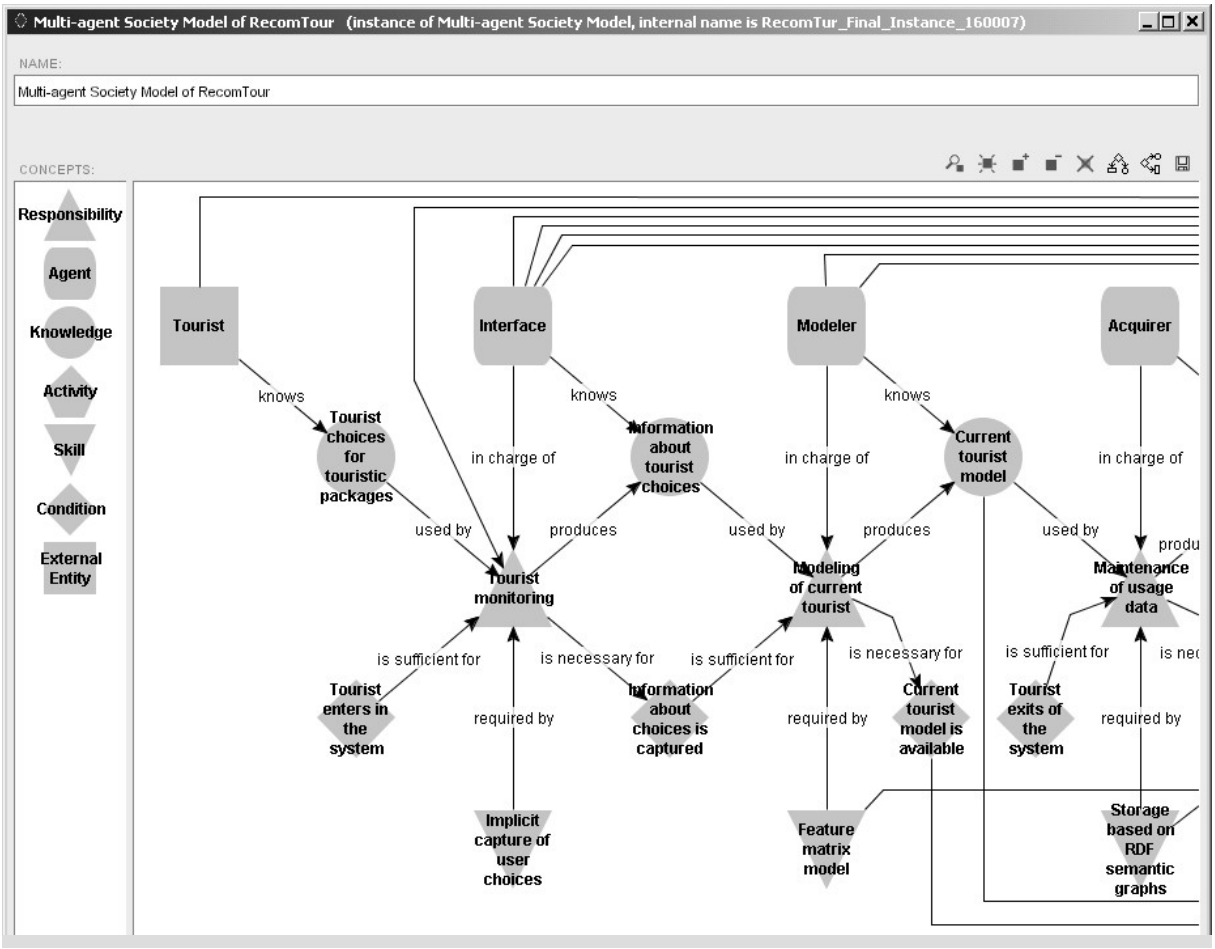


Figura 5.15: Primeira parte do Modelo da Sociedade Multiagente da *RecomTour*

Quanto à *descoberta de padrões de consumo*, que envolve a *extração de modelos de turista* e a *aplicação de algoritmos de mineração*, pensou-se, a princípio, em se adotar uma *técnica para extração de modelos de turista* e *uma técnica para mineração de uso*, respectivamente, no entanto, sem se considerar os detalhes de uma e de outra.

Assim, para a *extração de modelos de turista,* escolheu-se novamente o *modelo de matrizes de características*. Deste modo, os modelos de turista são dados pelas matrizes de características que são extraídas do arquivo *RDF* gravado anteriormente. Já para a *aplicação de algoritmos de mineração*, foi escolhido o uso do *algoritmo de agrupamento K-Means* (SHAHABI, BANAEI-KASHANI, 2003), que foi também selecionado para fazer a *classificação do turista*, uma vez que ele tanto constrói os modelos de grupo de turistas quanto classifica o turista em um deles.

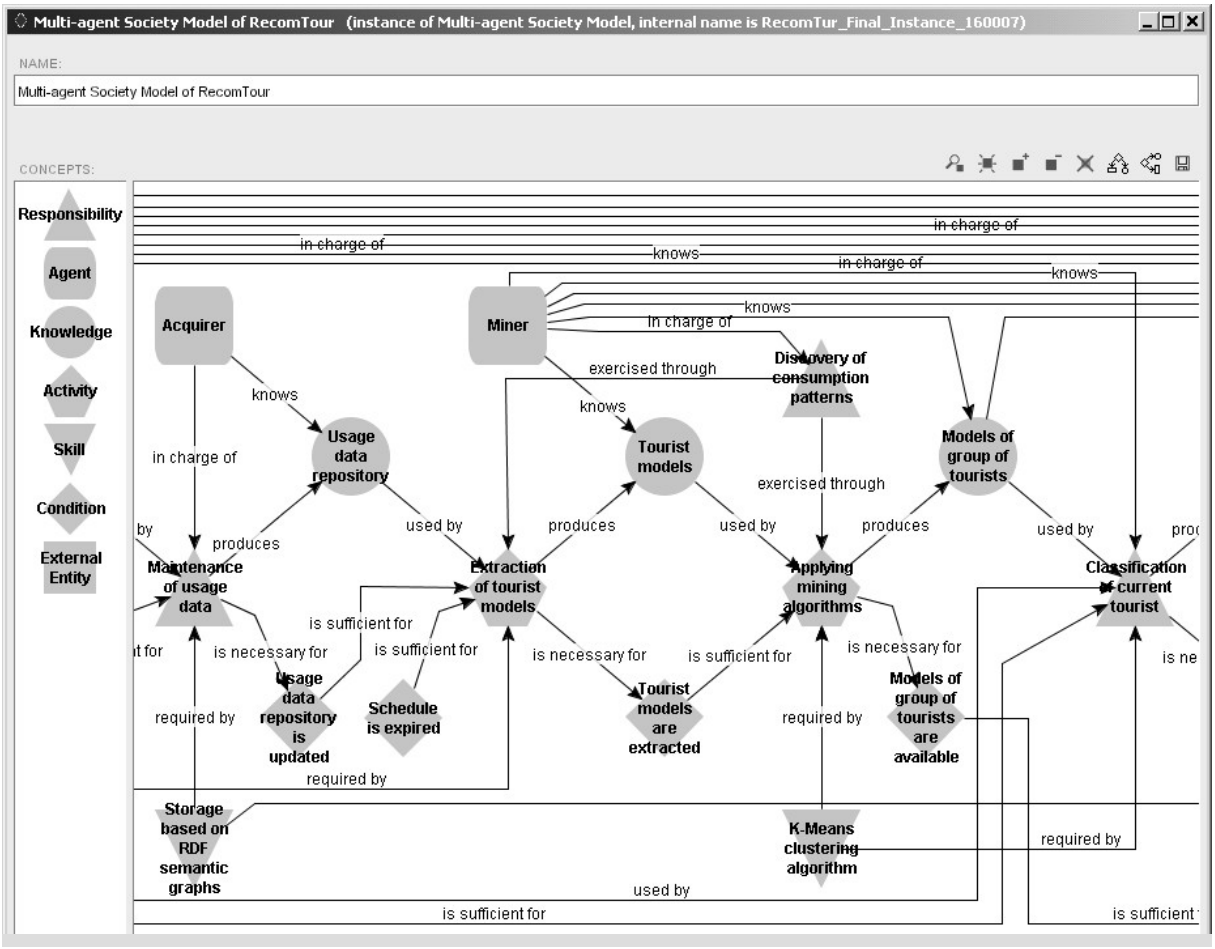


Figura 5.16: Segunda parte do Modelo da Sociedade Multiagente da *RecomTour*

Para a *construção do modelo de recomendação*, optou-se por utilizar a *abordagem da filtragem colaborativa* (HERLOCKER *et al.*, 2004), de forma a se

produzir uma lista de pacotes turísticos que serão recomendados ao turista corrente, tomando por base as escolhas de outros turistas que estão no grupo do qual o turista corrente faz parte.

Na seqüência, para a *oferta de pacotes turísticos* foi escolhida a *apresentação de janelas de alerta.* Partindo-se do modelo de recomendação anteriormente gerado, os pacotes turísticos mais recomendados são exibidos em uma janela assim que o turista fizer uma escolha qualquer.

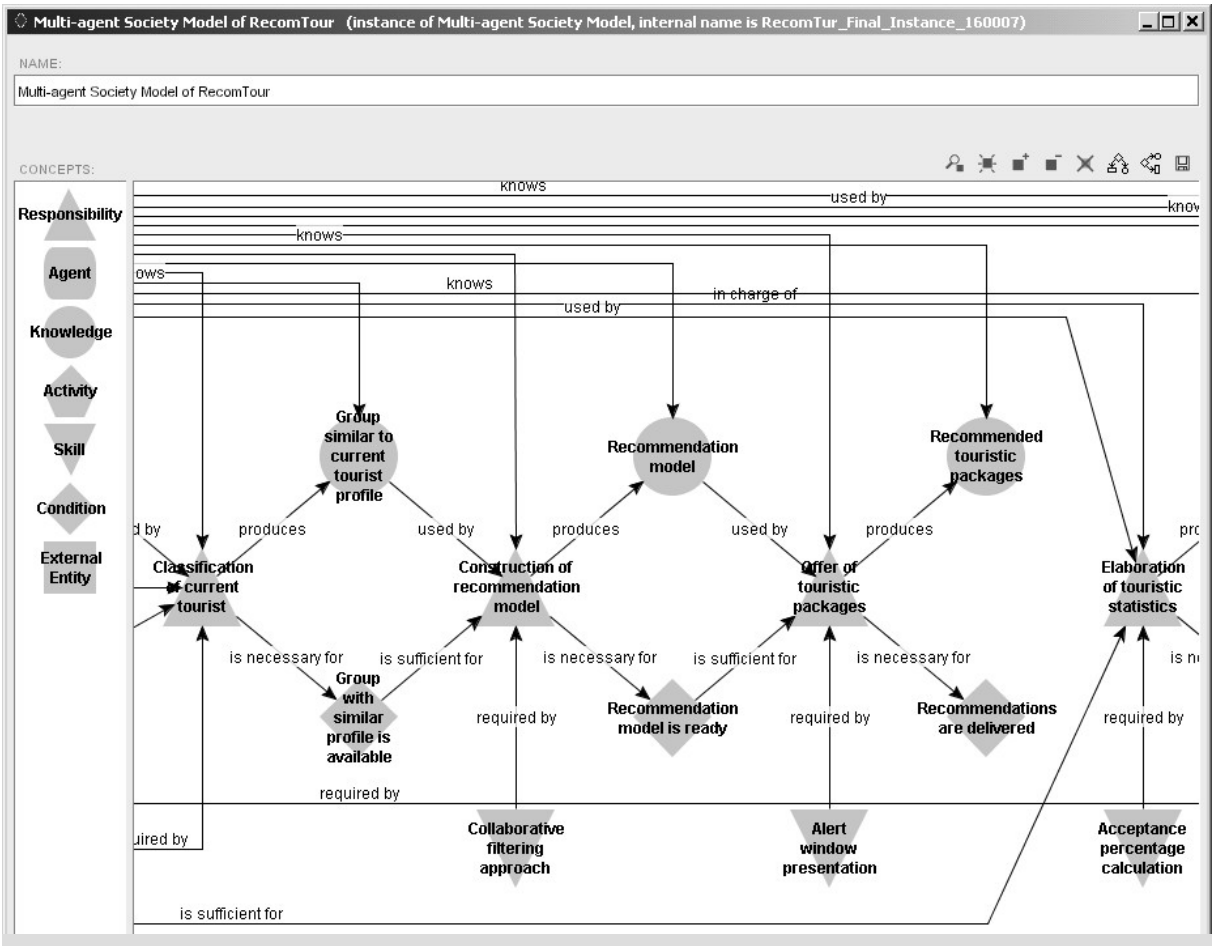


Figura 5.17: Terceira parte do Modelo da Sociedade Multiagente da *RecomTour*

No tocante à *elaboração de estatísticas turísticas*, definiu-se que esta se dá através do *cálculo da porcentagem de aceitação* de cada pacote turístico. Neste caso, para cada pacote turístico, é contada a quantidade de turistas que selecionaram tal pacote. Esta contagem se dá através da análise dos modelos de grupo de turistas. Assim, de posse do resultado dessa contagem, transforma-se esse resultado para a forma de percentual, em relação ao total de turistas usuários

do sistema. Tal porcentagem representa o percentual de aceitação do pacote turístico. No final, tem-se pares de pacotes turísticos e porcentagens.

Quanto ao *gerenciamento de pacotes turísticos*, que envolve o *cadastro de pacotes turísticos*, a *atualização de pacotes turísticos* e a *manutenção de pacotes turísticos*, cogitou-se a seleção de uma *técnica para representação de pacotes turísticos*, em relação às primeiras, e de uma *técnica para manutenção de pacotes turísticos*, referente à última.

Assim, para o *cadastro de pacotes turísticos,* decidiu-se representar um pacote turístico através da *representação matricial*, na qual cada linha representa uma característica do pacote turístico. Dessa forma, a agência de turismo especifica um pacote turístico através do preenchimento de um formulário, especificando em cada campo uma característica do pacote turístico. Cada campo é mapeado como uma linha da matriz, que representa o modelo do pacote em questão. Para a *atualização de pacotes turísticos* decidiu-se utilizar a abordagem acima explicada.

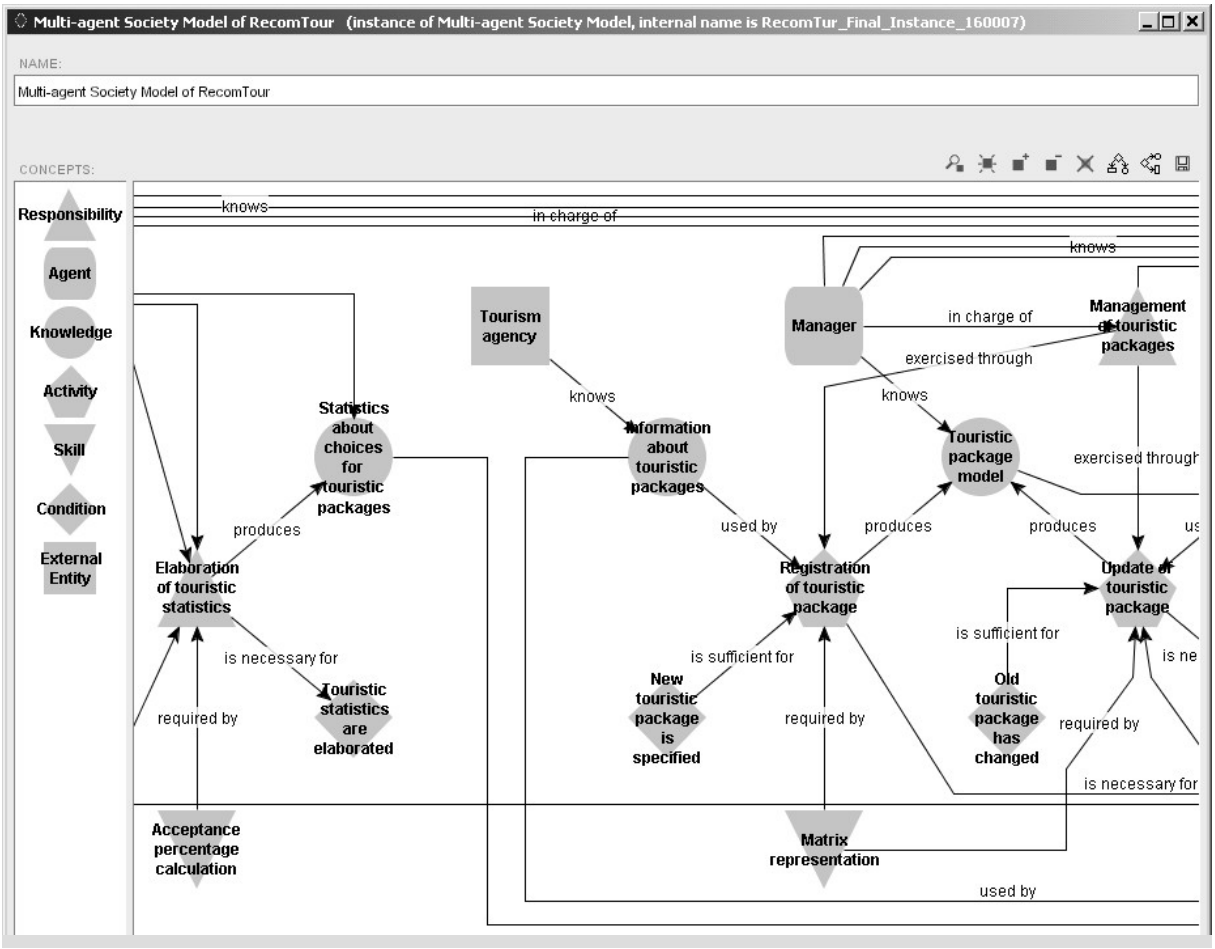


Figura 5.18: Quarta parte do Modelo da Sociedade Multiagente da *RecomTour*

No que diz respeito à *manutenção de pacotes turísticos*, a destreza selecionada corresponde àquela usada para a *manutenção de dados de uso*, de forma uniformizar a manutenção de dados no sistema.

A *busca de pacotes turísticos* compreende a *representação da consulta* e *comparação e recuperação*. Deste modo, para a *representação da consulta* utilizou-se a *criação de vetores de palavras-chave* como destreza. O turista especifica uma busca através do preenchimento de um campo de consulta, a qual será convertida no referido vetor, que dará suporte à atividade seguinte.

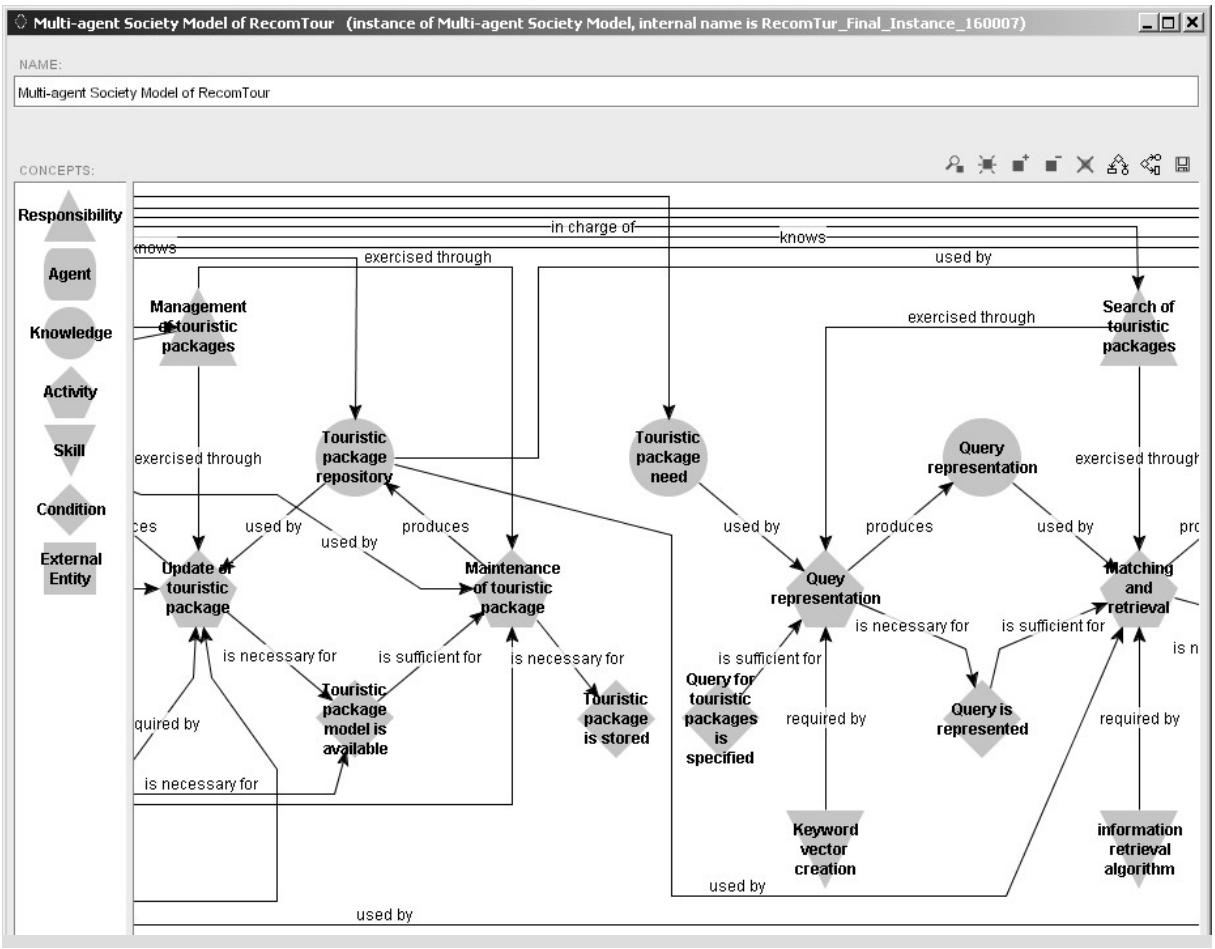


Figura 5.19: Quinta parte do Modelo da Sociedade Multiagente da *RecomTour*

Para a *comparação e recuperação*, emprega-se um *algoritmo de recuperação de informações*. Este utilizará o vetor de palavras-chave anteriormente elaborado para realizar a comparação entre as palavras-chave e os modelos de pacotes turísticos a fim de recuperar aqueles que se adequem à consulta.

Finalmente, em relação à *exibição de pacotes turísticos,* utiliza-se a *listagem ordenada de informações* para a exibição dos pacotes turísticos pertinentes

a uma agência de turismo, bem como as estatísticas referentes a cada um deles, sendo que tais informações são extraídas do repositório de pacotes turísticos.

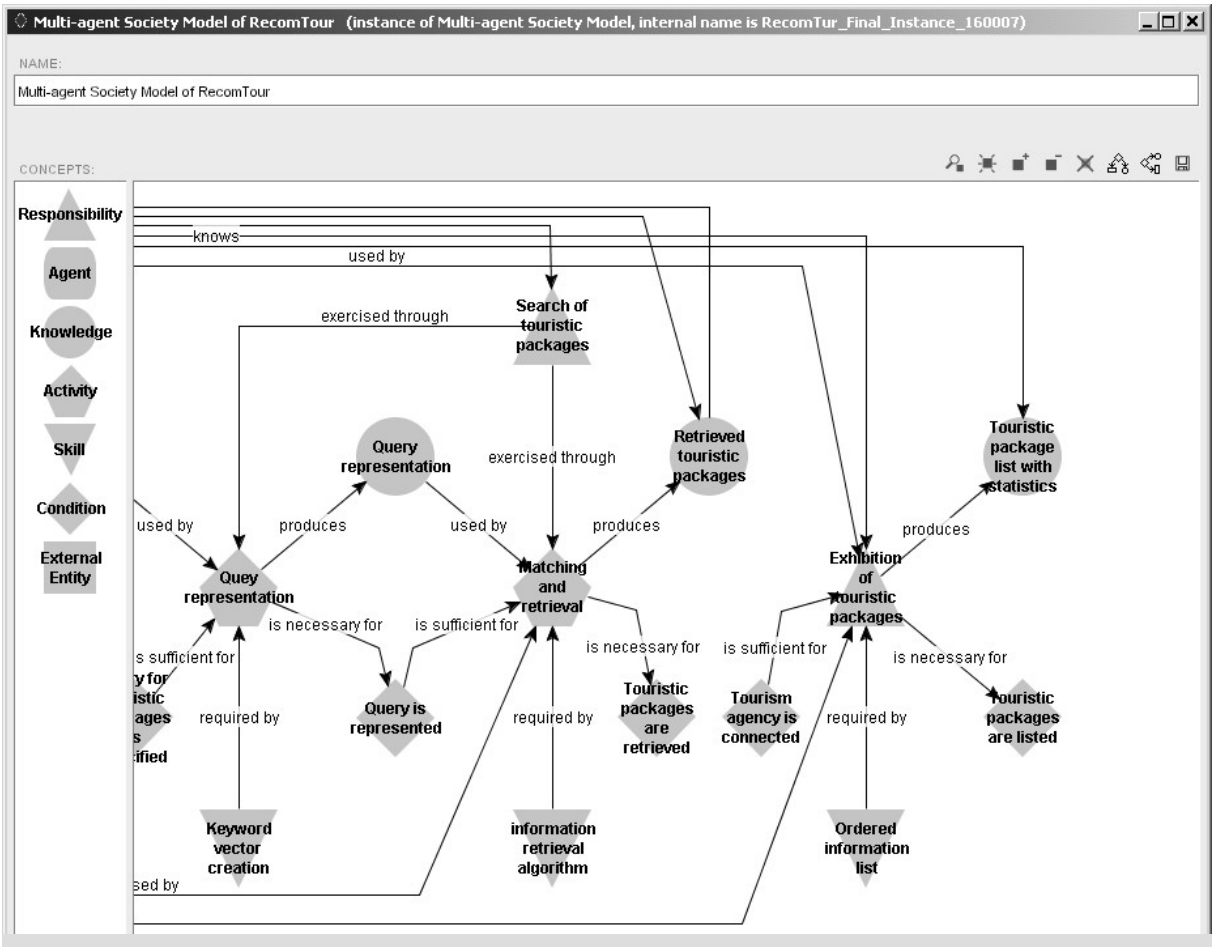


Figura 5.20: Sexta parte do Modelo da Sociedade Multiagente da *RecomTour*

A conclusão do projeto da sociedade multiagente representa um passo fundamental tanto para o termino da arquitetura de maneira geral quanto para o detalhamento de cada agente, conforme será feito a seguir.

5.3.1.2 Modelagem das interações entre agentes

A Figura 5.21 mostra o *Modelo de Interações entre Agentes* da RecomTour, no qual os agentes e as entidades externas interagem ordenadamente ao longo de suas linhas de vida, seja os primeiros trocando mensagens entre si seja sendo notificados de eventos externos pelas segundas.

Assim, inicialmente ocorre a notificação de que um "*turista entrou no sistema*", feita ao agente Interfaceador, sendo essa interação que permite o início do

funcionamento da aplicação em benefício dele, posto que é a partir das escolhas de pacotes turísticos que ele faz que são efetuadas as possíveis recomendações.

Em seguida, e como consequência imediata da interação anterior, uma mensagem "*INFORM_REF (informações sobre as escolhas de pacotes turísticos)*" é passada do agente Interfaceador para o Modelador, para que este construa o modelo do usuário corrente, de modo que possa ser empregado diretamente na classificação do turista detendor dele e de maneira indireta na geração de recomendações para outros turistas.

Depois disso, caso haja alguns outros turistas modelados, a aplicação já poderá classificar o turista corrente, o que faz após o agente Minerador receber uma mensagem "*INFORM-REF (modelo do turista corrente)*" do Modelador, sendo tal modelo uma parte do subsídio necessário para a classificação.

Logo após, o agente Minerador requer ao Aquisitor, através de uma mensagem "*QUERY_REF (repositório de dados de uso)*", os dados de uso a partir dos quais são extraídos os modelos de turista para formação de grupos em que se classifica o turista corrente, por meio da mineração de uso.

Como resultado, o agente Minerador remete ao Modelador uma mensagem "*INFORM_REF (grupo similar ao perfil do turista corrente)*" contendo o grupo em que se enquadra o turista corrente, no qual é baseado o processo de recomendação de pacotes turísticos, então realizado.

Dessa feita, resta apenas que o agente Modelador passe ao Interfaceador a mensagem "*INFORM_REF (modelo de recomendação)*" para que ele possa providenciar, com base em tal modelo, a efetiva oferta ao turista dos pacotes turísticos recomendados, recebendo uma confirmação de que os "*pacotes turísticos recomendados foram entregues*", quer dizer, aquele toma conhecimento através deste de que houve sucesso na operação que realizou.

Em paralelo, pode acontecer a notificação de que um "*turista saiu do sistema*", também feita ao agente Interfaceador, originando outra mensagem "*INFORM_REF (informações sobre as escolhas de pacotes turísticos)*" dele para o Modelador e, por fim, uma "*INFORM-REF (modelo do turista corrente)*" ao Aquisitor, o que possibilita o armazenamento dos dados da presente sessão de uso, os quais, juntamente com os outras sessões passadas e futuras, constituem outra parte do subsídio demandado pela classificação.

Além disso, de modo alternativo ao caso acima descrito, em que o turista simplesmente vasculha a interface da aplicação à procura de pacotes turísticos que lhe interessem, por ventura escolhendo algum, ele pode ainda realizar um busca determinada, que é expressa pela notificação ao agente Interfaceador de que uma "*consulta por pacotes turísticos foi especificada*", caso em que este próprio agente requisita ao Gerenciador, por meio de uma mensagem "*QUERY_REF (repositório de pacotes turísticos)*", uma relação dos pacotes disponíveis para que possa efetivar a recuperação de acordo com os critérios de busca formulados. Depois disso, dado um retorno àquele turista, há confirmação de que os "*pacotes turísticos recuperados foram entregues*".

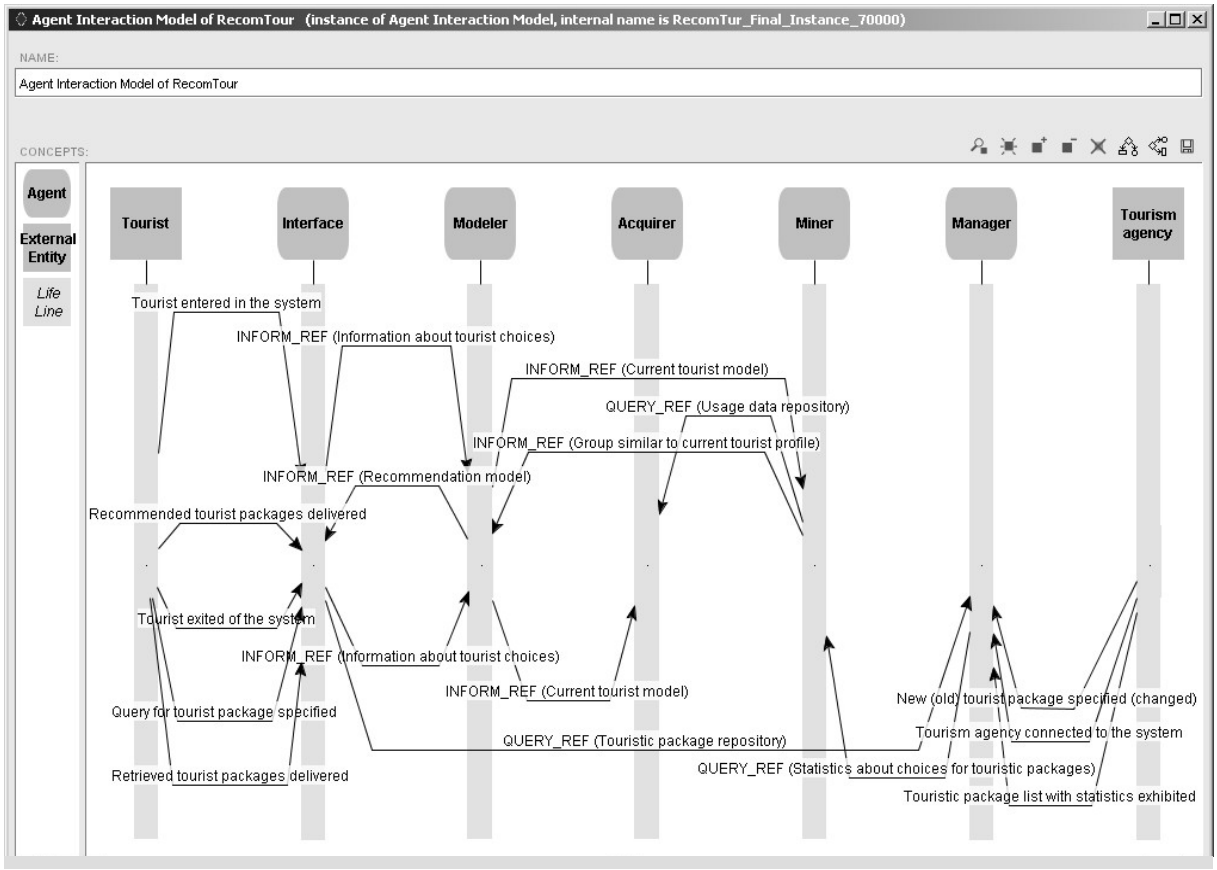


Figura 5.21: Modelo das Interações entre Agentes da *RecomTour*

Por fim, o último cenário possível tem lugar quando uma agência de turismo interage com o agente Gerenciador, notificando-a de eventos relativos a que um "*novo pacote turístico tenha sido especificado*" ou que um "*pacote turístico antigo tenha sofrido mudanças*", hipótese na qual este mesmo agente realizará o processamento exigido à manutenção dos pacotes turísticos pela aplicação.

No mais, de um jeito complementar, a notificação pode ser de que uma "*agência de turismo está conectada*", hipótese em que o movimento previsto é a exibição a ela dos seus respectivos pacotes acompanhados de estatísticas referentes à opção por tais da parte dos turistas. O sucesso nessa operação é denotado pelo aviso de que a "*lista de pacotes turísticos com estatísticas foi exibida*".

Definidas as interações entre agentes, chega-se à hora de organizá-los como uma arquitetura propriamente dita, visto que, até agora, somente se fizera um esboço de tal organização.

5.3.1.3 Modelagem dos mecanismos de cooperação e coordenação

A Figura 5.22 exibe o *Modelo dos Mecanismos de Cooperação e Coordenação* da RecomTour, que organiza os agentes da aplicação de acordo com requisitos funcionais e não-funcionais e com mecanismos de cooperação e coordenação na forma de uma arquitetura padronizada.

Em face disso, foi adotado o padrão arquitetural *Camada Multiagente* (GIRARDI *et al.*, 2005b), o qual:

- é apropriado justamente para o contexto dos sistemas multiagente;
- foca o problema de projetar a arquitetura do sistema em grupos de agentes com responsabilidades similares, sendo cada qual um diferente nível de abstração;
- apresenta como forças a contribuição para o entendimento, para a manutenção e para a reutilização dos sistemas, já que alterações em uma camada afetam no máximo as outras adjacentes;
- propõe uma solução que envolve a organização do sistema multiagente em uma hierarquia de camadas e a definição de um critério de abstração para a divisão das responsabilidades entre as camadas, sendo que cada uma delas reúne agentes que se comunicam entre si e com as camadas vizinhas, visando prover serviços à camada superior, que opera como sua cliente, e delegar tarefas à camada inferior, que trabalha com sua servidora;

- tem como um de seus usos principais a estruturação de subsistemas, o que acontece tipicamente nos sistemas multiagente de recuperação e filtragem de informação.

Adicionalmente, quanto à arquitetura em camadas, cabe ressaltar que a estrutura e a funcionalidade de cada uma das camadas pode ser descrita por padrões independentes. Além disso, o seu mecanismo de cooperação é suportado pelas relações de dependência entre as camadas, enquanto que a coordenação pode ser abordada através de outros padrões, tais como *federativo* ou *mestre-escravo*.

No caso presente, foram levadas em conta questões relativas a coesão e acoplamento entre as entidades internas, eliminação de interações entre partes separadas etc, bem como características particulares da aplicação, em especial as decorrentes da clara separação entre dois de seus sub-objetivos, os quais cuidam do turista, do outro, que se ocupa da agência de turismo. Isso levou à decisão por duas camadas, que por sua vez interagem com as respectivas entidades externas.

Assim, no nível de abstração "*2*", encontra-se a camada de *recomendação de pacotes turísticos*, da qual fazem parte os agentes Interfaceador, Modelador, Aquisitor e Minerador, sendo que o primeiro interage com o turista tanto para monitorar *escolhas de pacotes turísticos* e processar qualquer *consulta por pacotes turísticos* quanto para encaminhar *pacotes turísticos recomendados* ou *pacotes turísticos recuperados*, e com a camada inferior, da qual recebe uma cópia do *repositório de pacotes turísticos.* Além disso, o segundo interage com todos os demais e o terceiro interage com o quarto, que, por sua vez, produz conhecimentos para o outro nível.

Já no nível de abstração "*1*", situa-se a camada de *gerenciamento de pacotes turísticos*, participando dela apenas o agente Gerenciador, o qual concentra as atividades de processar tanto a *especificação de um novo pacote turístico* quanto as *mudanças em um pacote turístico antigo*, vindas da agência de turismo, e também usa *estatísticas sobre escolhas de pacotes turísticos* recebidas da camada superior, entregando-as na forma de uma *lista de pacotes turísticos com estatísticas* à entidade externa.

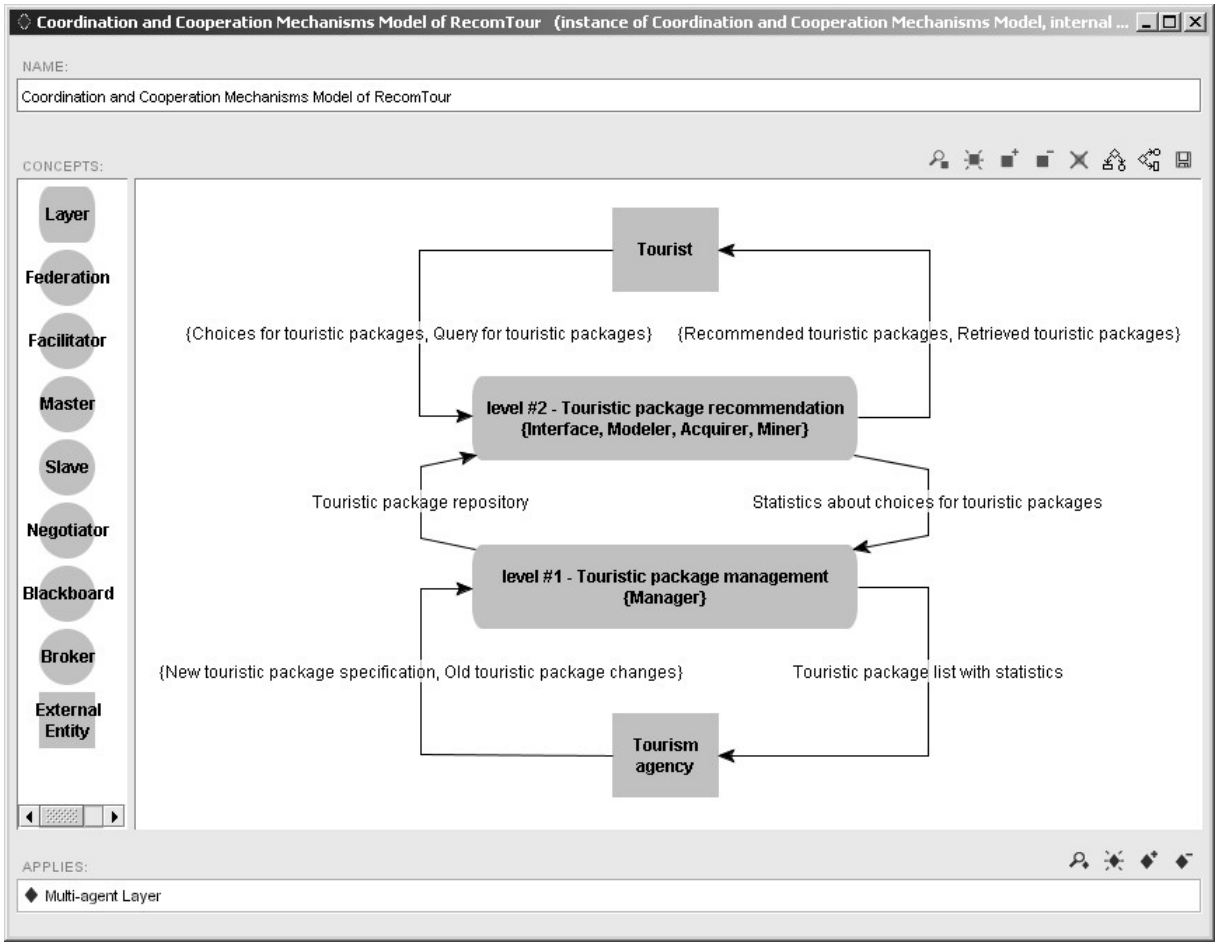


Figura 5.22: Modelo dos Mecanismos de Cooperação e Coordenação da *RecomTour*

Concluída a arquitetura geral da aplicação, mostrando externamente quais são os seus agentes e como eles se relacionam, passa-se desse ponto em diante a detalhar cada um de tais agentes, nos termos seguintes.

5.3.2 Projeto do agente *Interfaceador*

Nesta primeira iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Interfaceador da RecomTour (Figura 5.23), que é formado pelos *Modelos do Conhecimento e das Atividades do Agente* Interfaceador, sendo um para cada uma de suas três responsabilidades, e pelo *Modelo dos Estados do Agente* Interfaceador.

No caso desse agente, foi possível a reutilização do padrão *WUMA-Interface* (GIRARDI *et al.*, 2005b), que detalha o projeto de um agente Interfaceador no contexto dos sistemas multiagente, o qual soluciona problemas de administração

das interações dos usuários com o sistema e da aquisição de informações sobre o comportamento do usuário a fim de prover a ele uma interface personalizada.

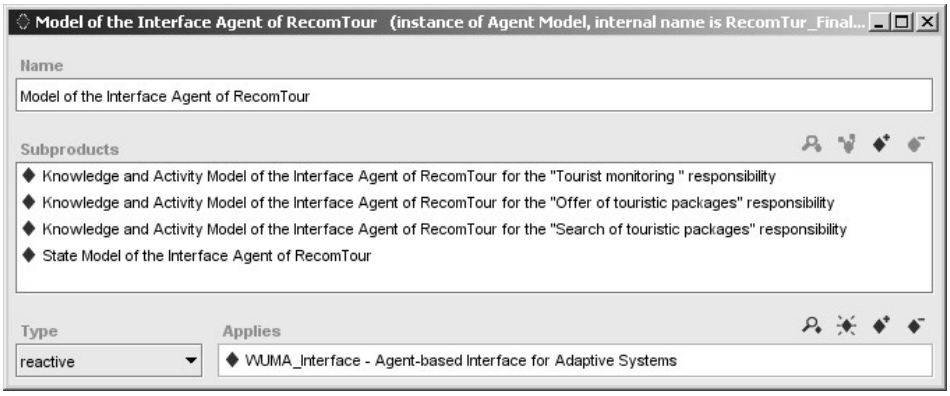


Figura 5.23: Modelo do Agente *Interfaceador* da *RecomTour*

5.3.2.1 Modelagem do conhecimento e das atividades do agente Interfaceador

A Figura 5.24, a Figura 5.25 e a Figura 5.26 mostram os três *Modelos do Conhecimento e das Atividades do Agente* Interfaceador da RecomTour, referentes à *monitoração de turistas,* à *oferta de pacotes turísticos* e à *busca de pacotes turísticos*, respectivamente, que são as três responsabilidades desse agente. Tais modelos representam e detalham de maneira isolada o contexto de execução dessas responsabilidades, sendo que a última delas está decomposta nas atividades de *representação da consulta* e de *comparação e recuperação*.

Assim, quanto às pré e pós-condições, respectivamente, para a *monitoração de turistas*, é suficiente que um *turista entre no sistema*, enquanto que a execução de tal responsabilidade, mediada pela *captura implícita de escolhas de turistas*, é necessária para que as *informações sobre as escolhas estejam capturadas*. Nesse processo, para cada de turista monitorado, são usadas as respectivas *escolhas do turista por pacotes turísticos* e são produzidas as *informações sobre as escolhas do turista*, sendo estes últimos conhecidos pelo agente *Interfaceador*.

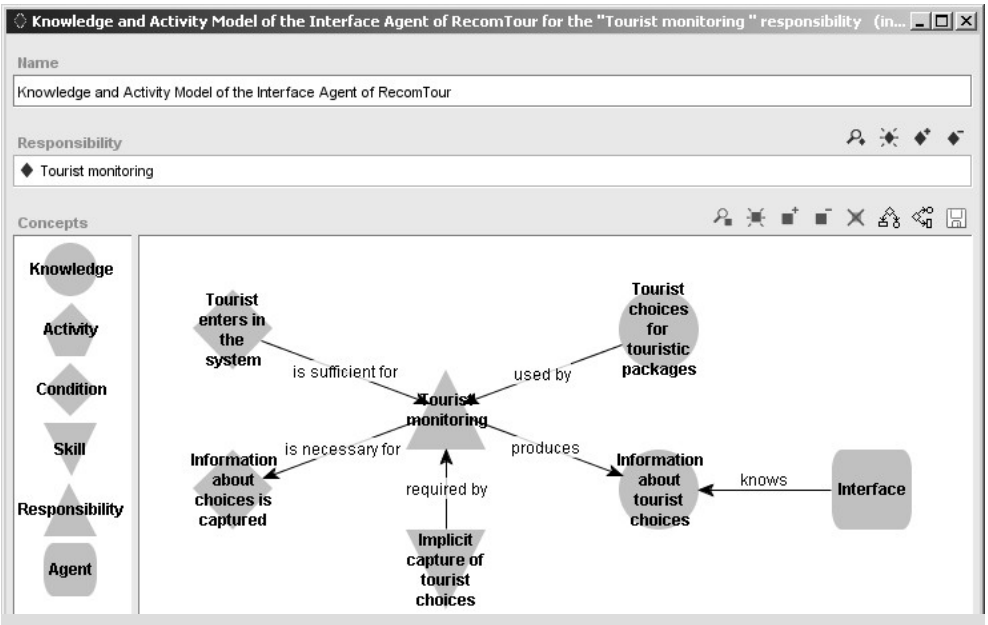


Figura 5.24: Modelo do Conhecimento e das Atividades do Agente *Interfaceador* para a responsabilidade *monitoração de turistas*

Em relação à *oferta de pacotes turísticos* é suficiente que um *modelo de recomendação esteja pronto*, sendo esta responsabilidade necessária para que *recomendações sejam entregues*, isso feito através da *apresentação de recomendações em janelas de alerta*. Nesse processo, é usado o *modelo de recomendação*, que é de conhecimento do agente *Modelador*, para produzir *recomendações de pacotes turísticos*, que são de conhecimento do *Interfaceador*.

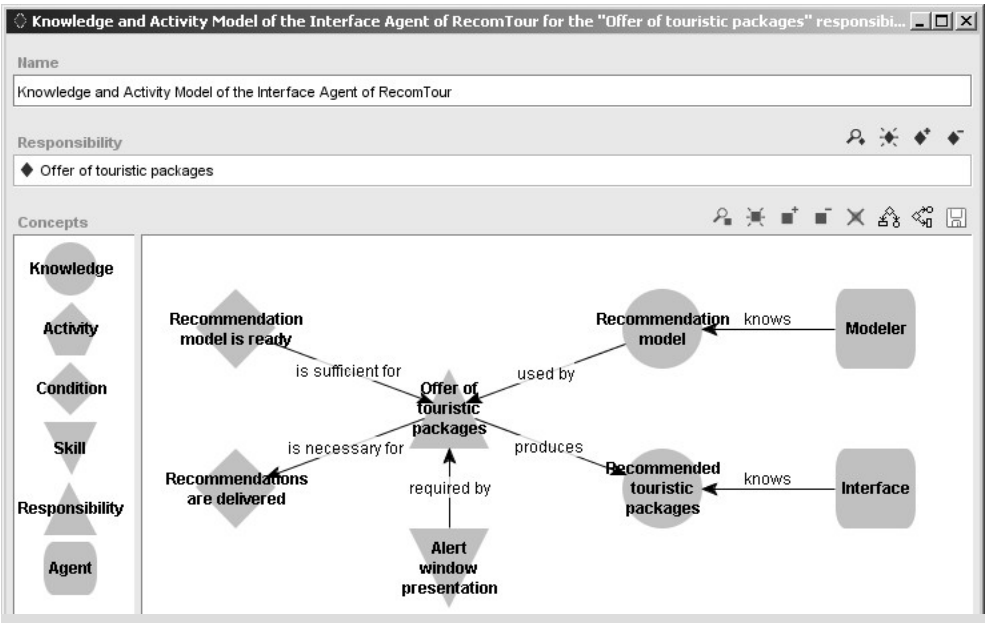


Figura 5.25: Modelo do Conhecimento e das Atividades do Agente *Interfaceador* para a responsabilidade *oferta de pacotes turísticos*

Assim, a *representação da consulta* tem como pré-condição que uma *consulta por pacotes turísticos tenha sido especificada* e como pós-condição que a *consulta esteja representada*, sendo a destreza requerida para tanto a *criação de vetores de palavras-chave*. Há, na consecução dessa atividade, o uso de uma *necessidade de pacote turístico* e a produção da *representação da consulta*, que é de conhecimento do agente *Interfaceador*.

Já para a *comparação e recuperação*, é suficiente que a *consulta esteja representada*, sendo tais necessárias para que *pacotes turísticos sejam recuperados*, tudo através de um *algoritmo de recuperação de informações*. Nesse processo, são usadas a *representação da consulta* e o *repositório de pacotes turísticos*, que é de conhecimento do agente *Gerenciador*, sendo produzidos *pacotes turísticos recuperados*, também conhecidos pelo agente *Interfaceador*.

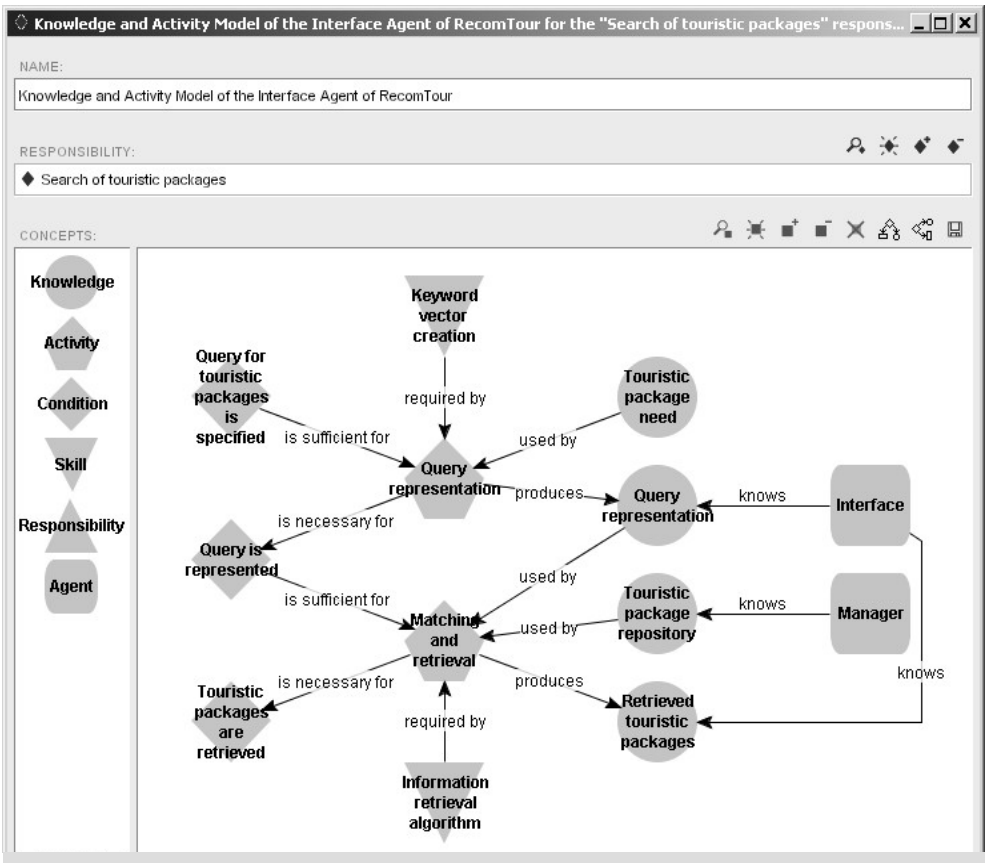


Figura 5.26: Modelo do Conhecimento e das Atividades do Agente *Interfaceador* para a responsabilidade *busca de pacotes turísticos*

5.3.2.2 Modelagem dos estados do agente Interfaceador

A Figura 5.27 exibe o *Modelo dos Estados do Agente* Interfaceador da RecomTour, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Dessa forma, uma vez que um *turista entre no sistema*, o agente Interfaceador inicia sua operação, permanecendo no estado *monitorando o comportamento de consumo do turista ou aguardando um consulta por pacotes turísticos*. Se uma *consulta por pacotes turísticos é especificada*, então o ele assume o estado *representando uma consulta por pacotes turísticos*, no qual ele verifica os dados fornecidos pelo turista e cria um vetor de palavras-chave a partir da consulta.

Quando a *consulta está representada*, então o agente transita para o estado *comparando e recuperando pacotes turísticos*, em que é realizada a comparação entre a consulta e os pacotes turísticos contidos no repositório, havendo a recuperação de alguns. No momento em que os *pacotes turísticos são recuperados,* o agente retorna ao seu estado de monitoração e espera.

Já quando um *modelo de recomendação está pronto*, então o agente vai para o estado *processando recomendações de pacotes turísticos*, o que é realizado a partir do modelo, fazendo com que ele retorne para o seu estado de monitoração e espera, do qual pode passar para o final se o *turista sair do sistema*.

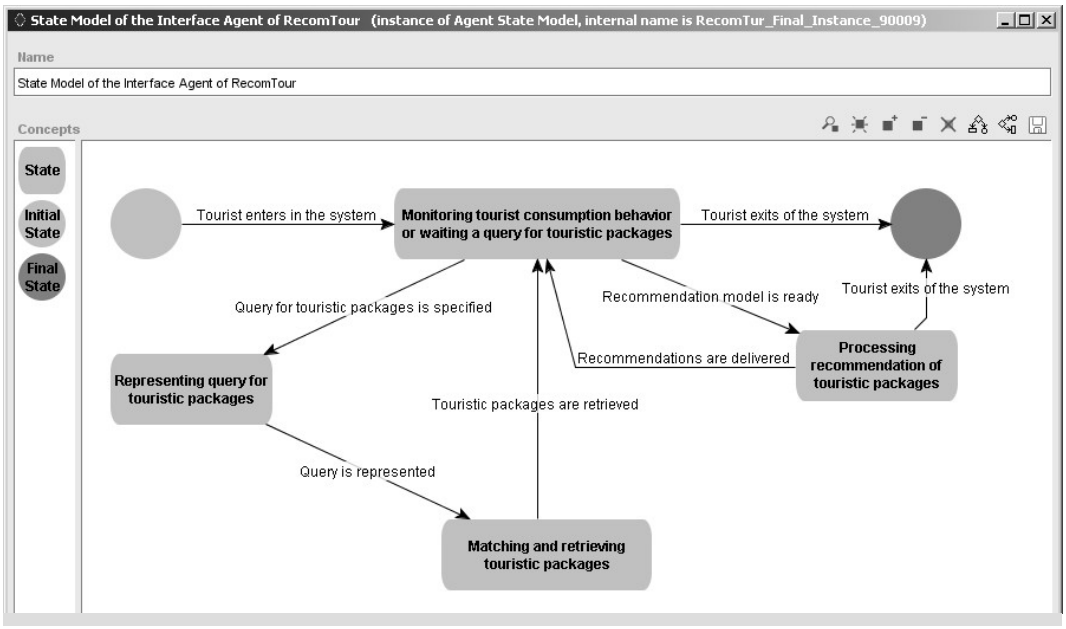


Figura 5.27: Modelo dos Estados do Agente *Interfaceador*

5.3.3 Projeto do agente *Modelador*

Nesta segunda iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Modelador da RecomTour (Figura 5.28), que é formado pelos *Modelos do Conhecimento e das Atividades do Agente* Modelador, sendo um para cada uma de suas duas responsabilidades, e pelo *Modelo dos Estados do Agente* Modelador.

Quanto a esse agente, houve o reuso do padrão *WUMA-UserM* (GIRARDI *et al.*, 2005b), que descreve o projeto de um agente para a representação e atualização de modelos de usuários, visto que em sistemas adaptáveis é necessária a identificação e caracterização dos usuários.

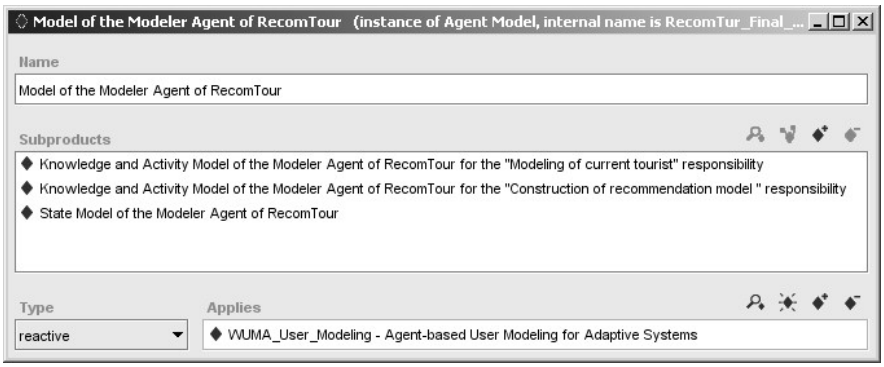


Figura 5.28: Modelo do Agente *Modelador* da *RecomTour*

5.3.3.1 Modelagem do conhecimento e das atividades do agente Modelador

A Figura 5.29 e a Figura 5.30 mostram os dois *Modelos do Conhecimento e das Atividades do Agente* Modelador da RecomTour, referentes à *modelagem do turista corrente* e à *construção do modelo de recomendação*, respectivamente, que são as duas responsabilidades desse agente. Tais modelos representam e detalham de maneira isolada o contexto de execução dessas responsabilidades.

Desse modo, para a *modelagem de turista corrente* é suficiente que *informações sobre escolhas do turista tenham sido capturadas*, sendo que ela é necessária para que o *modelo do turista corrente esteja disponível*, isso através do *modelo de matrizes de características*. Nesse processo são usadas *informações sobre escolhas do turista*, que são de conhecimento do agente *Interfaceador*, e é produzido o *modelo do turista corrente*, conhecido pelo *Modelador*.

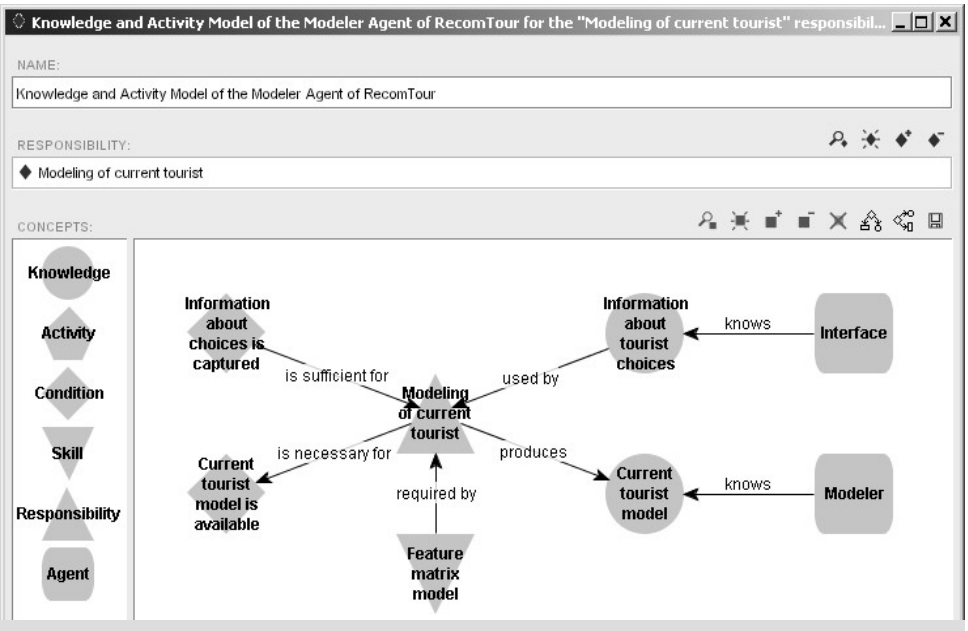


Figura 5.29: Modelo do Conhecimento e das Atividades do Agente *Modelador* para a responsabilidade *modelagem do turista corrente*

Já a *construção do modelo de recomendação* tem como pré-condição que o *grupo com perfil similar ao do turista corrente esteja disponível* e como pós-condição que *o modelo de recomendação esteja pronto*, sendo a destreza requerida para tanto a *abordagem da filtragem colaborativa*. Há, na execução dessa atividade, o uso do *grupo similar ao perfil do turista corrente*, de conhecimento do agente *Minerador*, e a produção do *modelo de recomendação*, conhecido pelo *Modelador*.

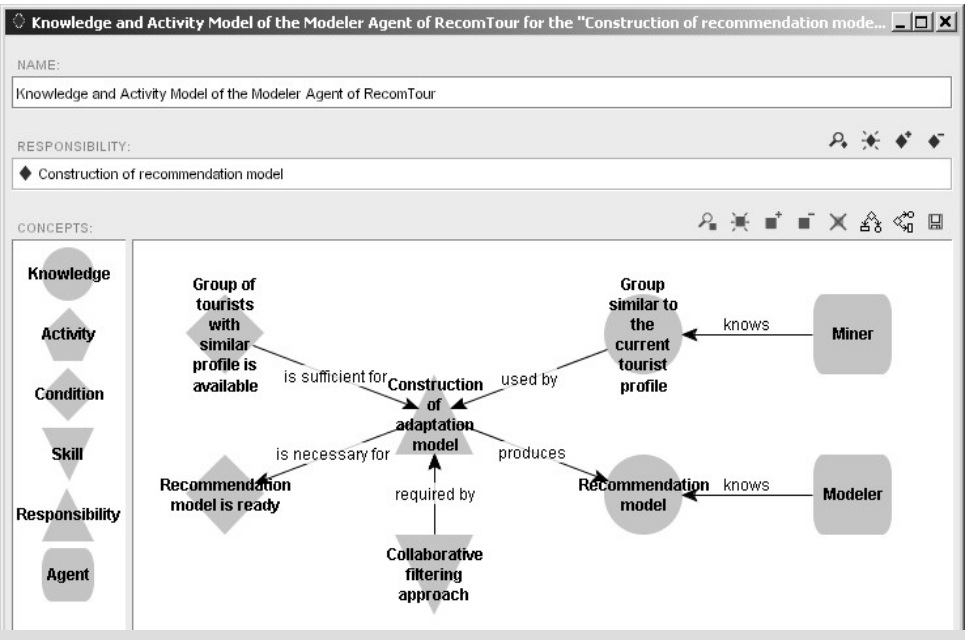


Figura 5.30: Modelo do Conhecimento e das Atividades do Agente *Modelador* para a responsabilidade *construção do modelo de recomendação*

5.3.3.2 Modelagem dos estados do agente Modelador

A Figura 5.31 exibe o *Modelo dos Estados do Agente* Modelador da RecomTour, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

O agente Modelador inicia sua execução no estado *aguardando informações sobre escolhas do turista vindas do agente Interfaceador*, isso após que um *turista entrar no sistema*. Quando *informações sobre escolhas do turista são capturadas*, o estado dele passa a ser *modelando o turista corrente*, em que ele utiliza tais informações para construir um novo modelo ou atualizar já existente.

Quando o *modelo do turista corrente está disponível,* o agente passa ao estado *aguardando a classificação do turista corrente feita pelo agente Minerador*, Uma vez que o *grupo com perfil similar ao do turista está disponível,* o agente muda para o estado *construindo modelo de recomendação*, no qual ele identifica os pacotes turísticos escolhidos por turistas com perfis similares ao do corrente.

Logo que o *modelo de adaptação está disponível*, o agente volta para o estado *aguardando por informações sobre escolhas do turista vindas do agente Interfaceador*, sendo que deste ou de qualquer dos outros estados ele pode transitar para o final quando o *turista sai do sistema*.

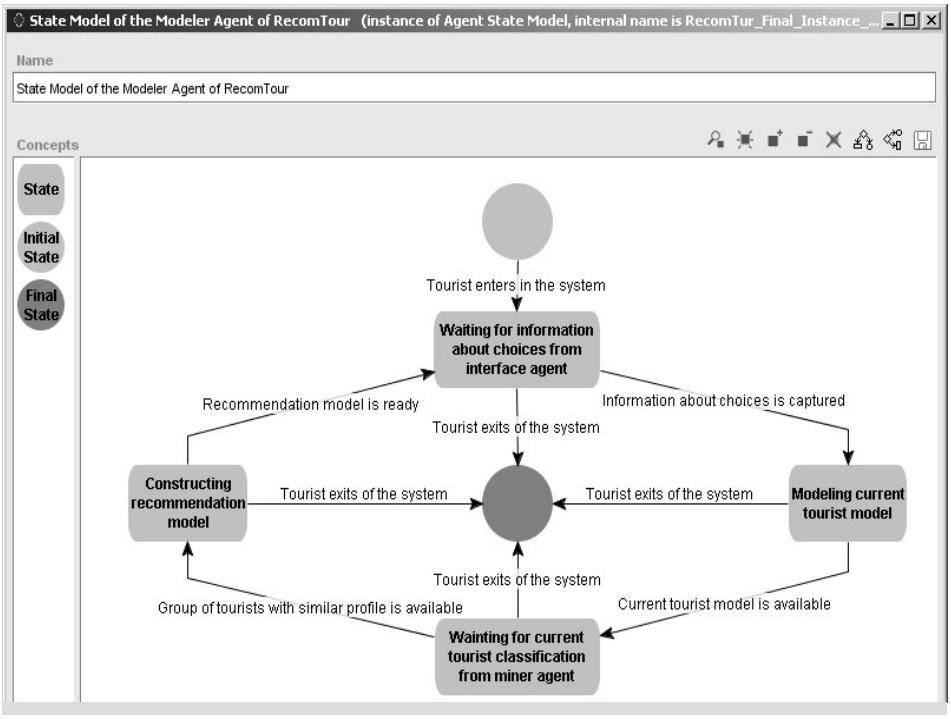


Figura 5.31: Modelo dos Estados do Agente *Modelador*

5.3.4 Projeto do agente *Aquisitor*

Nesta terceira iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Aquisitor da RecomTour (Figura 5.32), que é formado pelo *Modelo do Conhecimento e das Atividades do Agente* Aquisitor para sua única responsabilidade e pelo *Modelo dos Estados do Agente* Aquisitor.

Em relação a tal agente, reutilizou-se o padrão *WUMA-Acquirer* (GIRARDI *et al.*, 2005b), que detalha o projeto de um agente Aquisitor em um sistema adaptável baseado na mineração de uso, sendo que ele possibilita a manutenção de um repositório de dados de uso, o qual provê informações sobre o comportamento de navegação de cada usuário.

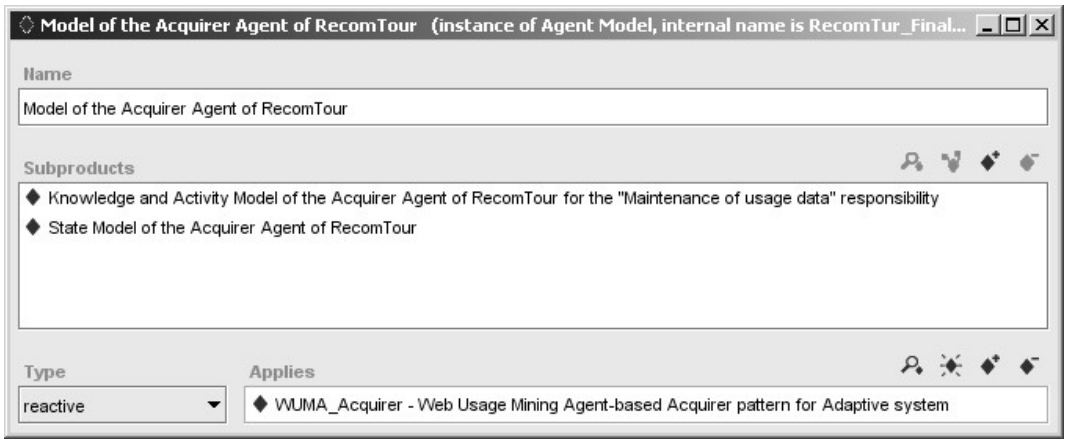


Figura 5.32: Modelo do Agente *Aquisitor* da *RecomTour*

5.3.4.1 Modelagem do conhecimento e das atividades do agente Aquisitor

A Figura 5.33 mostra o *Modelo do Conhecimento e das Atividades do Agente* Aquisitor da RecomTour, referente à *manutenção de dados de uso*, que é a única responsabilidade desse agente. Tal modelo representa e detalha de maneira isolada o contexto de execução dessa responsabilidade.

Assim, a *manutenção de dado de uso* tem como pré-condição que o *turista saia do sistema* e como pós-condição que o *repositório de dados de uso esteja atualizado*, requerendo como destreza os *grafos semânticos em RDF*. Na execução dessa responsabilidade, ocorre o uso do *modelo do turista corrente*, de conhecimento do agente *Modelador*, e a produção do *repositório de dados de uso*, conhecido pelo *Aquisitor*.

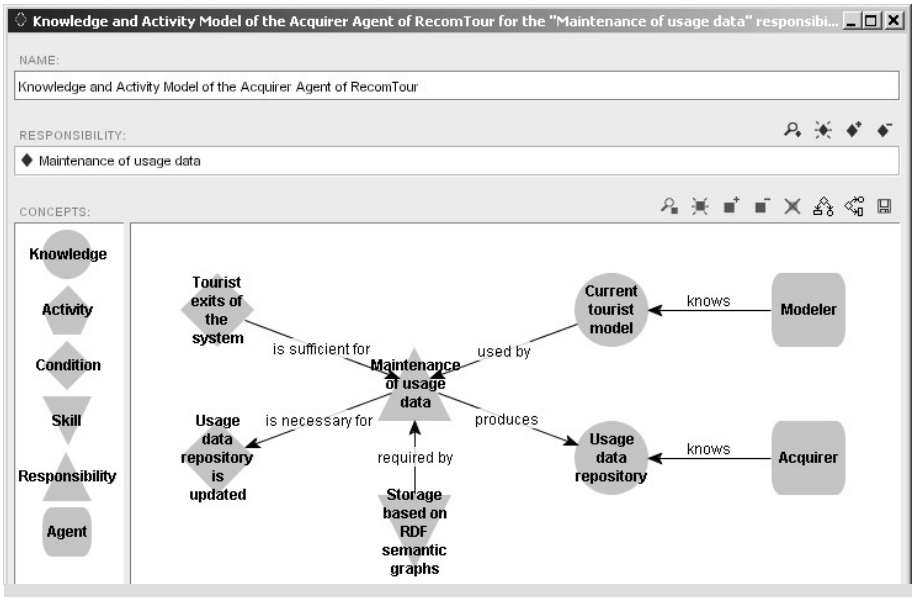


Figura 5.33: Modelo do Conhecimento e das Atividades do Agente *Aquisitor* para a responsabilidade *manutenção de dados de uso*

5.3.4.2 Modelagem dos estados do agente Aquisitor

A Figura 5.34 exibe o *Modelo dos Estados do Agente* Aquisitor da RecomTour, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Desse modo, o agente Aquisitor transita de seu estado inicial quando o *sistema é inicializado* e fica *aguardando o modelo do turista corrente vindo do agente Modelador*, sendo que isso ocorre quando o *turista sai do sistema*, levando ao estado *atualizando o repositório de dados de uso*., no qual é gerado um identificador para a sessão de consumo do turista , de modo que ela possa ser mantida em um *arquivo RDF* para uso futuro na classificação de outros turistas.

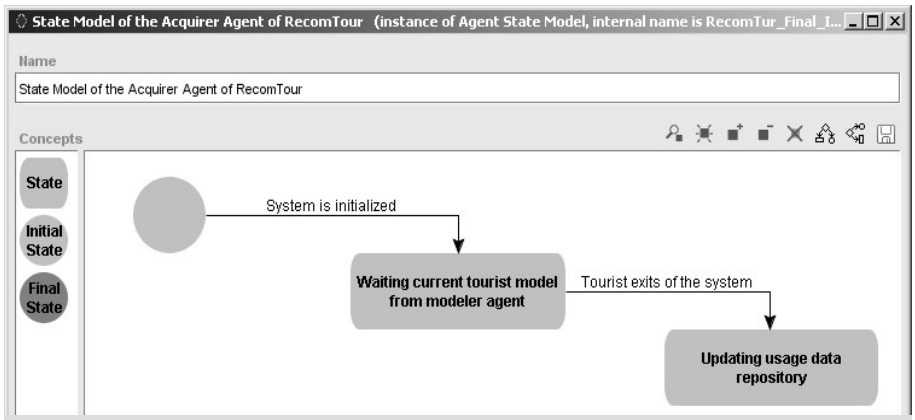


Figura 5.34: Modelo dos Estados do Agente *Aquisitor*

5.3.5 Projeto do agente *Minerador*

Nesta quarta iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Minerador da RecomTour (Figura 5.35), que é formado pelos *Modelos do Conhecimento e das Atividades do Agente* Minerador, sendo um para cada uma de suas três responsabilidades, e pelo *Modelo dos Estados do Agente* Minerador.

No tocante a esse agente, fez o reuso do padrão *WUMA-Miner* (GIRARDI *et al.*, 2005b), que descreve o projeto de um agente Minerador para sistemas multiagente adaptáveis baseados em técnicas de mineração de uso, tendo ele como finalidade a descoberta de grupos de usuários com perfis similares e a qual desses grupos pertence um determinado usuário.

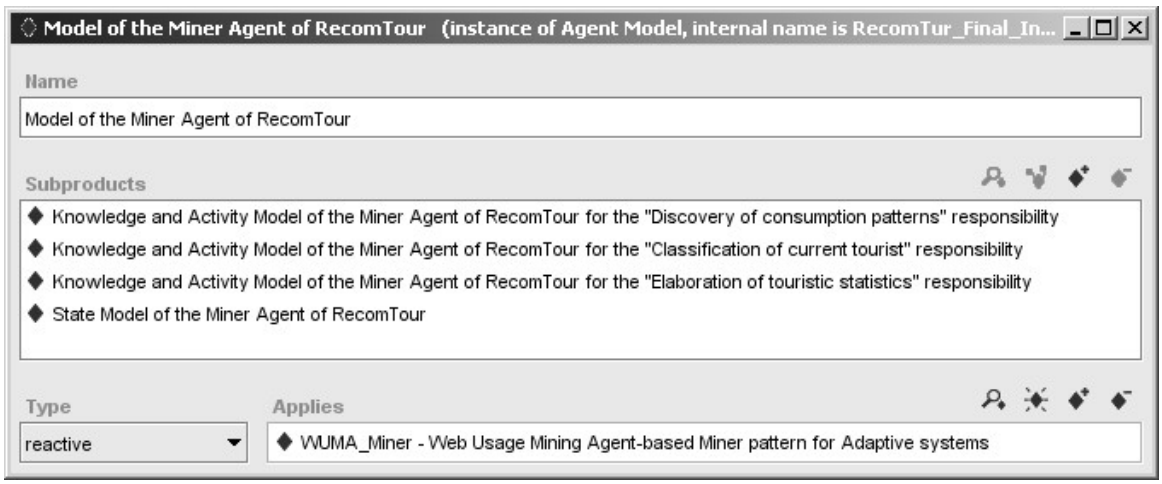


Figura 5.35: Modelo do Agente *Minerador* da *RecomTour*

5.3.5.1 Modelagem do conhecimento e das atividades do agente Minerador

A Figura 5.36, a Figura 5.37 e a Figura 5.38 mostram os três *Modelos do Conhecimento e das Atividades do Agente* Minerador da RecomTour, referentes à *descoberta de padrões de consumo*, à *classificação do turista corrente* e *elaboração de estatíticas turísticas*, respectivamente, que são as três responsabilidades desse agente. Tais modelos representam e detalham de maneira isolada o contexto de execução dessas responsabilidades, sendo que a primeira delas está decomposta nas atividades de *extração de modelos de turista* e de *aplicação de algoritmos de mineração*.

A *extração de modelos de turista* tem como pré-condições que um *tempo de espera tenha transcorrido* e que o *repositório de dados de uso esteja atualizado* e como pós-condição que *modelos de turista tenham sido extraídos*, sendo que a destreza requerida para isso é o *modelo de matrizes de características*. Há, na execução dessa atividade, o uso do *repositório de dados de uso*, de conhecimento do agente *Aquisitor*, e a produção de *modelos de turista*, conhecidos pelo *Minerador*.

Já para a *aplicação de algoritmos de mineração*, é suficiente que *modelos de turista tenham sido extraídos*, sendo que tal é necessária para que *modelos de grupo de turistas estejam disponíveis,* isto através do *algoritmo de agrupamento K-Means*. Nesta atividade ocorre o uso de *modelos de turista* e a produção de *modelos de grupo de turistas*, ambos de conhecimento do *Minerador*.

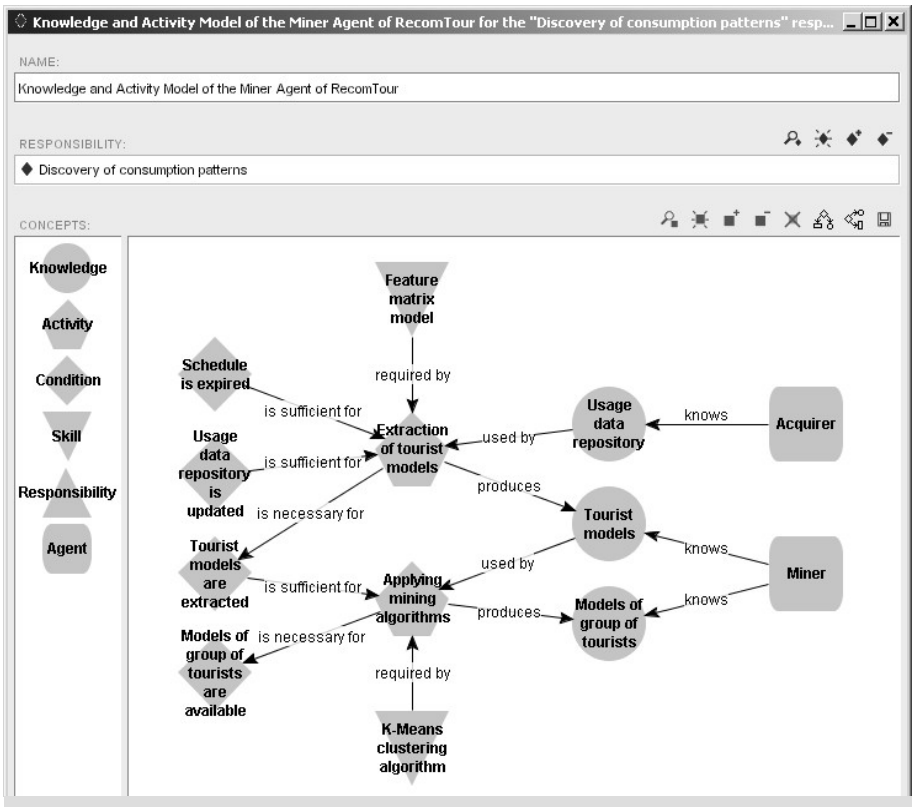


Figura 5.36: Modelo do Conhecimento e das Atividades do Agente *Minerador* para a responsabilidade *descoberta de padrões de consumo*

Quanto à *classificação do turista corrente*, a pré-condição é que a *modelo do turista corrente esteja disponível* e e a pós-condição que um *grupo com perfil similar ao do turista corrente esteja disponível*, sendo que a destreza requerida para isto é o *algoritmo de agrupamento K-Means*. Há no exercício desta responsabilidade o uso do *modelo do turista corrente*, de conhecimento do agente *Modelador*, e dos

modelos de grupo de turistas, conhecidos pelo *Minerador*, e a produção do *grupo similar ao perfil do turista corrente*, também de conhecimento do *Minerador*.

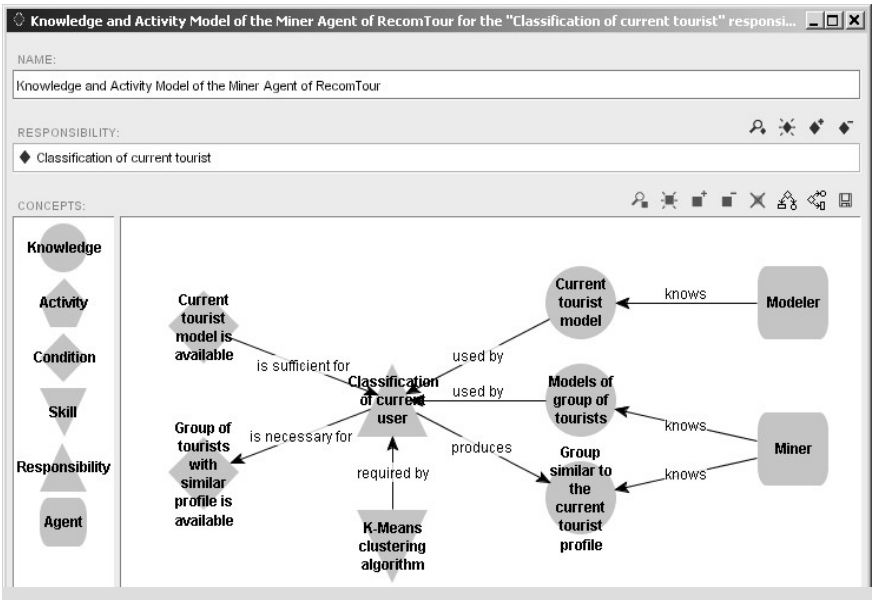


Figura 5.37: Modelo do Conhecimento e das Atividades do Agente *Minerador* para a responsabilidade *classificação do turista corrente*

Já em relação à *elaboração de estatísticas turísticas* é suficiente que *modelos de grupo de turistas estejam disponíveis*, e ela é necessária para que *estatísticas turísticas sejam elaboradas*, sendo a destreza para tal o *cálculo da porcentagem de aceitação*. São usados os *modelos de grupo de turista* e produzidas *estatísticas sobre escolhas de pacotes turísticos*, conhecidos pelo Minerador.

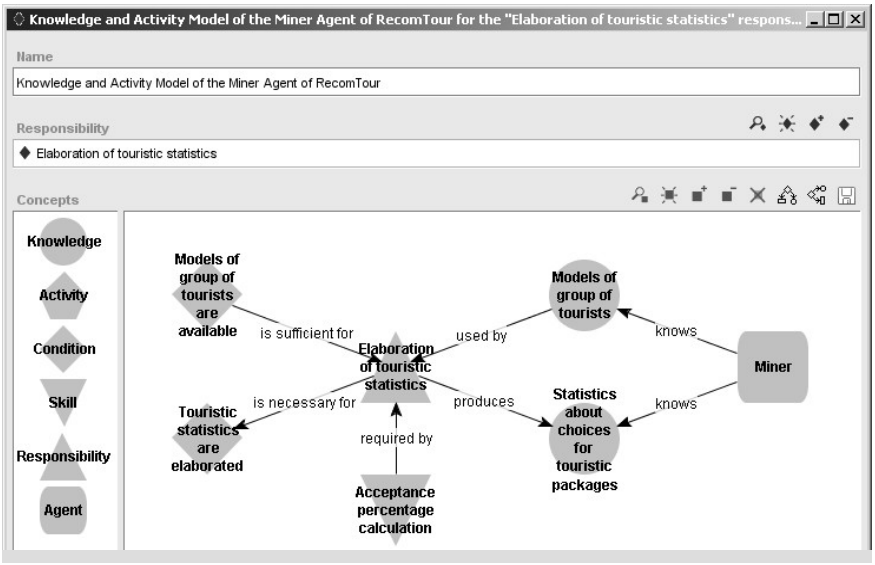


Figura 5.38: Modelo do Conhecimento e das Atividades do Agente *Minerador* para a responsabilidade *elaboração de estatíticas turísticas*

5.3.5.2 Modelagem dos estados do agente Minerador

A Figura 5.39 exibe o *Modelo dos Estados do Agente* Minerador da RecomTour, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Dessa maneira, quando o *sistema é inicializado*, o agente Minerador assume o estado *aguardando a expiração do primeiro tempo de espera*. Uma vez que isso ocorra, ele assume o estado *extraindo modelos de turista do repositório de dados de uso*, no qual o agente emprega as características de consumo armazenadas para a representação de modelos de turista.

Assim que *modelos de turista estão extraídos*, ele entra no estado *criando modelos de grupo de turistas através de algoritmos de mineração*, sendo que é esse agrupamento que possibilita a classificação do turista, permitindo a recomendação de pacotes turísticos por meio de filtragem colaborativa.

Quando os *modelos de grupo de turistas estão disponíveis*, o agente passa para o estado *elaborando estatísticas turísticas*, onde são calculadas porcentagens de aceitação de cada pacote turístico.

Logo que as *estatísticas turísticas estão elaboradas*, o ela transita para o estado *aguardando o modelo do turista corrente vindo do agente Modelador*. Uma vez chegado tal modelo, o agente entra no estado *classificando o turista corrente*, em que o turista é classificado em um grupo de turistas com perfil similar ao seu, através de algoritmos de agrupamento dinâmico.

Feita a classificação, um *grupo com perfil similar ao do turista corrente fica disponível* e, então, o agente retorna para o estado *aguardando o modelo do turista corrente vindo do agente Modelador*.

A partir de tal modelo outros turistas correntes podem ser classificados, saindo dele para um estado diverso somente se o *tempo de espera tiver expirado*, hipótese em que o agrupamento é repetido com base em um repositório mais atualizado.

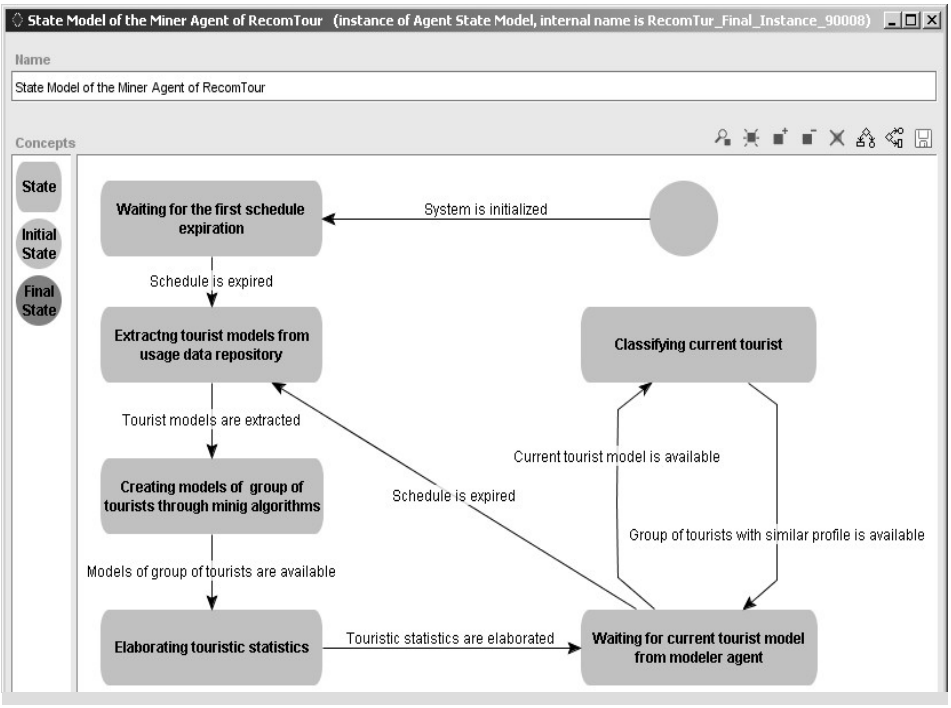


Figura 5.39: Modelo dos Estados do Agente *Minerador*

5.3.6 Projeto do agente *Gerenciador*

Nesta quinta iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Gerenciador da RecomTour (Figura 5.40), que é formado pelos *Modelos do Conhecimento e das Atividades do Agente* Gerenciador, sendo um para cada uma de suas duas responsabilidades, e pelo *Modelo dos Estados do Agente* Gerenciador.

Referente a tal agente, não se dispunha de qualquer padrão de projeto detalhado o qual pudesse facilitar a sua modelagem, que, por isso, realiza de forma direcionada ao presente desenvolvimento.

Figura 5.40: Modelo do Agente *Gerenciador* da *RecomTour*

5.3.6.1 Modelagem do conhecimento e das atividades do agente Gerenciador

A Figura 5.41 e a Figura 5.42 mostram os dois *Modelos do Conhecimento e das Atividades do Agente* Gerenciador da RecomTour, referentes ao *gerenciamento de pacotes turísticos* e à *exibição de pacotes turísticos*, respectivamente, que são as duas responsabilidades desse agente. Tais modelos representam e detalham de maneira isolada o contexto de execução dessas responsabilidades, sendo que a primeira delas está decomposta nas atividades de *cadastro de pacotes turísticos,* de *alteração de pacotes turísticos* e de *manutenção de pacotes turísticos*.

O *cadastro de pacotes turísticos* tem como pré-condição que um *novo pacote turístico seja especificado* e como pós-condição que o *modelo de pacote turístico esteja disponível*.

Sendo para isto requerida como destreza o *modelo de matrizes de características*. No exercício dessa atividade há o uso de *informações sobre pacotes turísticos* e a produção do *modelo de pacote turístico*, este de conhecimento do agente *Gerenciador*.

Para a *alteração de pacotes turísticos*, é suficiente que um *pacote turístico antigo tenha sofrido mudanças*, sendo isto necessário para que um *modelo de pacote turístico esteja disponível*, sendo para tal requerido o *modelo de matrizes de características*. No exercício dessa atividade é usado o *repositório de pacotes turísticos* para gerar um *modelo de pacote turístico*, ambos de conhecimento do agente *Gerenciador*.

A atividade *manutenção de pacotes turísticos* tem como pré-condição que um *modelo de pacote turístico esteja disponível* e como pós-condição que o *pacote turístico esteja armazenado*.

Esta atividade mantém um *repositório de pacotes turísticos*, utilizando o *modelo de pacote turístico* e requerendo com destreza para isso os *grafos semânticos em RDF*.

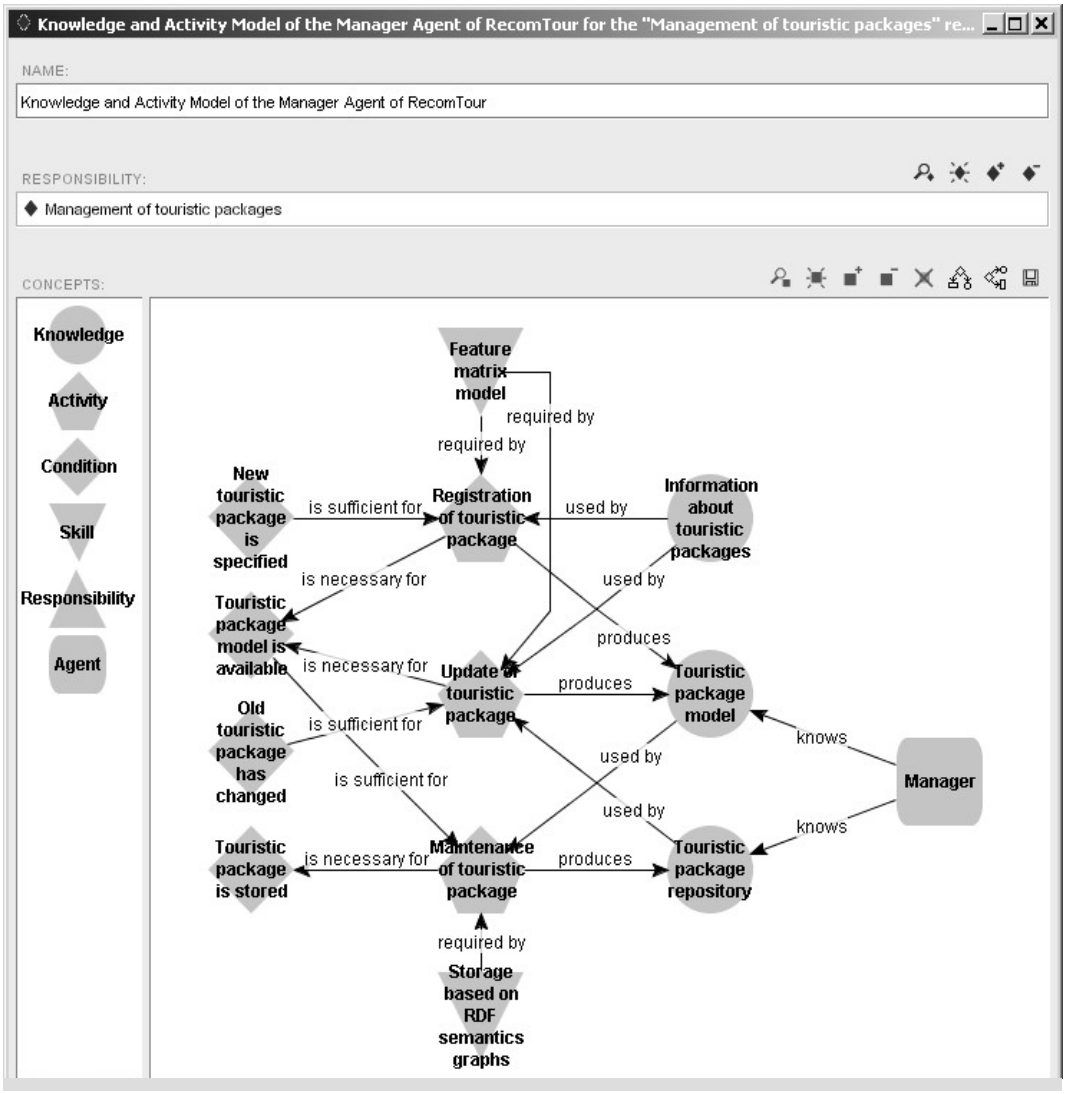


Figura 5.41: Modelo do Conhecimento e das Atividades do Agente *Gerenciador* para a responsabilidade *gerenciamento de pacotes turísticos*

Em relação à *exibição de pacotes turísticos* é suficiente que uma *agência de turismo está conectada ao sistema*, sendo isso necessário para que *pacotes turísticos sejam listados*, sendo tal feito através da *listagem ordenada de informações*.

Para a execução dessa responsabilidade são usadas as *estatísticas sobre escolhas de pacotes turísticos*, de conhecimento do agente *Minerador*, e o *repositório de pacotes turísticos*, para produzir a *lista de pacotes turísticos com estatísticas*, os dois últimos conhecidos pelo *Gerenciador*.

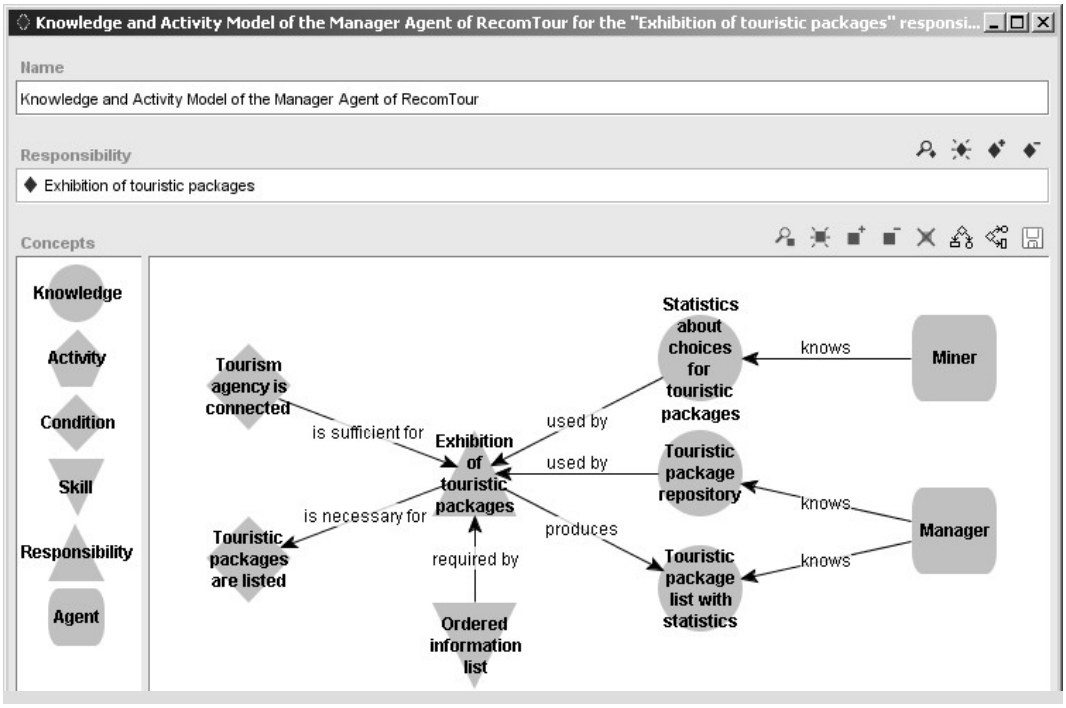


Figura 5.42: Modelo do Conhecimento e das Atividades do Agente *Gerenciador* para a responsabilidade *exibição de pacotes turísticos*

5.3.6.2 Modelagem dos estados do agente Gerenciador

A Figura 5.43 exibe o *Modelo dos Estados do Agente* Gerenciador da RecomTour, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Desse jeito, o agente inicia seu ciclo de vida no momento em que uma *agência de turismo se conecta* ao sistema, assumindo o estado *exibindo a lista dos pacotes turísticos com estatísticas* anunciados pela agência de turismo conectada. Neste estado, o agente recolhe informações sobre pacotes turísticos, as quais podem constituir um *novo pacote turístico especificado*, levando ao estado *cadastrando pacote turístico*, agência, um *pacote turístico antigo modificado*, fazendo transitar para o estado *atualizando pacote turístico*.

Tanto no caso do cadastro quanto no da atualização, o agente passa em seguida para o estado *mantendo pacotes turísticos*. Então, quando o *modelo de pacote turístico está gravado*, o agente retorna para o estado *exibindo a lista dos pacotes turísticos com estatísticas*, no qual permanece até que a *agência de turismo se desconecte* do sistema.

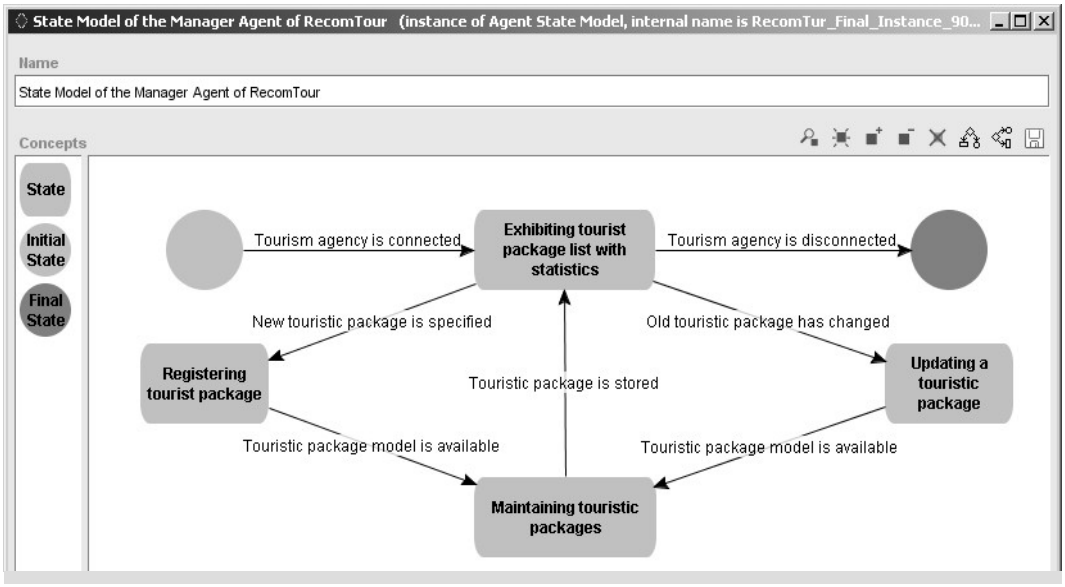


Figura 5.43: Modelo dos Estados do Agente *Gerenciador*

5.3.7 Modelagem do conhecimento da sociedade multiagente

Nesta terceira subfase, é produzido o *Modelo do Conhecimento da Sociedade Multiagente* da RecomTour (Figura 5.44), sendo nada mais que a base para a construção da ontologia de comunicação dos agentes, reunindo os conceitos carregados nas mensagens trocadas entre eles.

Então, no caso da recomendação de pacotes, os cinco agentes irão se comunicar mediante a troca de sete conhecimentos: *escolhas do turista*, *modelo do turista*, *repositório de dados de uso*, *grupo do turista*, *modelo de recomendação*, *estatísticas turísticas* e *repositório de pacotes turísticos*.

As *escolhas do turista* recaem sobre *pacotes turísticos* de interesse dele, os quais são representados por *modelos de pacotes turísticos*, que por sua vez possuem a *identificação do pacote* e as *características do pacote turístico* – sendo tais o *destino*, o *transporte*, a *hospedagem*, as *preferências*, o *valor* e as *atividades* – e ficam armazenados em um *repositório de pacotes turísticos*.

Um *repositório de dados de uso* mantém as *escolhas dos turistas*, que são parte do *modelo do turista*. Este possui a *identificação do turista*, que é dada por um *nome de usuário* e uma *senha*, sendo classificado em um *grupo de turistas*, o qual tem um *número de grupo* e uma *cardinalidade de grupo*.

Um *modelo de recomendação* é baseado em um *grupo de turista* e indicas um conjunto de *pacotes turísticos* que são recomendáveis ao turista. Por fim,

estatísticas turísticas associam as *identificações dos pacotes turísticos* às respectivas *porcentagens de aceitação* de cada um deles.

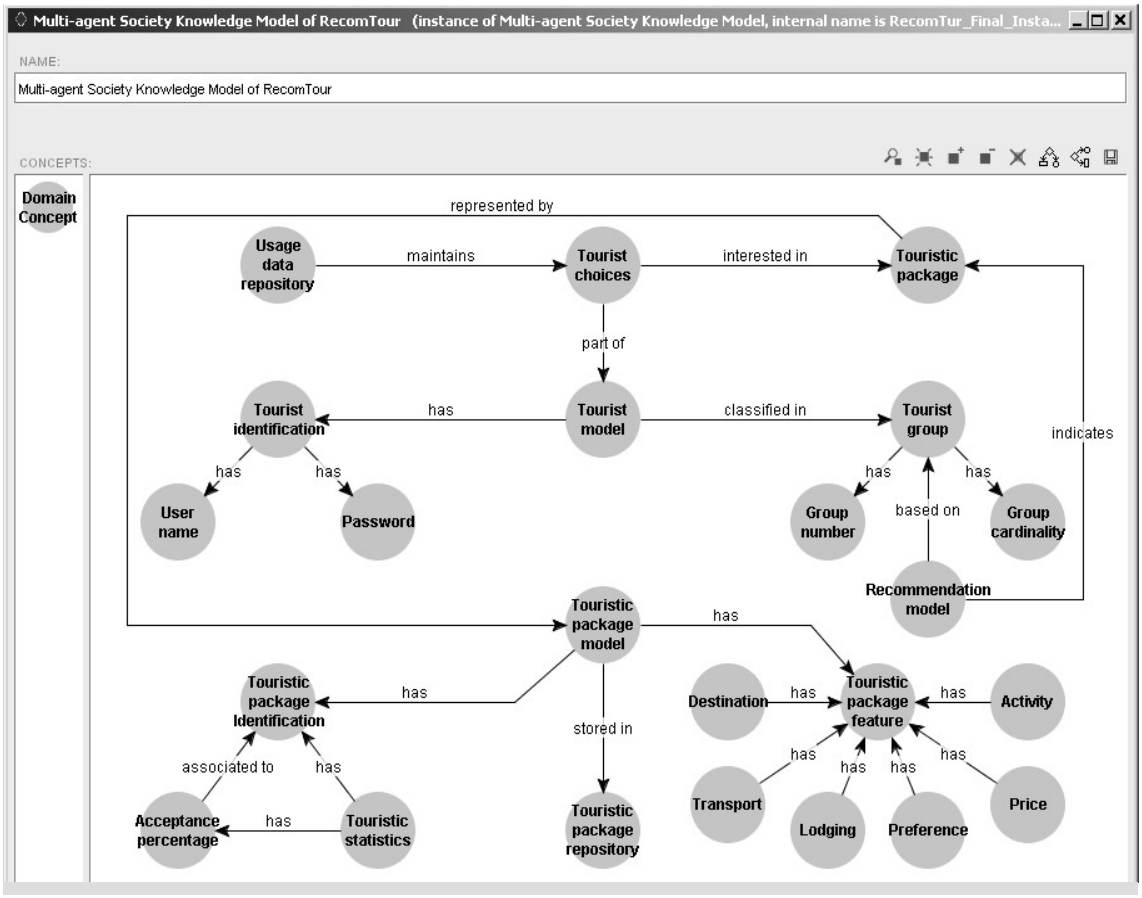


Figura 5.44: Modelo do Conhecimento da Sociedade Multiagente da *RecomTour*

Uma vez concluídas todas as subfases e todos os passos da fase de projeto da aplicação, chega-se à implementação, em que haverá o mapeamento dos elementos aqui modelados para a elaboração de modelos relativos a uma tecnologia particular.

5.4 Implementação da aplicação *RecomTour*

Na fase de Implementação da Aplicação, de acordo com a metodologia MAAEM, resta ao desenvolver elaborar o *Modelo de Implementação da Aplicação* (Figura 5.45), o qual é composto por dois subprodutos: o *Modelo de Agentes e Comportamentos* e o *Modelo de Atos de Comunicação entre Agentes*. Nas próximas subseções, é pormenorizadamente demonstrada a confecção desses modelos na implementação da RecomTour.

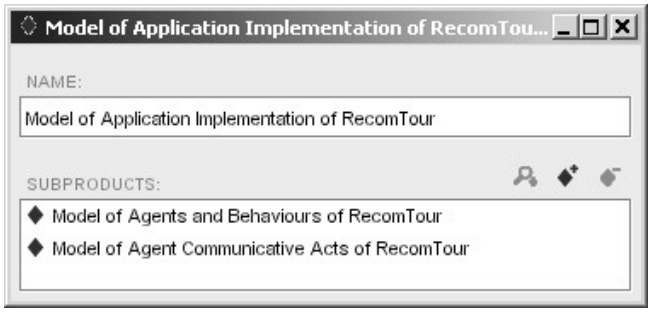


Figura 5.45: Modelo de Implementação da Aplicação *RecomTour*

5.4.1 Mapeamento de agentes do projeto à implementação e de responsabilidades em comportamentos

A Figura 5.46 mostra o *Modelo de Agentes e Comportamentos* da RecomTour, o qual relaciona instâncias das classes *agente* e *comportamento* da plataforma JADE, a tecnologia de implementação adotada, identificadas a partir dos *agentes* e *responsabilidades* projeto, respectivamente.

Assim, foram instanciados os agentes *Interfaceador*, *Modelador, Aquisitor*, *Minerador* e *Gerenciador*. E, de acordo com as responsabilidades por eles assumidas, procedeu-se a instanciação dos comportamentos de cada um, os quais apresentam diversos tipos, conforme detalhado a seguir.

O *Interfaceador* desempenha três comportamentos: *monitorar escolhas do turista*, que é *cíclico*, pois permanece em constante estado de captura das informações sobre as escolhas do turista após ele ter entrado no sistema; *buscar pacotes turísticos*, que é *seqüencial* e tem como subcomportamentos *representar consulta*, que é *simples*, pois é executado sempre que um turista especifica uma consulta e *comparação e recuperação*, que também é simples, porque é executado a cada vez que há um consulta representada; e *oferecer pacotes turísticos*, que é *simples*, posto que é disparado quando há um modelo de recomendação disponível;

O *Modelador* fica encarregado de dois comportamentos: *modelar turista corrente*, que é *simples*, já que é executado cada vez que sejam repassadas informações sobre escolhas; e *construir modelo de recomendação*, que também é *simples*, uma vez que entra em execução sempre que um grupo esteja disponível.

Em seguida, o *Aquisitor* cuida do comportamento *manter dados de uso*, que é atômico, porque é executado uma única vez quando um turista sai do sistema para que sejam armazenados no repositório os dados referentes às escolhas dele.

Depois disso, o *Minerador* executa os três seguintes comportamentos: *descobrir padrões de consumo*, que é *seqüencial*, tendo como subcomportamentos *extrair modelos de turista*, que é *cíclico*, pois é executado dentro de intervalos de tempos pré-definidos, e *aplicar algoritmos de mineração*, que é simples, já que é executado sempre que modelos de turistos são extraídos; *classificar o turista corrente*, que é *simples*, pois é executado cada vez que um modelo de turista corrente esteja disponível; e *elaborar estatísticas turísticas*, que é *simples*, visto que é executado quando modelos de grupo de turistas estejam disponíveis.

Finalmente, ao *Gerenciador* são atribuídos dois comportamentos: *gerenciar pacotes turísticos* e, que é *composto* e tem como subcomportamentos *cadastrar pacote turístico*, que é *simples,* já que é executado sempre que uma agência de turismo especifica um novo pacote turístico, *alterar pacote turístico*, que também é *simples*, pois é executado quando uma agência de turismo muda um pacote turístico antigo e *manter pacotes turísticos,* que igualmente é *simples,* visto que é executado sempre que há um modelo de pacote turístico disponível; e *exibir pacotes turísticos*, que é atômico, pois é unicamente quando uma agência se conecta ao sistema.

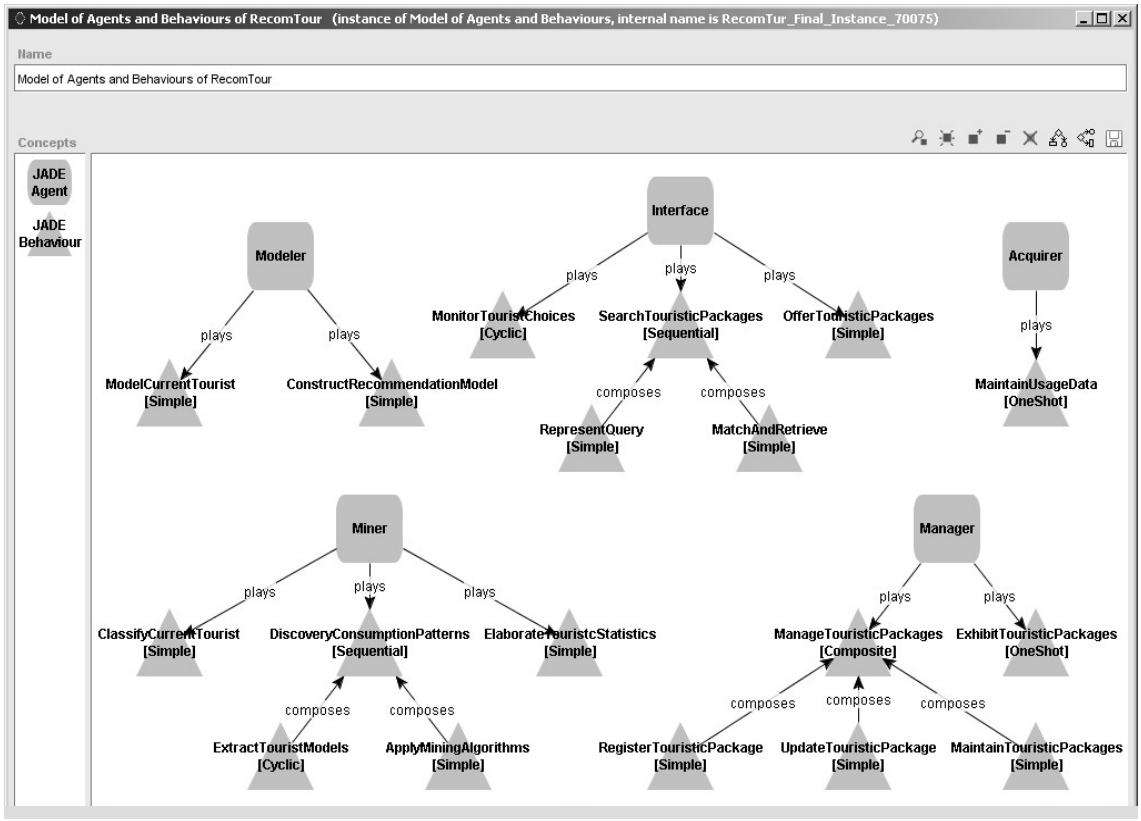


Figura 5.46: Modelo de Agentes e Comportamentos da *RecomTour*

Realizada essa parte da implementação, tendo sido modelados todos os agentes e comportamento nos termos do JADE, resta apenas modelar os atos de comunicação entre os agentes.

5.4.2 Mapeamento de interações entre agentes em atos de comunicação

A Figura 5.47 traz o *Modelo de Atos de Comunicação entre Agentes* da RecomTour, que exibe a forma como o conhecimento da sociedade multiagente é trocado entre os agentes por meio das mensagens que uns enviam para outros.

Em primeiro lugar, uma mensagem "*1-INFORM_REF (escolhas do turista)"* é remetida pelo agente Interfaceador ao *Modelador*, a qual carrega um objeto que contém as informações acerca do consumo de pacotes turísticos pelo turista corrente. Tais informações serão utilizadas para a modelagem do turista corrente, que será usado posteriormente no processo de classificação do turista e de manutenção do repositório de dados de uso.

Em seguida, tendo sido executada a tarefa do *Modelador*, este envia uma mensagem "*2-INFORM_REF (modelo do turista)*" ao agente *Minerador*, contendo o modelo de turista corrente recém construído para que, efetuado a criação dos modelos de grupo de turistas, haja a classificação do turista corrente. Uma mensagem "*6-INFORM_REF (modelo do turista)*" também é enviada ao *Aquisitor*, a fim que este possa manter atualizado o repositório de dados de uso, através do armazenamento das informações sobre as escolhas dos turistas.

O *Minerador* envia uma mensagem "*3-QUERY_REF (repositório de dados de uso)*" ao *Aquisitor*, solicitando o conteúdo do repositório de dados de uso, o qual será usado pelo na extração de modelos de turista.

Posteriormente, ao término da execução de sua tarefa, o *Minerador*, remete uma mensagem "*4-INFORM_REF (grupo do turista)*" para o *Modelador*, a fim de que ele construa um modelo de recomendação personalizado. O *Minerador* pode ainda receber a qualquer tempo uma mensagem *"7-QUERY_REF (estatísticas turísticas)"* do *Gerenciador,* solicitando dados estatísticos a serem exibidos juntamente com a lista de pacotes turísticos de uma certa agência de turismo.

Na seqüência, o *Modelador* manda uma mensagem *"5-INFORM_REF (modelo de recomendação)"* para o *Interfaceador*, no sentido de que este ofereça as recomendações ao turista.

Enfim, quando um turista especifica uma consulta por pacotes turísticos, o *Interfaceador* passa uma mensagem "*8-QUERY_REF (repositório de pacotes turísticos)"* para o *Gerenciador*, solicitando os pacotes turísticos mantidos no repositório para sejam utilizados em função da consulta especificada.

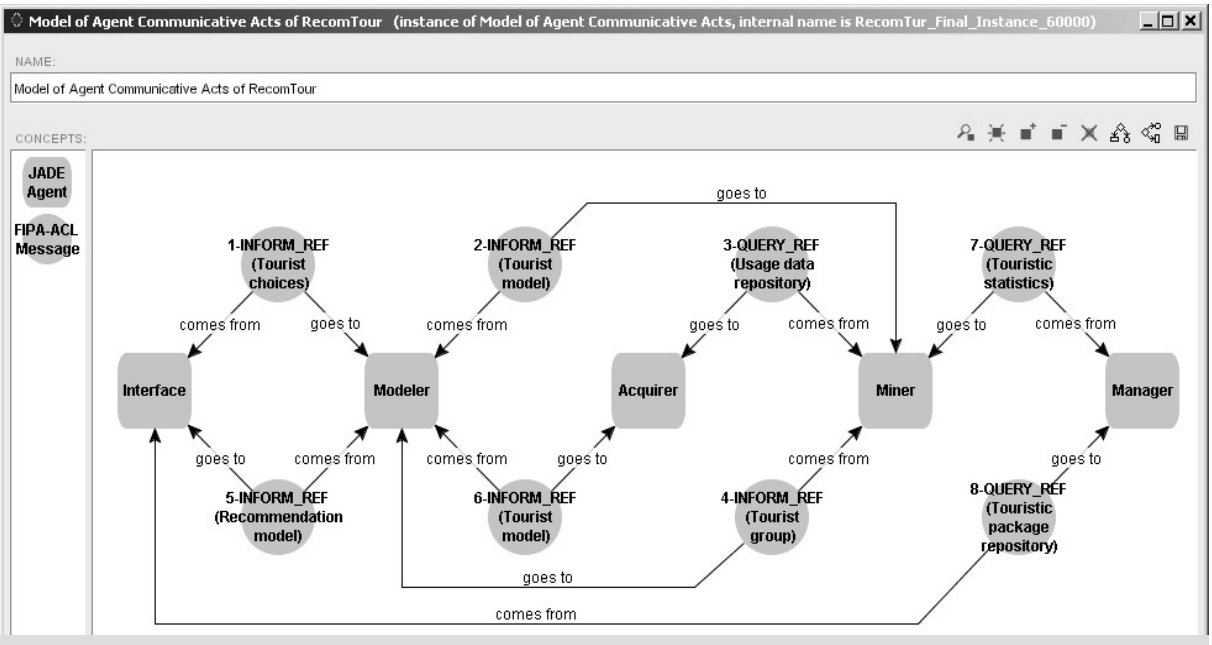


Figura 5.47: Modelo de Atos de Comunicação entre Agentes da *RecomTour*

Nesse ponto, findam-se as três fases do desenvolvimento da RecomTour, estando elaborados todos os modelos necessários para o término da construção da aplicação.

5.5 Considerações finais

Neste capítulo foi descrito o primeiro estudo de caso elaborado com o intuito de avaliar a metodologia MAAEM e a ontologia ONTORMAS. Tal estudo de caso fez o reuso de artefatos de software orientados a agentes anteriormente desenvolvidos, representados na ONTOWUM, os quais são voltados para o problema da Mineração de Uso na Web. Como resultado teve-se modelada a aplicação RecomTour.

Assim, de início, delineou-se em que consistiria a mencionada aplicação, indo-se desde sua contextualização até um apanhado preliminar de seus requisitos, passando-se por uma breve fundamentação teórica.

Após isso, entrando-se propriamente na avaliação da referida metodologia, foram seguidas as orientações concernentes às fases de análise, projeto e implementação da aplicação, a partir das quais foram produzidos os respectivos modelos.

A partir dessa avaliação, nota-se que a MAAEM facilitou a modelagem dessa aplicação multiagente, uma vez que se seguindo as fases e os passos propostos por tal metodologia, pôde-se, sem muita dificuldade, utilizar os conceitos já modelados na ONTOWUM, procedendo-se a adequção deles aos requisitos particulares do domínio turístico explorado. Isso deve-se, sem dúvida alguma, à adoção de ontologias para a representação do conhecimento de tal metodologia, funcionando, ao mesmo tempo, com ferramenta de modelagem e repositório de modelos.

As únicas dificuldades encontradas dizem respeito ao estabelecimento das pré e pós-condições, uma vez que algumas delas se mostram de fato apenas na forma de transições quando da elaboração dos modelos de estados, bem como à determinação do conhecimento da sociedade multiagente, visto que isto se torna mais intuitivo após a construção do modelo de atos de comunicação. Sugerir-se-ia uma reformulação dos passos envolvidos para melhorar a modelagem quanto a isso.

6 ESTUDO DE CASO SEM REUSO: *InfoNorma* – Uma Aplicação Multiagente para a Entrega Personalizada de Instrumentos Jurídico-Normativos através da Filtragem de Informação baseada no Conteúdo

6.1 Considerações iniciais

Para demonstrar a aplicabilidade da metodologia MAAEM e da ontologia ONTORMAS ao desenvolvimento de aplicações multiagente mesmo quando não é possível a reutilização de artefatos de software provenientes da Engenharia de Domínio Multiagente, foi concebido o presente estudo de caso, o qual consiste na construção da *InfoNorma* – Uma Aplicação Multiagente para a Entrega Personalizada de Instrumentos Jurídico-Normativos através da Filtragem de Informação baseada no Conteúdo.

Essa experiência aborda todas as fases da MAAEM, iniciando-se pela análise e especificação de requisitos da aplicação, passando pelo projeto arquitetural da sociedade multiagente e pelo projeto detalhado de cada agente, cujo produto é a arquitetura da aplicação, a qual, encerrando o processo, é convertida em um projeto dependente de uma determinada tecnologia de implementação orientada a agentes, que pode, então, ser prototipado e codificado na forma da aplicação final.

6.1.1 Contexto da aplicação: o domínio jurídico

Quaisquer grupos de indivíduos podem ser classificados segundo uma enorme variedade de critérios, tais como: idade, gênero, origem, classe econômico-social, interesses etc. Este último em particular, é caracterizado, por exemplo, pela natureza das informações que satisfazem razoavelmente todos os membros do grupo.

No contexto da aplicação ora proposta, o grupo de usuários a que ela se destina é essencialmente composto por acadêmicos de Direito e por profissionais do meio jurídico, os quais se interessam, dentre muitos outros tópicos, pelo que se pode denominar genericamente de instrumentos jurídico-normativos, sendo eles, além da própria constituição, as leis, os decretos, as resoluções, as medidas provisórias etc. É justamente o conhecimento sempre atualizado dessas informações que permite a esses sujeitos bem desempenhar as atividades inerentes a suas áreas de atuação.

Tal interesse, porém, não recai apenas sobre as leis e afins atualmente existentes, abrangendo também os novos instrumentos jurídico-normativos que são freqüentemente promulgados, seja versando acerca de matéria inédita seja alterando ou revogando comandos legislativos vigentes. Para tanto, faz-se necessário um contínuo acompanhamento do processo legislativo, através do qual aqueles documentos legais são elaborados por parte dos poderes legiferantes.

Desse modo, há que se selecionar parcelas daquela informação de acordo com os interesses específicos de cada usuário. Assim, seria bem-vindo dispor de um meio através do qual se pudesse definir interesses – por exemplo, em função dos tipos e das categorias de instrumentos jurídico-normativos – e, após isso, tão somente se aguardasse que fossem entregues automaticamente as novas informações na medida em que surgissem. Além disso, dever-se-ia promover a evolução da representação dos interesses dos usuários toda vez que esses se modificassem, de modo a sempre satisfazer suas necessidades de informação.

6.1.2 Requisitos da aplicação

O que a aplicação *InfoNorma* objetiva é promover a filtragem de informações relativas a instrumentos jurídico-normativos e a entrega de maneira personalizada para cada usuário, conforme os perfis de interesse que eles apresentem. Como consequência, resulta facilitado o acesso a tais informações, uma vez que são fornecidas aos usuários as leis, decretos, resoluções, medidas provisórias etc que realmente possam lhes interessar em alguma medida, otimizando o seu aproveitamento.

Desse modo, a *InfoNorma* tem dois requisitos principais. O primeiro diz respeito aos usuários, ou seja, à forma pela qual cada um destes pode especificar explicitamente sua identificação e seus interesses por certos tipos e categorias de instrumentos jurídico-normativos, através da escolha direta dentre opções ou mesmo por meio da listagem livre de palavras-chave, constituindo diversos perfis de usuários. Isso para que, ao final processo, possa-se ter um atendimento personalizado daqueles interesses.

Já o segundo requisito é relativo à informação propriamente considerada, que provém de uma fonte, composta por índices de elementos de informação, a qual deve ser constantemente monitorada em busca de mudanças, isto é, do surgimento

de novos elementos que potencialmente satisfaçam os interesses de cada usuário, ocorrendo um processo de filtragem baseado na comparação do conteúdo desses elementos com as necessidades extraídas dos modelos daqueles usuários.

6.1.3 Fundamentos da aplicação

Como já delineado, a *InfoNorma* visa promover a entrega personalizada de instrumentos jurídico-normativos, de acordo com os interesses de cada usuário, fazendo-o através da filtragem de informação baseada no conteúdo. Diante disso, constatou-se que não havia à disposição do desenvolver nenhum modelo de domínio, framework multiagente ou agentes de software que pudessem ser reutilizados, impondo a construção sem qualquer reuso.

Assim, buscou-se fundamentos para a modelagem da *InfoNorma* em tópicos referentes à modelagem explícita de usuários e à filtragem de informação baseada no conteúdo, ambos resumidamente descritos a seguir.

6.1.3.1 Modelagem explícita de usuários

A modelagem de usuários é uma linha de pesquisa da Engenharia de Software que estuda as formas através das quais as informações dos usuários podem ser adquiridas, representadas e usadas por sistemas computacionais. O modelo de usuário é uma fonte de conhecimento que contém informações, explícitas ou implicitamente adquiridas, de todos os aspectos relevantes do usuário para serem utilizados por uma aplicação de software.

O conhecimento do modelo de usuário é importante, pois através dele se pode facilitar e melhorar a interação do usuário com o sistema, por exemplo, através da adaptação de interfaces ou da geração de recomendações.

Várias técnicas têm sido investigadas para abordar os problemas da modelagem explícita de usuários, algumas parcialmente tomadas de outras áreas de pesquisa e outras desenvolvidas no próprio campo da modelagem de usuário (KOBSA, 1994). Essas técnicas são aplicadas nas três fases do processo de modelagem de usuários: a aquisição; a representação; e a manutenção de modelos de usuário (SERRA JUNIOR, 2001).

Na fase de aquisição, o sistema coleta informações através de entradas fornecidas explicitamente pelo usuário, sendo utilizadas para isso técnicas como questionamentos através de formulários.

As informações coletadas são então processadas na fase de representação, onde são utilizadas técnicas para a representação formal das informações adquiridas sobre o usuário, ou seja, a construção do modelo do usuário.

A fase de manutenção contempla as técnicas para a incorporação de novas informações e atualização das informações existentes no modelo, bem como técnicas para fazer inferências a partir dessas informações.

6.1.3.2 Filtragem de informação baseada no conteúdo

A filtragem de informação é um processo utilizado para satisfazer as necessidades de informação de um usuário em longo prazo a partir de fontes de informação dinâmicas e desestruturadas (BURKE, 2000).

No processo de filtragem de informação baseado em conteúdo são feitas comparações entre as descrições dos itens de informações e os modelos de usuário. Poder-se-ia ter ainda modelos de usuário sendo atualizados com base na avaliação que o usuário faz de elementos de informação recebidos. Desta maneira, seriam filtrados itens similares a outros que o usuário tenha avaliado positivamente no passado.

O funcionamento básico de um sistema de filtragem de informação pode ser descrito da seguinte forma: primeiramente, cria modelos de usuário a partir de suas preferências, que podem ser capturadas de forma implícita, observando-se o seu comportamento, ou de forma explícita, especificados através de formulários; em seguida, são criadas as representações internas dos novos elementos de informação assim que são detectados; por fim, os elementos de informação são comparados com os modelos de usuário e aqueles com maior índice de similaridade são recomendados.

Existem várias técnicas para a representação interna dos elementos de informação, tais como: o modelo de espaço vetorial; as ontologias; as árvores de decisão etc, e para cada um existem técnicas para calcular a similaridade, como o coeficiente de Pearson e a distância angular entre vetores.

6.2 Análise da aplicação *InfoNorma*

Na fase de Análise da Aplicação, conforme estabelecido pela metodologia MAAEM, a preocupação do desenvolvedor é produzir a *Especificação da Aplicação* (Figura 6.1), a qual é composta por quatro subprodutos: o *Modelo de Conceitos*; o *Modelo de Objetivos*; o *Modelo de Papéis*; e o *Modelo de Interações entre Papéis*. Nas próximas subseções, é compassadamente descrita a elaboração desses modelos na análise da InfoNorma.

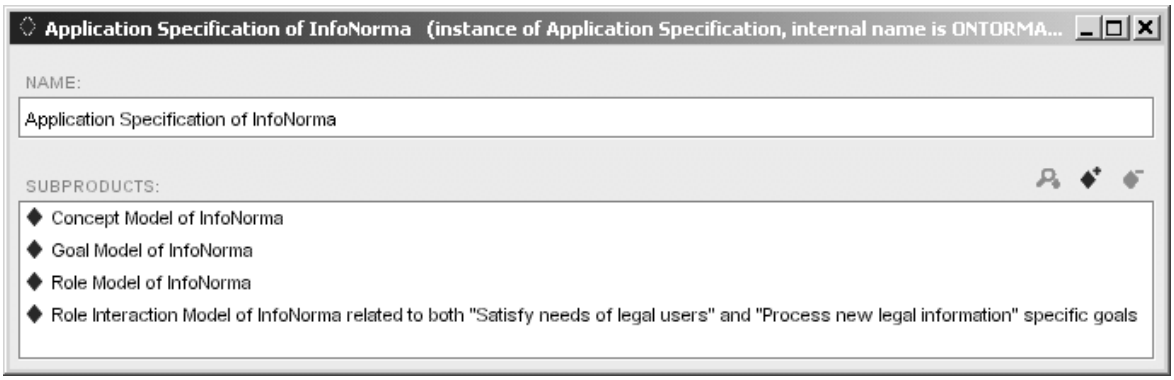


Figura 6.1: Especificação da Aplicação *InfoNorma*

6.2.1 Modelagem de conceitos

A Figura 6.2 adiante ilustra o *Modelo de Conceitos* da InfoNorma, cuja principal utilidade no processo de desenvolvimento da aplicação é servir como um primeiro momento para se levantar as idéias mais importantes relativas à aplicação, o que é expresso na forma de conceitos semanticamente relacionados.

Em relação à InfoNorma, sendo uma aplicação que visa oferecer informações selecionadas de acordo com os interesses de cada usuário, a primeira idéia que vem à mente se refere a *sistemas de filtragem de informação*, cujo fim é exatamente esse.

Assim, através do provimento de *serviços de filtragem de informação*, tais sistemas filtram *informações relevantes por interesses dos usuários*, e entregam *mensagens de correio eletrônico*, as quais portam as informações relevantes filtradas, sendo recebidas pelos respectivos *usuários* beneficiários.

A filtragem, por sua vez, ocorre através da comparação de *surrogates de interesses de usuários* – inferidos a partir de *modelos de usuário*, os quais

representam *perfis de usuário* – com *surrogates* de *elementos de informação* – que representam *elementos de informação* – sendo ambos construídos pelo sistema, que também monitora a *fonte de informação*.

O *repositório legislativo* é um tipo de *fonte de informação* e contêm *instrumentos jurídico-normativos*, um tipo de *elementos de informação*, os quais se caracterizam por possuir um *tipo* e uma *categoria*, além de ser deles que se constituem as *necessidades de informação em longo prazo*, que são parte dos *perfis de usuário* juntamente com a *identificação*.

Por fim, *usuários de leis* , sendo um tipo de *usuário*, têm interesse especial por *instrumentos jurídico-normativos*.

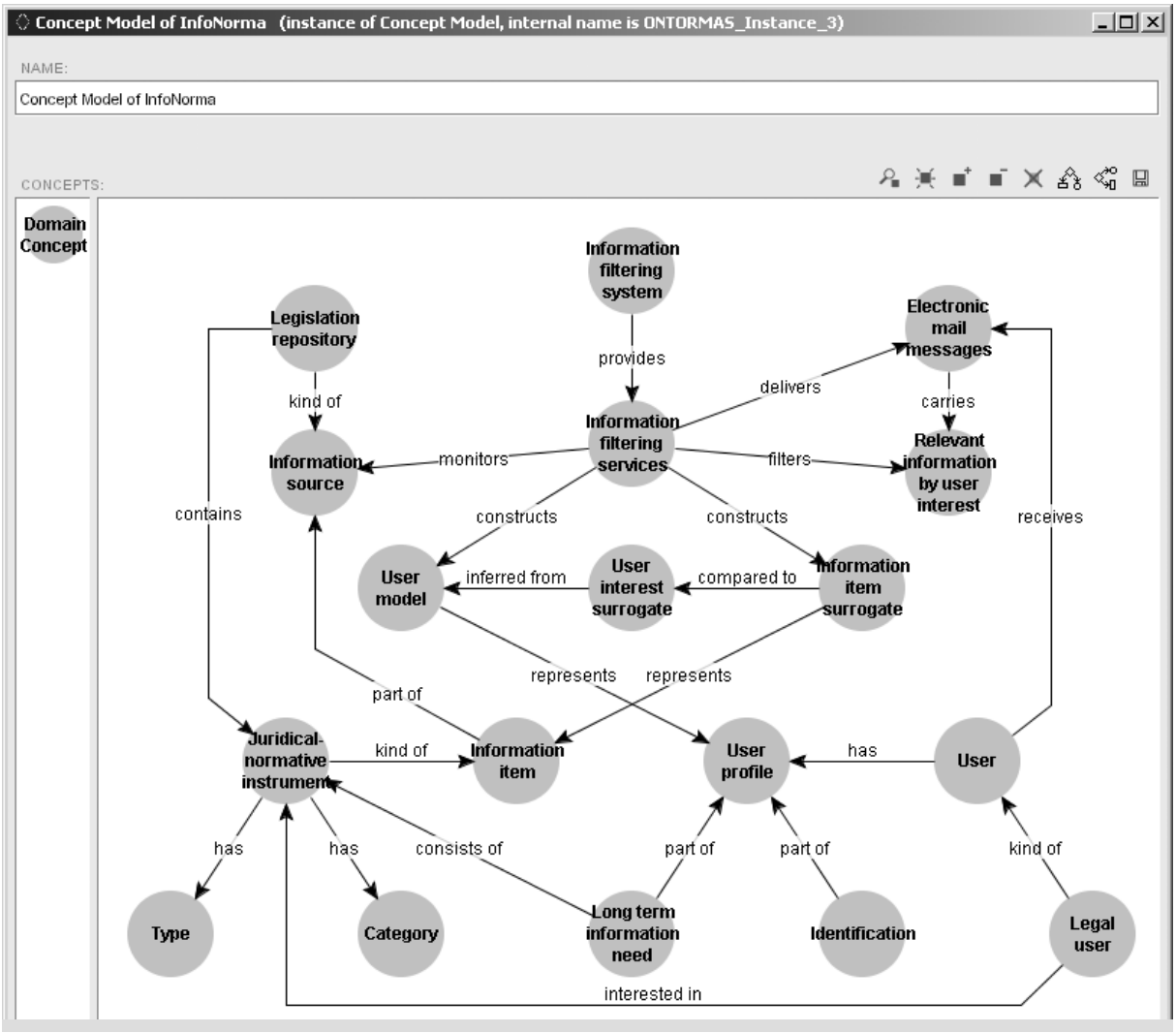


Figura 6.2: Modelo de Conceitos da *InfoNorma*

É o entendimento inicial de todos esses conceitos e relacionamentos – os quais, diga-se, não constituem um rol exaustivo, mas apenas exemplificativo – que embasa todo o restante da modelagem, como se poderá constatar nos demais passos descritos a seguir.

6.2.2 Modelagem de objetivos

A Figura 6.3 mostra o *Modelo de Objetivos* da InfoNorma, trazendo uma hierarquia de objetivos – geral e específicos – e de responsabilidades, que correspondem às funcionalidades esperadas da aplicação, sendo que uns permitem o alcance dos outros. Constam também desse modelo entidades externas que interagem destinatárias da aplicação ou com ela envolvidas.

De início, tem-se que o objetivo geral da aplicação é *prover informações jurídico-normativas personalizadas*. Fundado nos conceitos e relacionamentos identificados no passo anterior, procedeu-se à decomposição daquele objetivo geral nestes dois objetivos específicos, que a ele conduzem: *satisfazer necessidades de usuários de leis* e *processar novas informações legais*.

Duas entidades externas se relacionam com esses objetivos específicos. Uma é o *usuário*, de quem a modelagem obtém os dados necessários e que, por sua vez, recebe as informações filtradas. A outra é a *fonte de informação*, que fornece os subsídios para o processo de filtragem de informação.

Passando ao nível das responsabilidades, tem-se a *aquisição de perfis de usuário* e *manutenção de modelos de usuário* promovendo o alcance do objetivo de *satisfazer necessidades de usuários de leis*.

Já para se *processar novas informações legais* com êxito, são demandadas como responsabilidades o *monitoramento da fonte de informação*, a *representação dos elementos de informação*, a *comparação e análise de similaridade* e a *entrega de informações filtradas*.

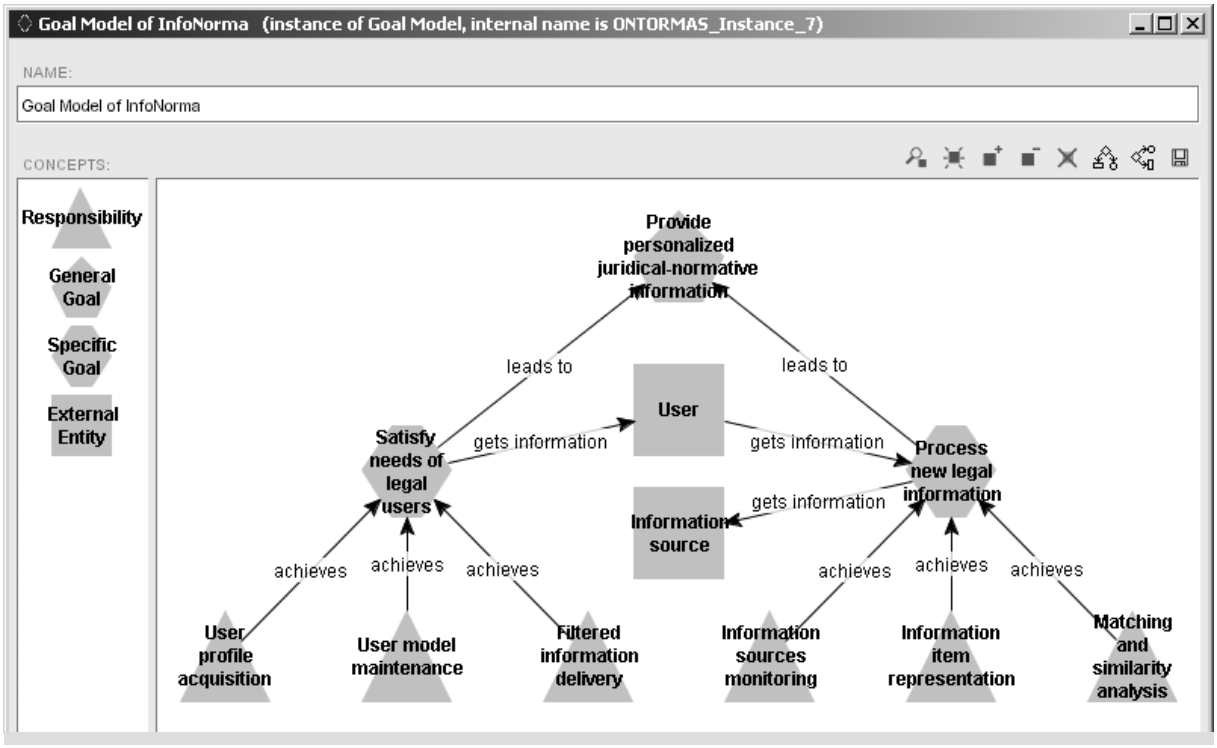


Figura 6.3: Modelo de Objetivos da *InfoNorma*

Depois de estabelecidos os objetivos da aplicação, bem como as correspondentes responsabilidades, pode-se, então, prosseguir com o desenvolvimento, posto que é a partir destas que se derivará os respectivos papéis no passo posterior.

6.2.3 Modelagem de papéis

A Figura 6.4, a Figura 6.5 e a Figura 6.6 exibem as partes do *Modelo de Papéis* da InfoNorma, o qual apresenta como principal função a atribuição das responsabilidades anteriormente identificadas a papéis, sendo que estes, no seu efetivo desempenho, envolvem o uso e a produção de determinados conhecimentos, o atendimento de pré-condições e de pós-condições, além de requererem destrezas específicas para esse fim. Pode haver casos em que uma responsabilidade seja abrangente demais, devendo ela ser decomposta em duas ou mais atividades, ressaltando-se que estas têm a mesma natureza daquela.

A primeira responsabilidade é a *aquisição de perfis de usuário*, sendo encarregado o papel *interface de entrada*, que usa como a *identificação e necessidades de informação do usuário* e produz *tipos e categorias de instrumentos*

jurídico-normativos. A sua pré-condição é que um *novo perfil de usuário seja especificado*, sendo requeridas *técnicas para a aquisição de perfis de usuário explicitamente*, e tendo como pós-condição que o *perfil do usuário esteja validado*.

Já a segunda responsabilidade, que é a *manutenção de modelos de usuário*, atribuída ao papel *modelador de usuário*, devido a sua complexidade, é exercida através de duas atividades. Uma delas é a *criação do modelo do usuário*, que faz uso do conhecimento produzido pelo papel precedente, bem como tem a pós-condição deste como sua pré-condição, gerando como produto o *modelo do usuário* mediante *técnicas para criação de modelos de usuário* e resultando na existência de um *novo modelo de usuário criado* como pós-condição.

A outra atividade, a *extração de interesses do usuário*, guiada por *técnicas para representação de interesses de usuários*, usa tais modelos para produzir *surrogate de interesses de usuários*, bem como *surrogates de elementos de informação*, quando ocorridas as pré-condições de *interesses de usuários terem sido requisitadas* e de *modelos de usuário terem sido criado*s, o que repercute em que *interesses de usuários tenham sido extraídos* como pós-condição.

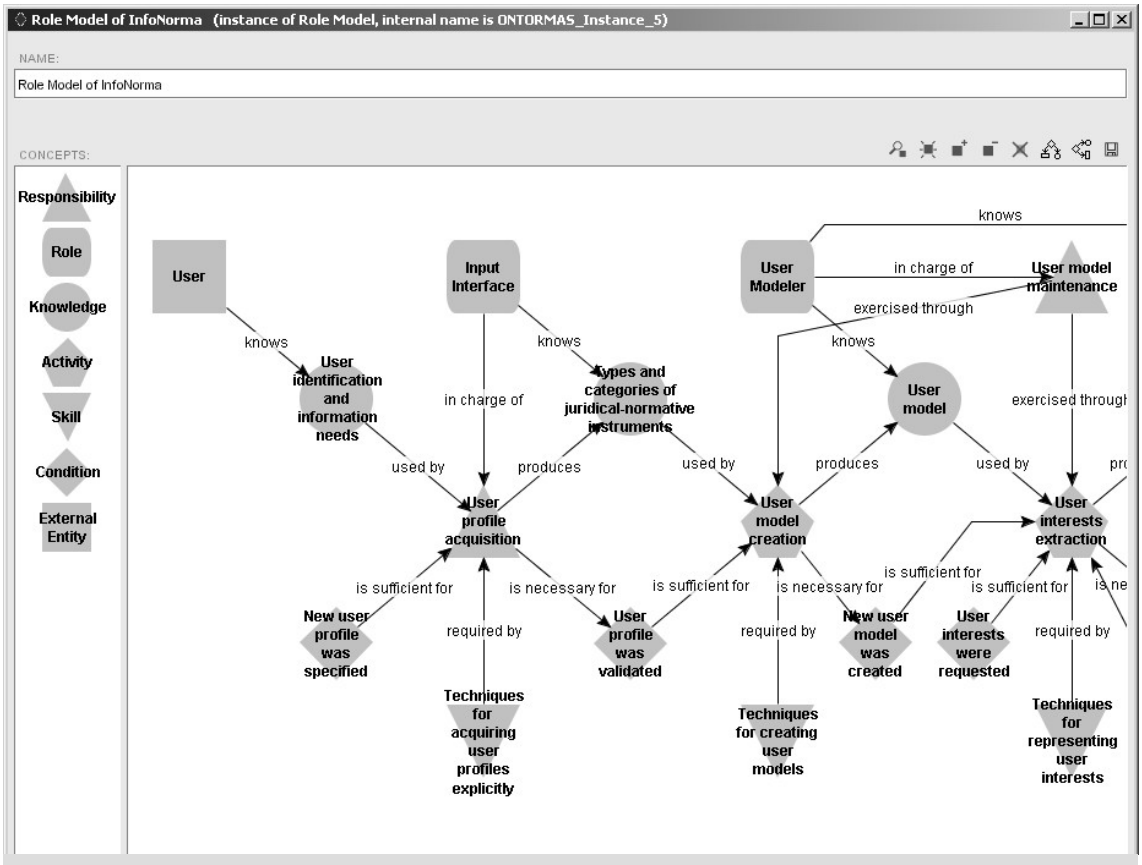


Figura 6.4: Primeira parte do Modelo de Papéis da *InfoNorma*

A terceira responsabilidade, consistente no *monitoramento da fonte de informação*, tem como responsável o papel *monitor de fonte*, o qual usa as *mudanças na fonte de informação*, atendida a pré-condição de que um dos seus *índices de informação tenha sido atualizado*, para originar *novos elementos de informação*, com base em *técnicas para o monitoramento da fonte de informação*, resultando na pós-condição de que *novos elementos de informação foram detectados*.

A quarta responsabilidade, que se refere à *representação dos elementos de informação*, é incumbência do papel *construtor de representação*, que cria *surrogates de elementos de informação* que tenham advindo do papel antecedente, o que faz por meio de *técnicas para representação de elementos de informação*, se atendida a sua pré-condição, que coincide com a pós-condição imediatamente anterior.

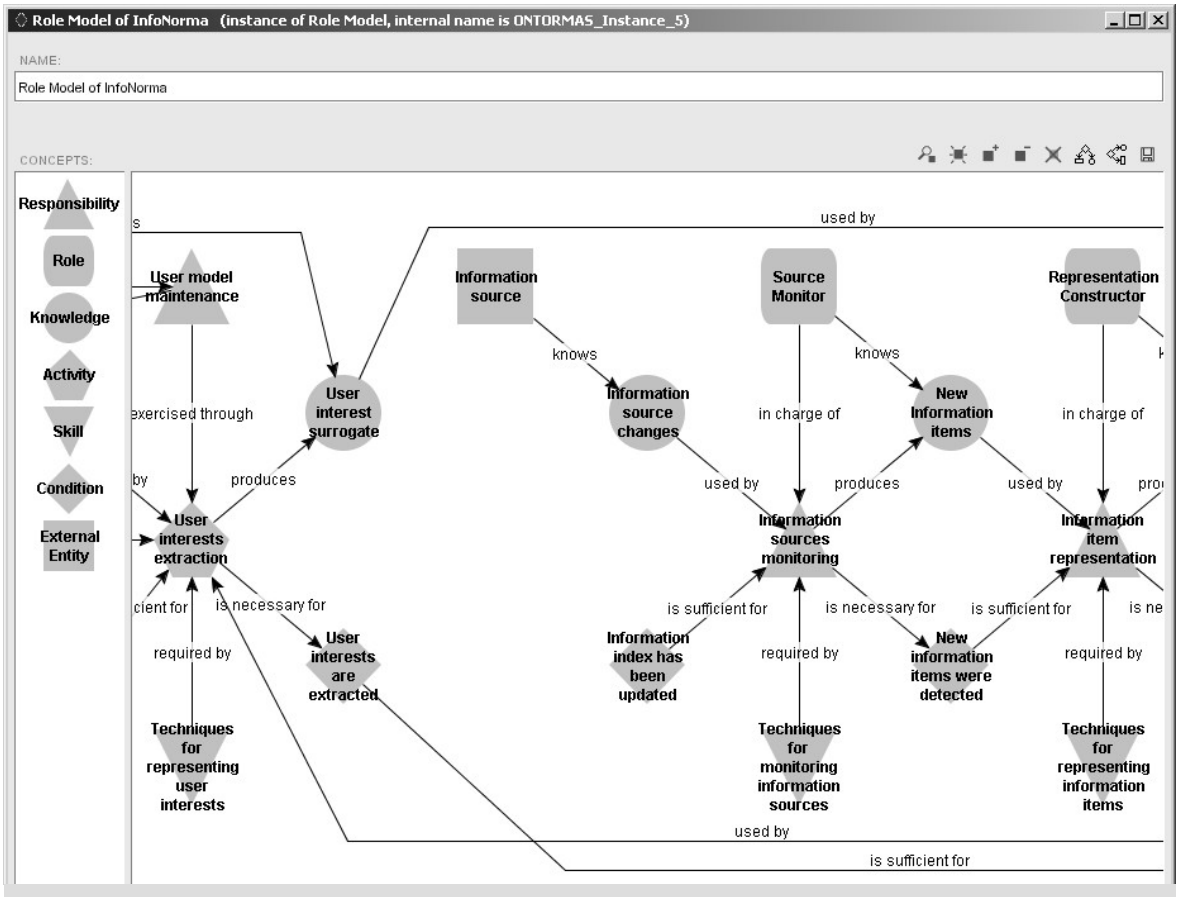


Figura 6.5: Segunda parte do Modelo de Papéis da *InfoNorma*

A quinta responsabilidade, que diz respeito à *comparação e análise de similaridade*, tem o papel *filtrador de informação* como encarregado, que produz os *elementos de informação filtrados* usando os *surrogates de elementos de informação* e os *surrogates de interesses de usuários*, nos termos de *técnicas para filtragem de informação baseada no conteúdo*, sendo pré-condições para tanto que *surrogates de elementos de informação estejam disponíveis* e *interesses de usuários tenham sido extraídas*, que são ambas pós-condições de responsabilidades já descritas.

A última responsabilidade, relativa à *entrega de informações filtradas*, cujo encargo é do papel *interface de saída*, que usa os resultados da filtragem e produz *mensagens de satisfação de informação*, tendo como pré-condição que *elementos de informação tenham sido filtrados* e como pós-condição que *mensagens de satisfação de informação tenham sido enviadas*, e sendo necessárias, no caso, *técnicas para entrega de informações*.

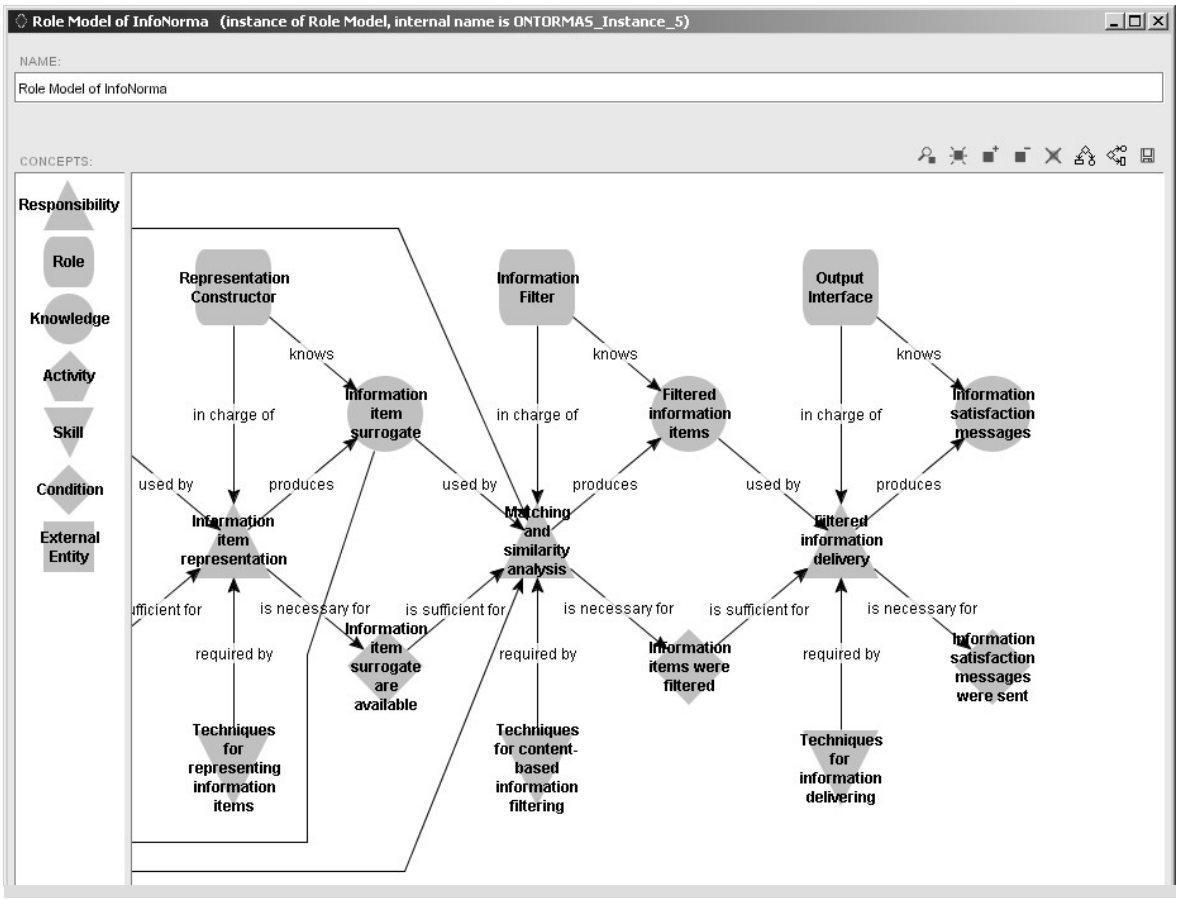


Figura 6.6: Terceira parte do Modelo de Papéis da *InfoNorma*

Os papéis, juntamente com os demais conceitos de modelagem que aparecem acima, representam uma visão estática da aplicação, sendo esse apenas

um de seus aspectos. Assim, a sua dinâmica é definida através de interações entre papéis como se verá a seguir.

6.2.4 Modelagem de interações entre papéis

A Figura 6.7 traz o *Modelo de Interações entre Papéis* da InfoNorma, relativo aos objetivos específicos *satisfazer necessidades de usuários de leis* e *processar novas informações legais*, sendo um para todos devido à simplicidade do problema, o qual ilustra de maneira seqüencial como os papéis e as entidades externas da aplicação interagem, através da troca de mensagens invocando responsabilidades ou atividades, bem como indicando que conhecimentos são passados como parâmetros.

Assim, a primeira interação é "*1 : aquisição de perfis de usuário (identificação e necessidades de informação do usuário)*", ocorrendo entre o usuário e a interface de entrada e consistindo no início do funcionamento da aplicação, já que para haver a filtragem é necessário que pelo menos um usuário manifeste algum interesse.

Em seguida, a segunda interação é "*2 : criação do modelo do usuário (tipos e categorias de instrumentos jurídico-normativos)*", sendo através da qual o papel de interface de entrada solicita ao modelador de usuário que elabore os devidos modelos, a serem mantidos para uso posterior.

De outro lado, a terceira interação é "*3 : monitoramento da fonte de informação (mudanças na fonte de informação)*", partindo da entidade externa fonte de informação para o papel monitor de fonte. Considera-se como uma mudança qualquer atualização de um dos índices da fonte que implique na adição de novos elementos de informação.

Logo após, a quarta interação é "4 *: representação dos elementos de informação (novos elementos de informação)*", acontece quando, de posse de elementos de informação recém-detectados, o monitor de fonte pede ao construtor de representação que estruture tais elementos da forma como a aplicação precisa para filtrá-los.

Depois, a quinta interação é "*5 : comparação e análise de similaridade (surrogates de elementos de informação)*", sendo a própria filtragem de informação e ocorrendo logo que os surrogates de elementos de informação estejam prontos, mas

não antes que o papel filtrador de informação receba do modelador de usuário os interesses extraídos de cada modelo de usuário, quando são comparadas umas com as outras e é quantificada a medida em que ambas se assemelham.

Prosseguindo, a sexta interação é "*6 : extração de interesses do usuário (surrogates de elementos de informação)*", consistindo justamente na solicitação que o filtrador de informação faz ao modelador de usuário referida no parágrafo anterior.

Como resultado, a sétima interação, "*7 : entrega de informações filtradas (elementos de informação filtrados)*", dando-se entre o papel filtrador de informação e a interface de saída, visa apenas a passagem dos elementos de informação filtrados com a indicação dos respectivos usuários a que se destinam.

Por fim, a oitava interação é "8 *: visualização (mensagens de satisfação de informação)*", sendo concluída com a visualização por parte do usuário das mensagens vindas da interface de saída, as quais contêm informações que potencialmente o satisfazem, segundo a filtragem realizada.

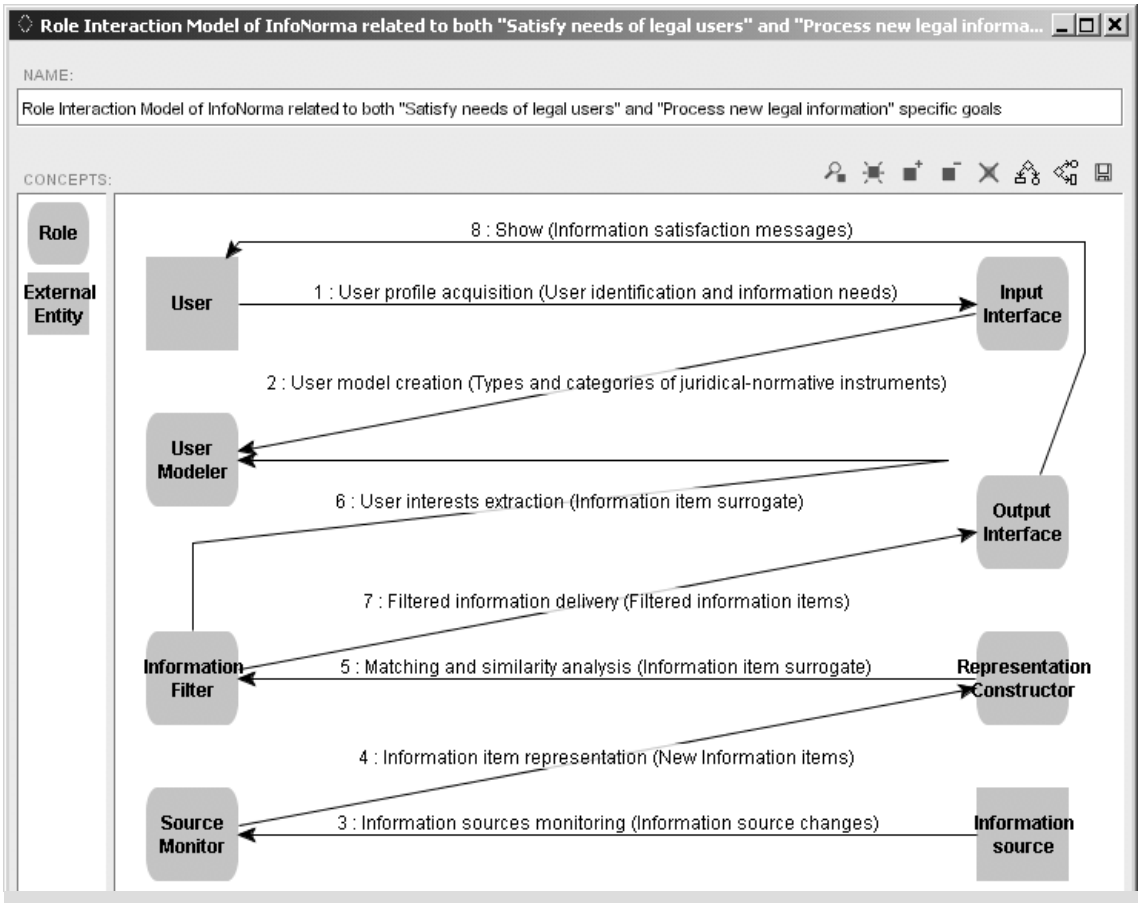


Figura 6.7: Modelo de Interações entre Papéis da *InfoNorma* referente aos objetivos específicos *satisfazer necessidades de usuários de leis* e *processar novas informações legais*

Uma vez estabelecidas as interações entre papéis, encerra-se a fase de análise da aplicação. Os próximos passos, pertencentes ao projeto, são todos baseados no refinamento dos modelos aqui elaborados.

6.3 Projeto da aplicação *InfoNorma*

Na fase de Projeto da Aplicação, segundo o passo a passo prescrito pela metodologia MAAEM, o desenvolvedor busca construir a *Arquitetura da Aplicação* (Figura 6.8), a qual é composta por três subprodutos: o *Modelo da Arquitetura*; o *Modelo do Agente*, sendo um para cada agente; e *Modelo do Conhecimento da Sociedade Multiagente*. Nas próximas subseções, é cuidadosamente mostrada a elaboração desses modelos no projeto da InfoNorma.

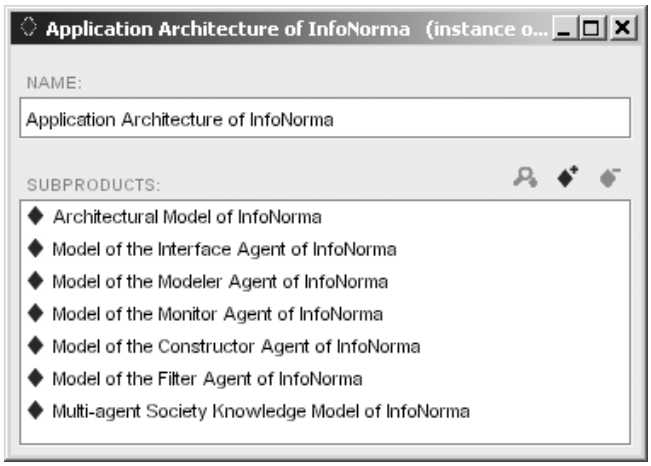


Figura 6.8: Arquitetura da Aplicação *InfoNorma*

6.3.1 Projeto da arquitetura

Nesta primeira subfase, é produzido o *Modelo da Arquitetura* da InfoNorma (Figura 6.9), que inclui o *Modelo da Sociedade Multiagente*, o *Modelo das Interações entre Agentes* e o *Modelo dos Mecanismos de Cooperação e Coordenação*.

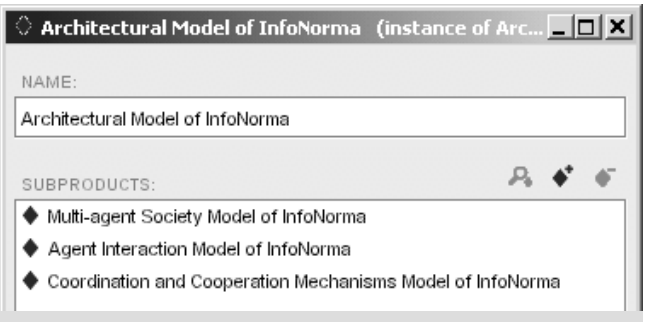


Figura 6.9: Modelo da Arquitetura da *InfoNorma*

6.3.1.1 Modelagem da sociedade multiagente

A Figura 6.10, a Figura 6.11 e a Figura 6.12 ilustram as partes do *Modelo da Sociedade Multiagente* da InfoNorma, que se diferencia do Modelo de Papéis por adotar como abstração principal os agentes, que são capazes de assumir vários papéis durante sua execução. Assim, faze-se o mapeamento de um ou mais destes para um daqueles, de acordo com a afinidade que haja entre as respectivas responsabilidades. Além disso, as destrezas requeridas são refinadas, tornando-se soluções concretas para os problemas dos quais cada agente é encarregado.

Dessa forma, os papéis de *interface de entrada* e de *interface de saída* passam a ser ambos desempenhados pelo agente *Interfaceador*, posto que ambos interagem com uma mesma entidade externa, que é o *usuário*. Quanto aos demais papéis, que são o *modelador de usuários*, o *monitor de fonte*, o *construtor de representação* e o *filtrador de informação*, ficam todos convertidos, respectivamente, nos agentes *Modelador*, *Monitor*, *Construtor* e *Filtrador*.

No projeto, para cada responsabilidade ou atividade, é selecionada uma destreza específica dentre aquelas que foram levantadas genericamente na fase de análise. Assim, em vez de várias técnicas que se prestam em potencial ao fim esperado, é definido um modo particular de se realizar a ação atribuída ao agente.

Com isso, para a *aquisição de perfis de usuário*, foi prevista a necessidade de uma *técnica para a aquisição de perfis de usuário explicitamente*. Optou-se pela forma explícita para se adquirir a identificação e as necessidades de informação dos usuários pelo fato de não haver interações suficientes entre eles e o sistema, inviabilizando fazê-lo implicitamente. Isso porque, sendo uma aplicação de filtragem com base no conteúdo da informação, têm-se usuários que sabem o que

desejam, porém a informação não é disponível num momento atual, restando aguardar que ela seja filtrada num futuro próximo.

Dessa maneira, a escolha recaiu sobre o *preenchimento de formulários em páginas da Web* como suporte para essa responsabilidade. A idéia é que cada usuário possua uma conta individual acessível por meio de uma senha, através da qual ele possa incluir, alterar, excluir ou visualizar seu respectivo perfil, que, uma vez adquirido pela aplicação, deverá ser modelado e então empregado na filtragem das informações que o satisfaçam. Tal aquisição efetivamente ocorre na inicialização da aplicação e sempre que o seu agente de interface constate modificação – decorrente de inclusão, alteração ou exclusão de dados – na situação do perfil, mediante frequentes verificações do correspondente registro, refletindo-se tudo no modelo do usuário.

Quanto à *manutenção de modelos de usuário*, que envolve a *criação do modelo do usuário* e a *extração de interesses do usuário*, pensou-se, a princípio em se adotar uma *técnica para criação de modelos de usuário* e uma *técnica para representação de interesses de usuários*, no entanto, sem se considerar os detalhes de uma e de outra. Tal decisão retardada para a fase de projeto não acarretaria qualquer prejuízo para a perfeita modelagem dos papéis.

Assim, para a *criação de modelos de usuário*, deu-se preferência para a *formação de pares de tipo e categoria*, já que os usuários especificam suas necessidades por determinados instrumentos jurídico-normativos justamente informando à aplicação certos tipos e categorias desses documentos que venham a lhes satisfazer. Então, a modelagem de usuários consiste essencialmente em formar pares válidos entre tais tipos e categorias de acordo com o conhecimento de um especialista no domínio legal e associá-los a identidade de cada usuário, para que a aplicação filtre informações com base neles.

Já para a *extração de interesses de usuários*, foi eleita a *seleção de categorias com base em tipos*, isto é, havendo novos elementos de informação devidamente representados e, portanto, aptos a serem filtrados, é preciso se dispor também dos interesses de usuários relacionadas com os tipos de cada um daqueles elementos de informação. Esses interesses nada mais são que conjuntos de categorias por usuário para cada elemento de informação, ou seja, listas de ramos jurídicos relacionados com o tipo indicado.

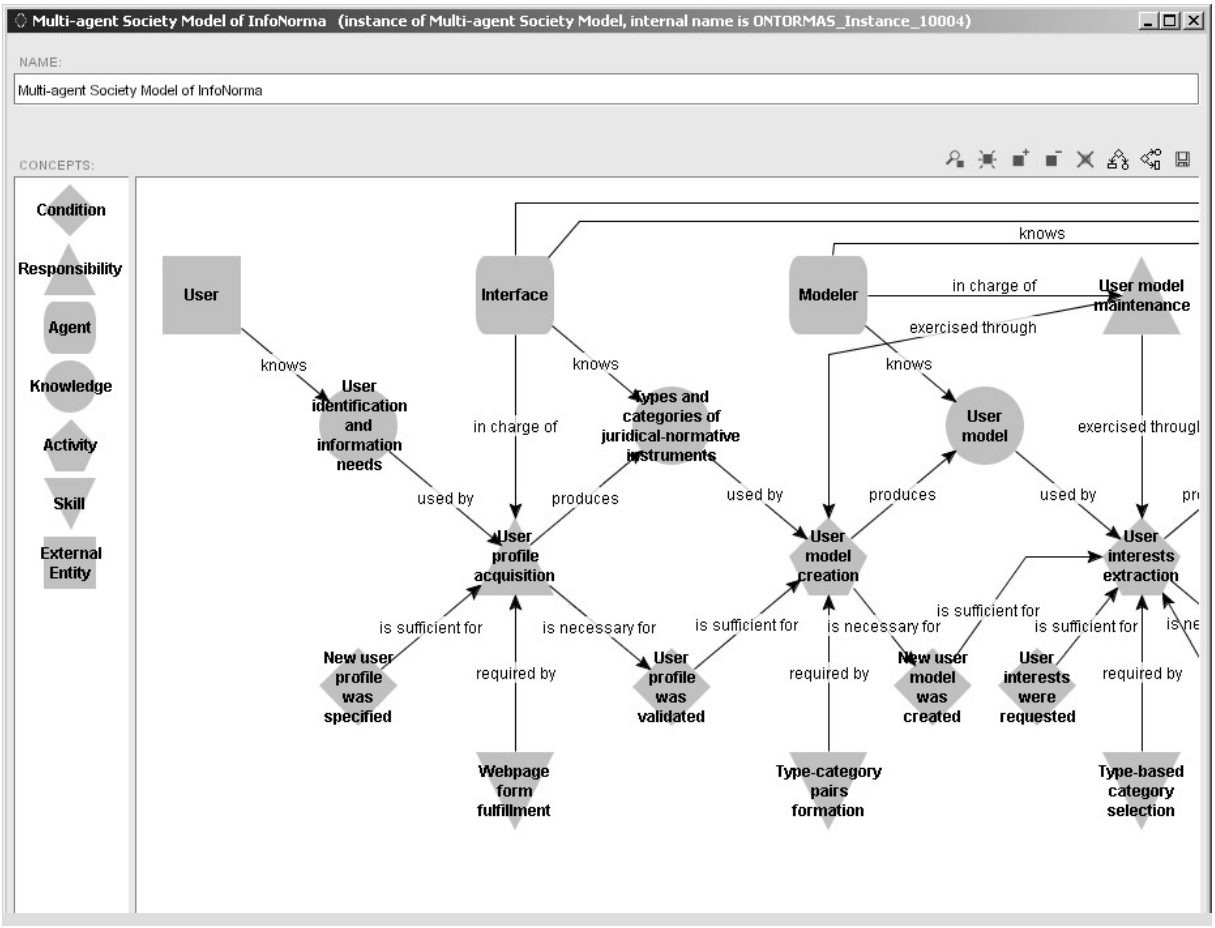


Figura 6.10: Primeira parte do Modelo da Sociedade Multiagente da *InfoNorma*

O *monitoramento da fonte de informação* é suportado, de modo geral por uma *técnica para o monitoramento da fonte de informação*, como forma de se perceber quando mudanças tenham ocorrido nela para que providências sejam tomadas.

Assim, a *detecção de atualizações em índices de informação* foi a saída encontrada, posto que a fonte tem como componentes centrais os índices que relacionam todos os elementos de informação nela contidos, sendo uma atualização representada pelo acréscimo de elementos a essa relação. Tal pode ser obtido procedendo-se a constantes checagens dos índices e se tendo um ponteiro para o último elemento acrescentado em cada índice na atualização anterior.

Então, como um índice é montado em ordem crescente, percorrendo-o decrescentemente, os elementos encontrados até o ponteiro serão novos e deverão ser considerados como mudanças na fonte de informação. E auxilia ainda nesse processo se observar a data e o horário de modificação do arquivo de texto que

contém o índice, o que previne o seguimento se não houver diferença de tempo em relação à última alteração.

Para a *representação dos elementos de informação*, em um momento inicial, é demandada simplesmente uma *técnica para representação de elementos de informação*. Tal visa conferir uma estruturação aos novos elementos de informação, os quais não se acham, até então, em uma forma adequada para serem manipulados pela aplicação.

Desse modo, mostram-se oportunas a *identificação do tipo do elemento e contagem da freqüência das palavras-chave*. Na primeira parte, a questão é tão somente identificar no próprio elemento o seu tipo, expressamente indicado.

Já quanto à segunda, o que se faz é construir uma tabela de pares "nome-valor", em que "nome" representa uma palavra-chave e "valor" significa o número de vezes que ela aparece no documento. Trabalha-se, nesse caso, com uma lista auxiliar de palavras que devem ser desconsideradas para tais efeitos, como artigos, pronomes, preposições etc.

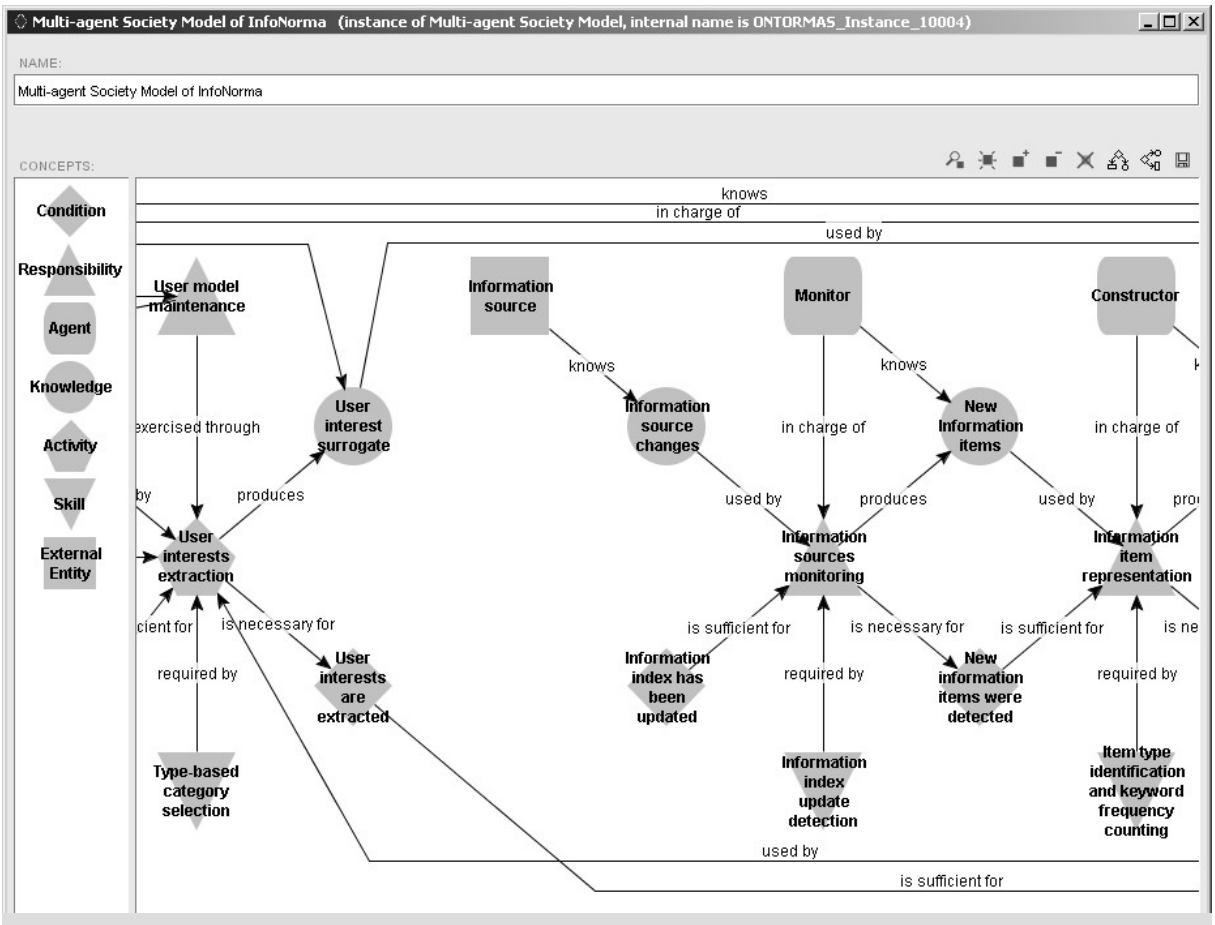


Figura 6.11: Segunda parte do Modelo da Sociedade Multiagente da *InfoNorma*

Como forma de realizar a *comparação e análise de similaridade*, faz-se necessária uma *técnica para filtragem de informação baseada no conteúdo*, que permita comparar os surrogates de elementos de informação e os surrogates de interesses de usuários, além de apurar o grau de semelhança entre ambas.

Ao se adotar para esse fim a *comparação de palavras-chave e cálculo de percentual e peso*, o intuito é, inicialmente, identificar a que categorias pertencem cada elemento de informação, ou seja, checar se existem palavras-chave contidas nos elementos de informação que equivalham a termos-chave associados às categorias listadas nos interesses, sendo um termo-chave composto por uma palavra-chave e um peso.

Em seguida, busca-se saber em que medida o elemento satisfaz um interesse, isto é, descobrir o grau de similaridade dele com a respectiva categoria, o que é feito com base no cálculo do percentual das palavras relacionadas com a categoria que aparecem no elemento, considerando-se, ainda, a ponderação entre palavras mais ou menos importantes no contexto.

Por fim, para a *entrega de informações filtradas*, precisa-se de uma *técnica para entrega de informações* que permita o encaminhamento de cada elemento de informação filtrado ao correspondente usuário, imediatamente após a filtragem e de maneira segura.

Para tanto, optou-se pelo *envio de mensagens de correio eletrônico*, posto que essa tecnologia é amplamente utilizada e implementa a segurança exigida, sendo a melhor solução para a entrega de informações de modo individualizado e a qualquer momento.

Assim, a partir de cada elemento resultante da filtragem por interesse de usuário, constroem-se uma ou mais mensagens de satisfação de informação, que são remetidas imediatamente para o endereço eletrônico apresentado no perfil do usuário.

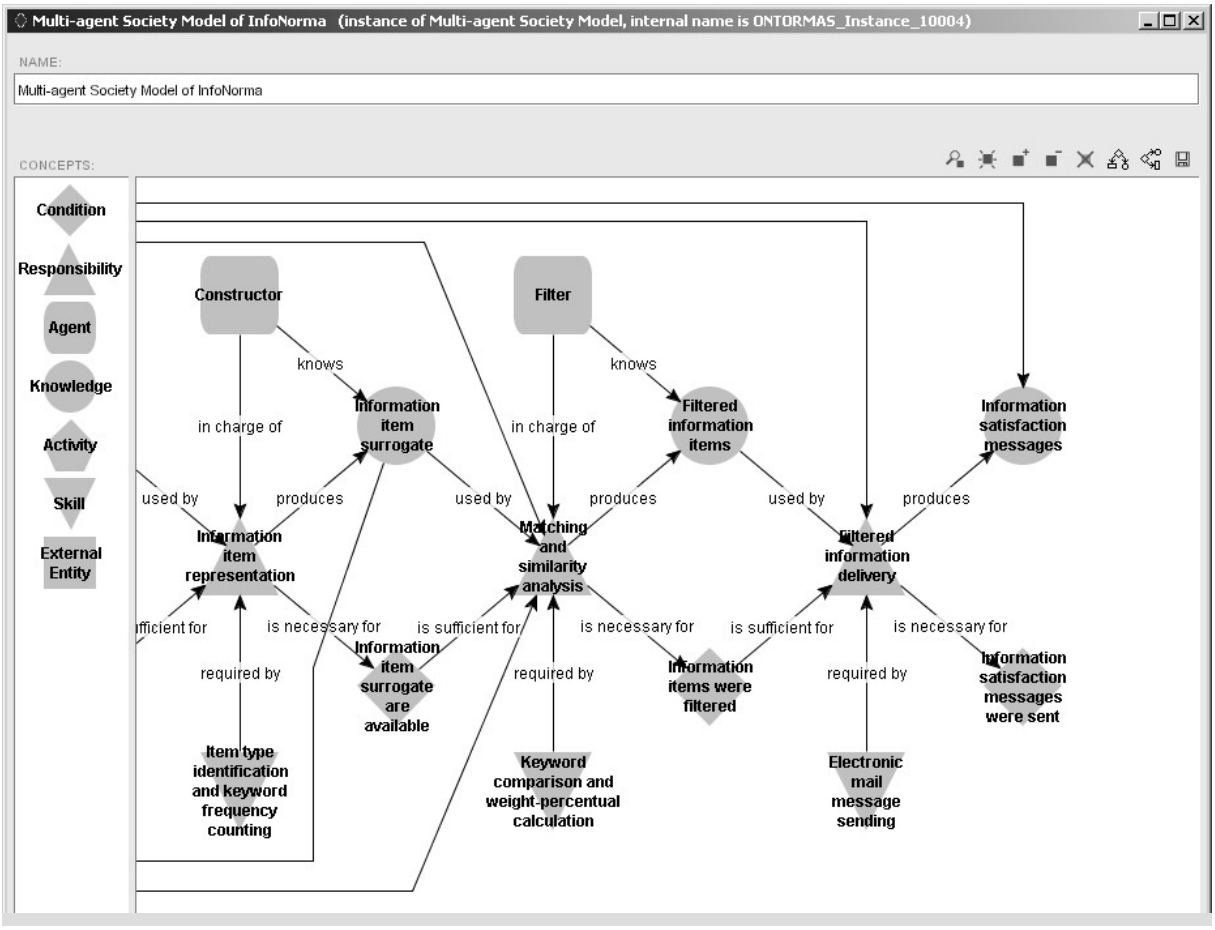


Figura 6.12: Terceira parte do Modelo da Sociedade Multiagente da *InfoNorma*

A conclusão do projeto da sociedade multiagente representa um passo fundamental tanto para o término da arquitetura de maneira geral quanto para o detalhamento de cada agente, conforme será feito a seguir.

6.3.1.2 Modelagem das interações entre agentes

A Figura 6.13 mostra o *Modelo de Interações entre Agentes* da InfoNorma, no qual os agentes e as entidades externas interagem ordenadamente ao longo de suas linhas de vida, seja os primeiros trocando mensagens entre si seja sendo notificados de eventos externos pelas segundas.

Assim, inicialmente ocorre a notificação do evento de que um "*novo perfil de usuário foi especificado*", feita pelo usuário ao agente Interfaceador, sendo essa interação que permite o início do funcionamento da aplicação, posto que sem ao menos um perfil de usuário conhecido isso não é possível.

Em seguida, e como consequência imediata da interação anterior, a mensagem “*INFORM_REF (tipos e categorias de instrumentos jurídico-normativos)*” é passada do agente Interfaceador para o Modelador, para que este crie e mantenha o modelo do perfil recém-adquirido por aquele, de modo a poder ser utilizado futuramente na filtragem de informação para o usuário detendor dele.

Depois disso, havendo um ou mais usuários modelados, resta à aplicação aguardar que mudanças na fonte de informação aconteçam, o que é percebido quando tal entidade notifica o agente Monitor de que um “*índice de informação foi atualizado*”.

Por seu turno, diante da mundança na fonte, o agente Monitor envia a mensagem “*INFORM (novos elementos de informação)*” informando ao Construtor acerca de que ele deve construir representações, para o seguimento do processo.

Uma vez que o agente Construtor tenha realizado sua tarefa, então ele encaminha ao Filtrador, via mensagem “*INFORM_REF (surrogates de elementos de informação)*”, uma parte do subsídio para a filtragem, sendo ela justamente que origina o processo, suspenso até então.

Logo após, mediante tais representações, o agente Filtrador requer ao Modelador, por meio da mensagem “*REQUEST (surrogates de elementos de informação)*”, que sejam extraídas dos modelos de usuário os interesses de cada um que correspondam a tais representações.

Como resposta, o agente Modelador remete ao Filtrador mensagens “*INFORM_REF (*interesse*s de usuários)*” contendo outra parte do subsídio para a filtragem, que é então realizada, resultando em satisfações de informação por usuários.

Dessa feita, resta apenas que o agente Filtrador passe ao Interfaceador mensagens “*INFORM_REF (elementos de informação filtrados)*” para que ele possa providenciar o efetivo envio aos destinatários finais que são os respectivos usuários.

Finalmente, o agente Interfaceador recebe do usuário a confirmação de que as “*mensagens de satisfação de informação foram enviadas*”, quer dizer, aquele toma conhecimento através deste de que houve sucesso na operação que realizou.

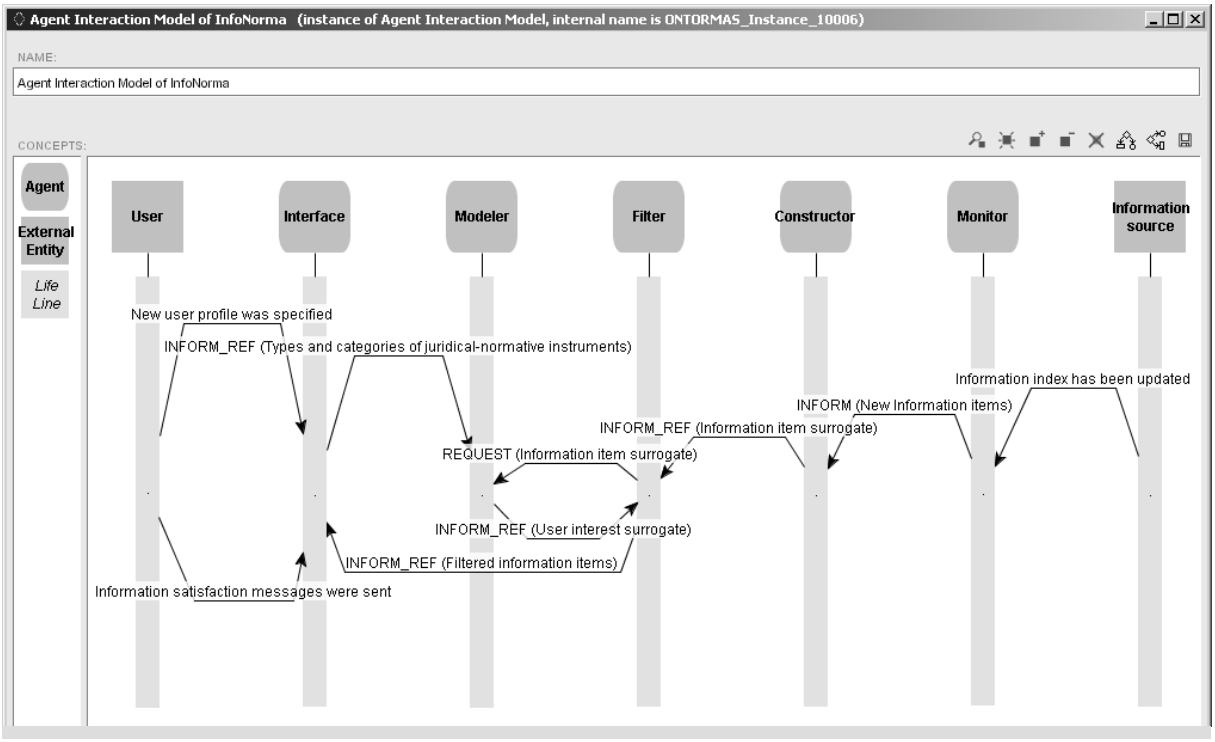


Figura 6.13: Modelo das Interações entre Agentes da *InfoNorma*

Definidas as interações entre agentes, chega-se à hora de organizá-los como uma arquitetura propriamente dita, visto que, até agora, somente se fizera um esboço de tal organização.

6.3.1.3 Modelagem dos mecanismos de cooperação e coordenação

A Figura 6.14 exibe o *Modelo dos Mecanismos de Cooperação e Coordenação* da InfoNorma, que organiza os agentes da aplicação de acordo com requisitos funcionais e não-funcionais e com mecanismos de cooperação e coordenação na forma de uma arquitetura padronizada.

Em face disso, foi adotado o padrão arquitetural *Camada Multiagente* (GIRARDI *et al.*, 2005b), o mesmo já descrito e de uso justificado na Seção 5.3.1.3 do capítulo anterior, que trata do passo correspondente a este naquele outro estudo de caso, aplicando-se aqui todas as considerações lá feitas.

Assim, no nível de abstração "*2*", encontra-se a camada de *processamento de usuários*, da qual fazem parte os agentes Interfaceador e Modelador, sendo que ela interage com o *usuário* para dele receber sua *identificação e necessidades de informação* e a ele entregar *mensagens de*

satisfação de informação, e também com a camada inferior para encaminhar *surrogates de interesses de usuários* e obter *surrogates de elementos de informação* e *elementos de informação filtrados*.

Já no nível de abstração "*1*", situa-se a camada de *processamento de informações*, participando dela os agentes Monitor, Construtor e Filtrador, sendo que ela interage com a *fonte de informação* para perceber *mudanças* que nesta ocorram e com a camada superior, conforme descrito no parágrafo anterior.

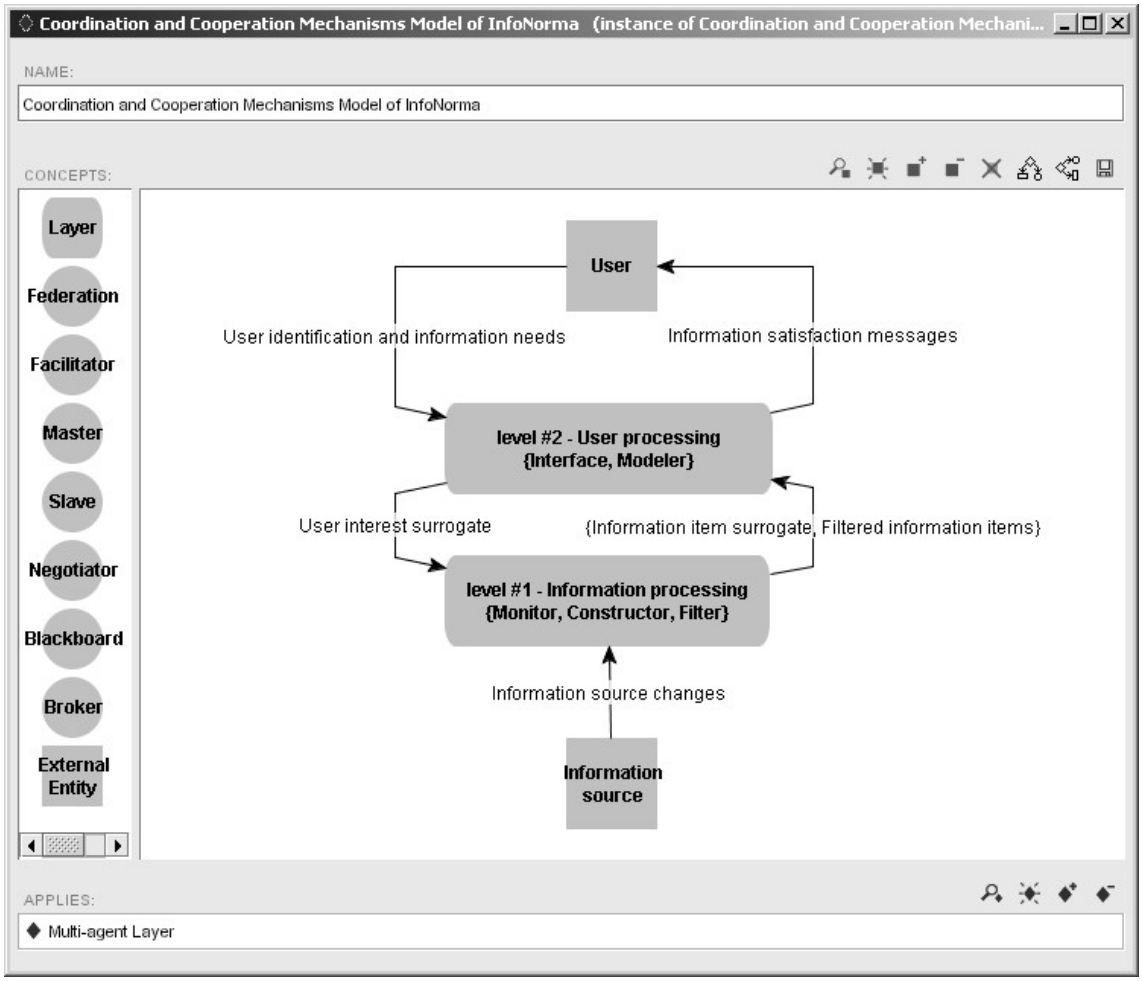


Figura 6.14: Modelo dos Mecanismos de Cooperação e Coordenação da *InfoNorma*

Concluída a arquitetura geral da aplicação, mostrando externamente quais são os seus agentes e como eles se relacionam, passa-se desse ponto em diante a detalhar cada um de tais agentes, nos termos seguintes.

6.3.2 Projeto do agente *Interfaceador*

Nesta primeira iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Interfaceador da InfoNorma (Figura 6.15), que é formado pelos *Modelos do Conhecimento e das Atividades do Agente* Interfaceador, sendo um para cada uma de suas duas responsabilidades, e pelo *Modelo dos Estados do Agente* Interfaceador.

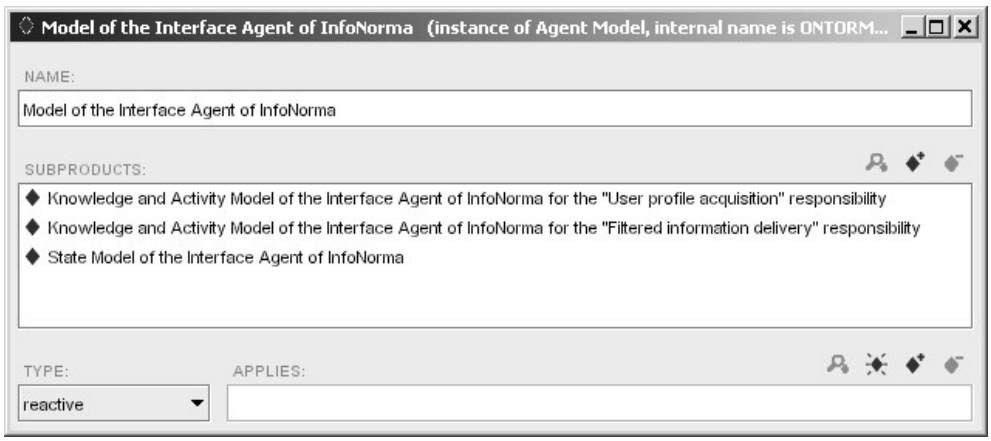


Figura 6.15: Modelo do Agente *Interfaceador* da *InfoNorma*

6.3.2.1 Modelagem do conhecimento e das atividades do agente Interfaceador

A Figura 6.16 e a Figura 6.17 mostram os dois *Modelos do Conhecimento e das Atividades do Agente* Interfaceador da InfoNorma, referentes à *aquisição de perfis de usuário* e à *entrega de informações filtradas*, respectivamente, que são as duas responsabilidades desse agente. Tais modelos representam e detalham de maneira isolada o contexto de execução dessas responsabilidades.

Assim, quanto às pré e pós-condições, respectivamente, para a *aquisição de perfis de usuário*, é suficiente que um *novo perfil de usuário tenha sido especificado*, enquanto que a execução de tal responsabilidade, mediada pelo *preenchimento de formulários em páginas da Web*, é necessária para que o *perfil do usuário esteja validado*.

Nesse processo, para cada perfil de usuário adquirido, são usadas a sua *identificação e as suas necessidades* e são produzidos *tipos e categorias de instrumentos jurídico-normativos*, sendo estes últimos conhecidos pelo agente *Interfaceador*.

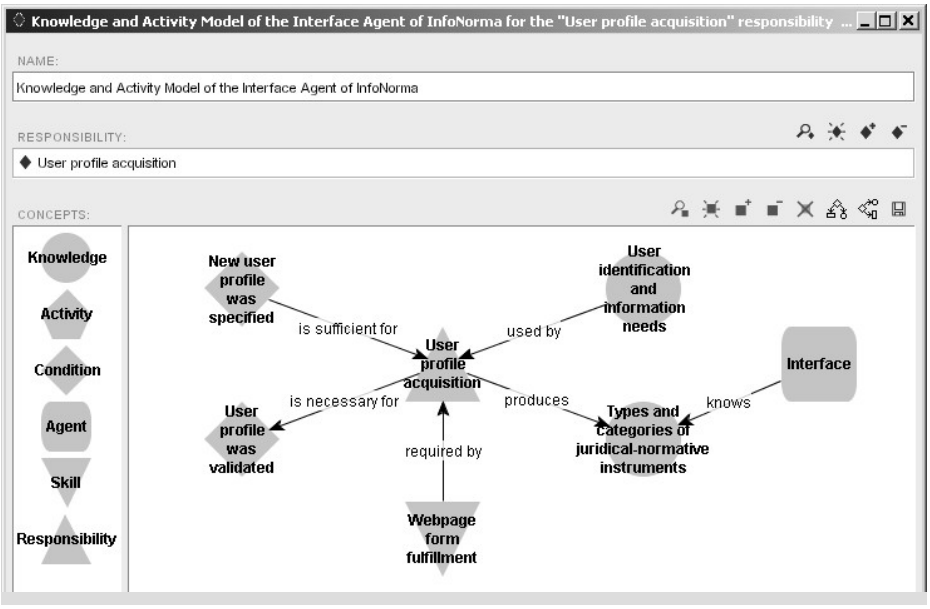


Figura 6.16: Modelo do Conhecimento e das Atividades do Agente *Interfaceador* para a responsabilidade *aquisição de perfis de usuário*

Já em relação à *entrega de informações filtradas*, é suficiente para tanto que *elementos de informação tenham sido filtrados*, sendo essa responsabilidade necessária para que *mensagens de satisfação de informação sejam enviadas*, isso feito através do *envio de mensagens de correio eletrônico*. Nesse processo, são usados *elementos de informação filtrados*, os quais são de conhecimento do agente *Filtrador*, enquanto que são produzidas *mensagens de satisfação de informação*, que fazem parte do conhecimento do agente *Interfaceador*.

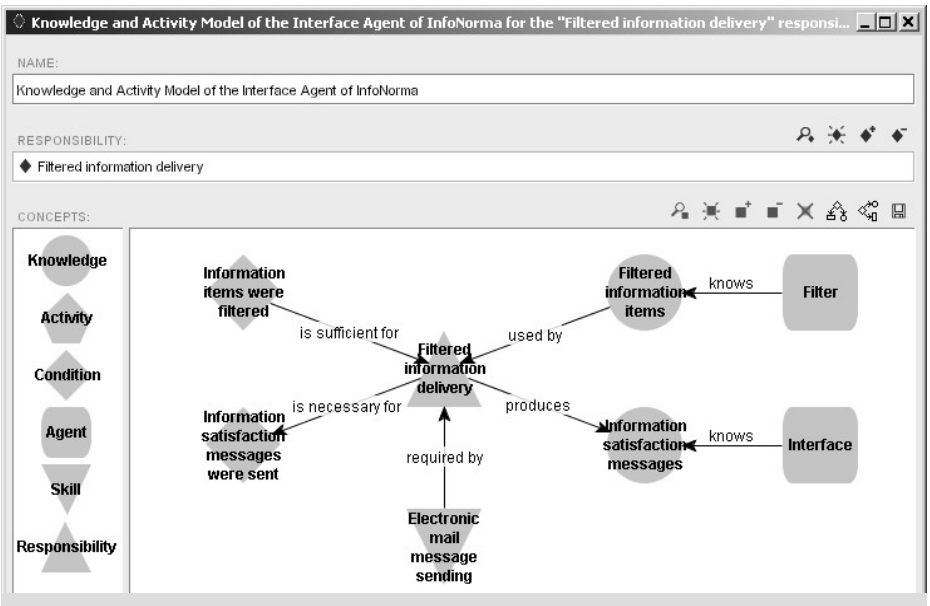


Figura 6.17: Modelo do Conhecimento e das Atividades do Agente *Interfaceador* para a responsabilidade *entrega de informações filtradas*

6.3.2.2 Modelagem dos estados do agente Interfaceador

A Figura 6.18 exibe o *Modelo dos Estados do Agente* Interfaceador da InfoNorma, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Desse modo, uma vez inicializada a aplicação, e consequentemente o agente *Interfaceador*, o seu estado passa a ser *aguardando novos perfis de usuário ou elementos de informação filtrados*, permanecendo apto a cumprir qualquer uma de suas duas responsabilidades, dependendo da transição de estado que ocorra.

Se um *novo perfil de usuário tiver sido especificado*, então o agente assume o estado *validando perfil de usuário*, no qual ele verifica a corretude dos dados fornecidos pelo usuário por meio do formulário próprio.

Senão, caso *elementos de informação tenham sido filtrados*, ele estará *enviado mensagens de correio eletrônico*, que carregam as informações a serem entregue aos usuários.

Em ambas as hipóteses, o agente retorna ao estado de espera, respectivamente, através das transições expressando que um *perfil do usuário foi validado* e que *mensagens de satisfação de informação foram enviadas*.

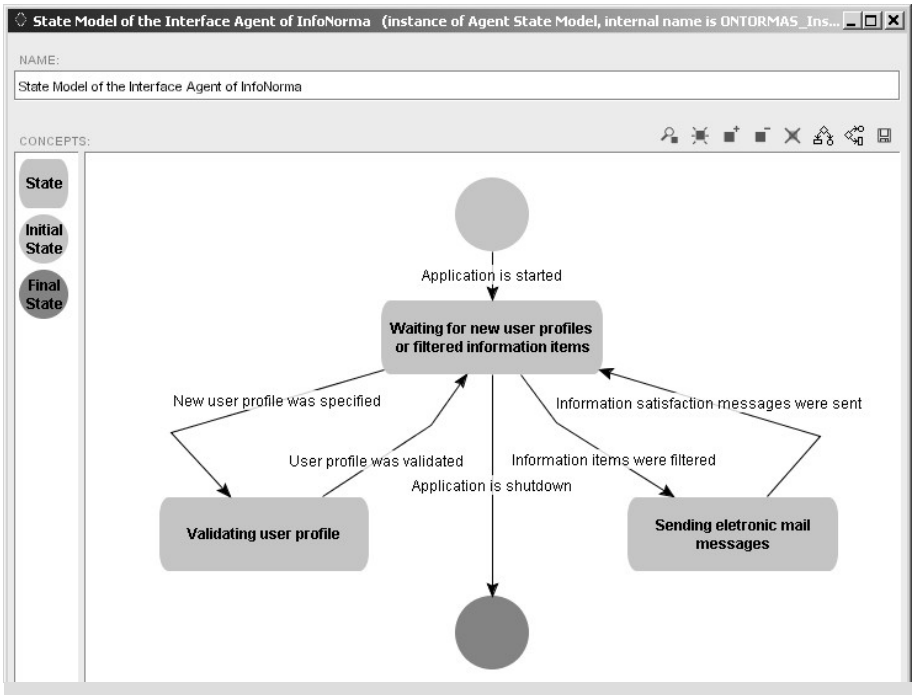


Figura 6.18: Modelo dos Estados do Agente *Interfaceador*

6.3.3 Projeto do agente *Modelador*

Nesta segunda iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Modelador da InfoNorma (Figura 6.19), que é formado pelo *Modelo do Conhecimento e das Atividades do Agente* Modelador e pelo *Modelo dos Estados do Agente* Modelador.

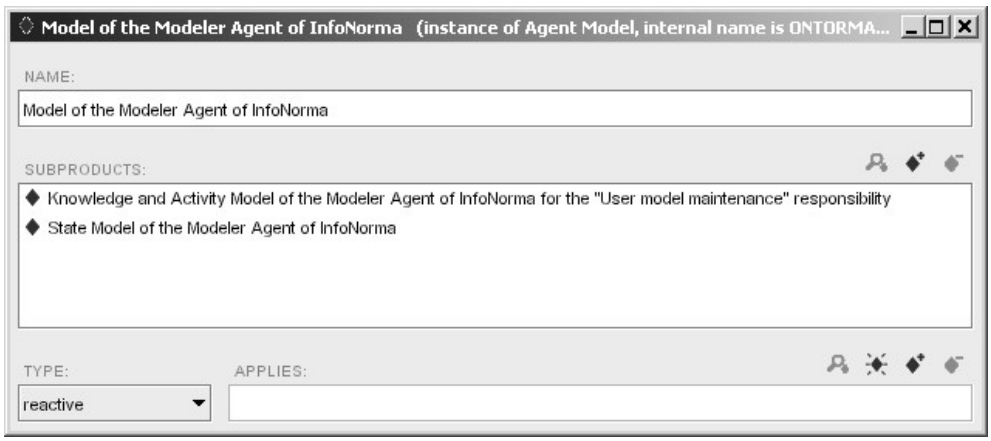


Figura 6.19: Modelo do Agente *Modelador* da *InfoNorma*

6.3.3.1 Modelagem do conhecimento e das atividades do agente Modelador

A Figura 6.20 mostra o *Modelo do Conhecimento e das Atividades do Agente* Modelador da InfoNorma, referente à *manutenção de modelos de usuários*, que é a única responsabilidade desse agente. Tal modelo representa e detalha de maneira isolada o contexto de execução das atividades de *criação de modelos de usuário* e de *extração de interesses de usuários*, que decompõem essa responsabilidade.

Assim, a *criação de modelos de usuário* tem como pré-condição que o *perfil do usuário esteja validado* e como pós-condição que um *novo modelo de usuário tenha sido criado*, sendo a destreza requerida para tanto a *formação de pares de tipo e categoria*.

Há, na consecução dessa atividade, o uso de *tipos e categorias de instrumentos jurídico-normativos*, que são do conhecimento do agente *Interfaceador*, e a produção de *modelos de usuário*, conhecidos pelo agente *Modelador*.

Já para a *extração de interesses de usuários*, é suficiente que *modelos de usuário tenham sido criados* e que *interesses de usuários tenham sido requisitadas*, sendo tal necessário para que *interesses de usuários sejam extraídos*, tudo através de uma técnica para a *formação de pares de tipo e categoria*.

Nesse processo, são usados *surrogates de elementos de informação*, de conhecimento do agente *Construtor*, e também *modelos de usuário*, sendo produzidas *surrogates de interesses de usuários*, estas, assim como aqueles, conhecidas pelo agente *Modelador*.

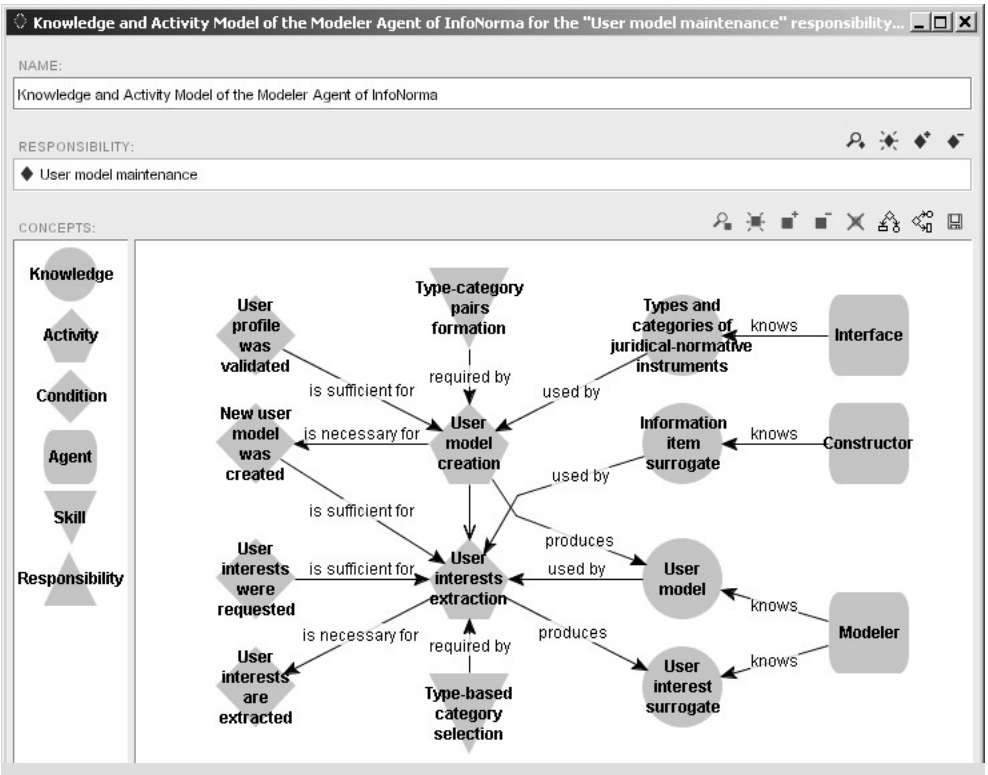


Figura 6.20: Modelo do Conhecimento e das Atividades do Agente *Modelador* para a responsabilidade *manutenção de modelos de usuários*

6.3.3.2 Modelagem dos estados do agente Modelador

A Figura 6.21 exibe o *Modelo dos Estados do Agente* Modelador da InfoNorma, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Dessa forma, enquanto não finalizada a aplicação, e na falta de outro estado, o agente *Modelador* permanece *aguardando dados de perfis de usuário ou elementos de informação filtrados*, sendo possível ele seguir deste, conforme a

transição que aconteça, para um ou outro estado correspondente a suas duas atividades.

Se um *perfil do usuário tiver sido validado*, então o agente passa ao estado *criando modelo de usuário*, em que ele procede à formação dos pares de tipos e categorias que constituem tais modelos.

Do contrário, se *interesses de usuários tiverem sido requisitadas*, ele estará *extraindo interesses de usuários*, que são utilizadas no processo de filtragem de informação.

Nas duas hipóteses, o agente retorna ao estado de aguardo através das transições informando que um *novo modelo de usuário foi criado* e que *interesses de usuários foram extraídos*, respectivamente.

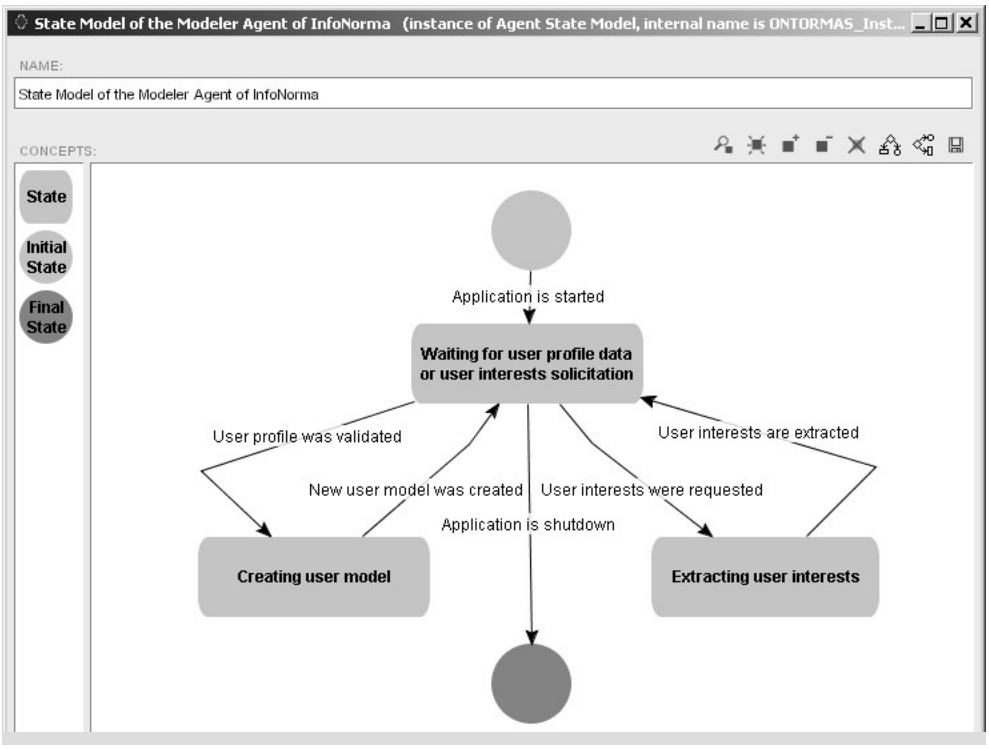


Figura 6.21: Modelo dos Estados do Agente *Modelador*

6.3.4 Projeto do agente *Monitor*

Nesta terceira iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Monitor da InfoNorma (Figura 6.22), que é formado pelo *Modelo do Conhecimento e das Atividades do Agente* Monitor e pelo *Modelo dos Estados do Agente* Monitor.

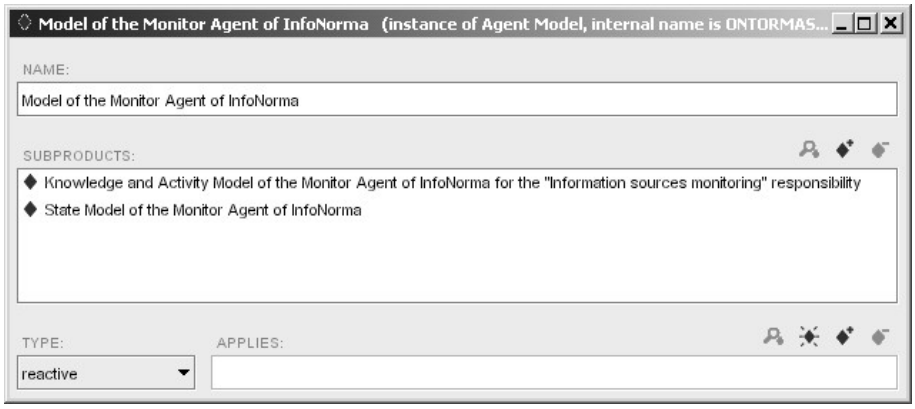


Figura 6.22: Modelo do Agente *Monitor* da *InfoNorma*

6.3.4.1 Modelagem do conhecimento e das atividades do agente Monitor

A Figura 6.23 mostra o *Modelo do Conhecimento e das Atividades do Agente* Monitor da InfoNorma, referente ao *monitoramento da fonte de informação*, que é a única responsabilidade desse agente. Tal modelo representa e detalha de maneira isolada o contexto de execução dessa responsabilidade.

O *monitoramento da fonte de informação* tem como pré-condição que pelo menos um *índice de informação tenha sido atualizado* e como pós-condição que alguns *novos elementos de informação tenham sido detectados*, sendo a destreza requerida para tanto a *detecção de atualizações em índices de informação*. E, no seu desempenho, ocorre o uso de *mudanças na fonte de informação* e a produção de *novos elementos de informação*, que são do conhecimento do agente *Monitor*.

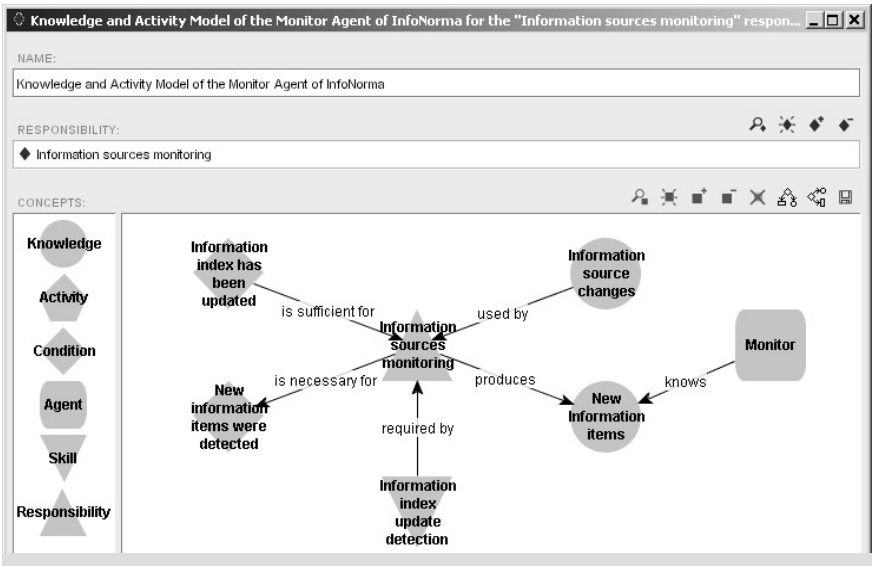


Figura 6.23: Modelo do Conhecimento e das Atividades do Agente *Monitor* para a responsabilidade *monitoramento da fonte de informação*

6.3.4.2 Modelagem dos estados do agente Monitor

A Figura 6.24 exibe o *Modelo dos Estados do Agente* Monitor da InfoNorma, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Dessa maneira, o agente *Monitor* começa seu ciclo de vida *aguardando mudanças na fonte de informação*, estado no qual ele fica até que uma transição tenha vez, anunciando que um processamento é demandado.

Se algum *índice de informação tiver sido atualizado*, então o agente entra no estado *detectando novos elementos de informação*, onde verifica os índices da fonte em busca de novas informações.

A saída desse estado coincide com uma transição indicando que *novos elementos de informação foram detectados*, quando se iniciam os passos prévios ao processo de filtragem de informação.

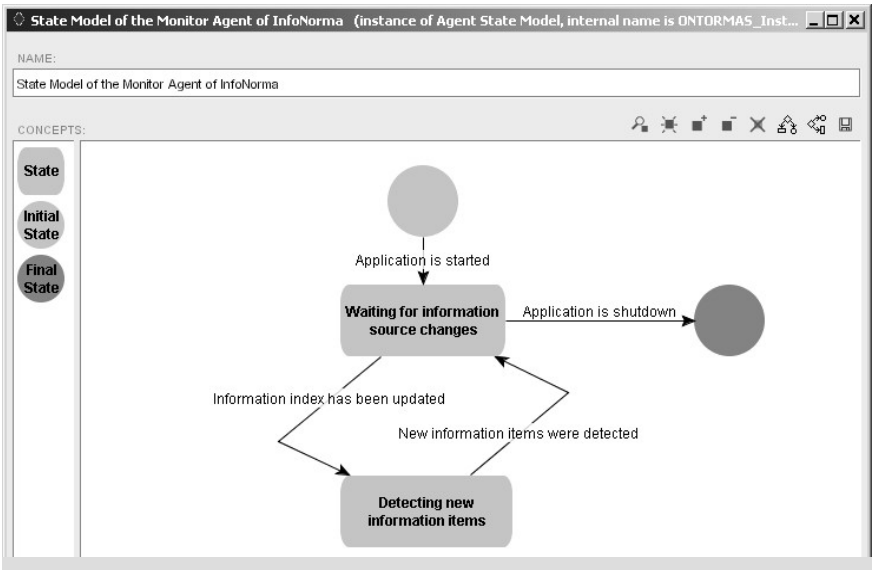


Figura 6.24: Modelo dos Estados do Agente *Monitor*

6.3.5 Projeto do agente *Construtor*

Nesta quarta iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Construtor da InfoNorma (Figura 6.25), que é formado pelo *Modelo do Conhecimento e das Atividades do Agente* Construtor e pelo *Modelo dos Estados do Agente* Construtor.

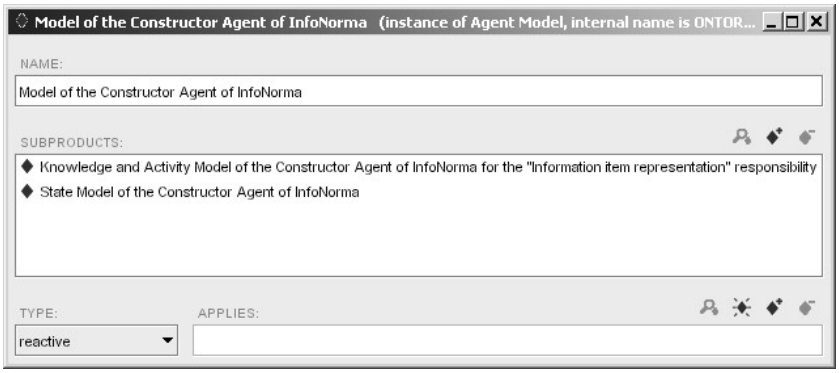


Figura 6.25: Modelo do Agente *Construtor* da *InfoNorma*

6.3.5.1 Modelagem do conhecimento e das atividades do agente Construtor

A Figura 6.26 mostra o *Modelo do Conhecimento e das Atividades do Agente* Construtor da InfoNorma, referente à *representação dos elementos de informação*, que é a única responsabilidade desse agente. Tal modelo representa e detalha de maneira isolada o contexto de execução dessa responsabilidade.

Para a *extração de interesses de usuários*, é suficiente que *novos elementos de informação tenham sido detectados*, sendo tal necessário para que *surrogates de elementos de informação estejam disponíveis*, isso através de uma técnica para a *identificação do tipo do elemento e contagem da freqüência das palavras-chave*. São usados *novos elementos de informação*, de conhecimento do *Monitor*, e produzidos *surrogates de elementos de informação*, conhecidos pelo.

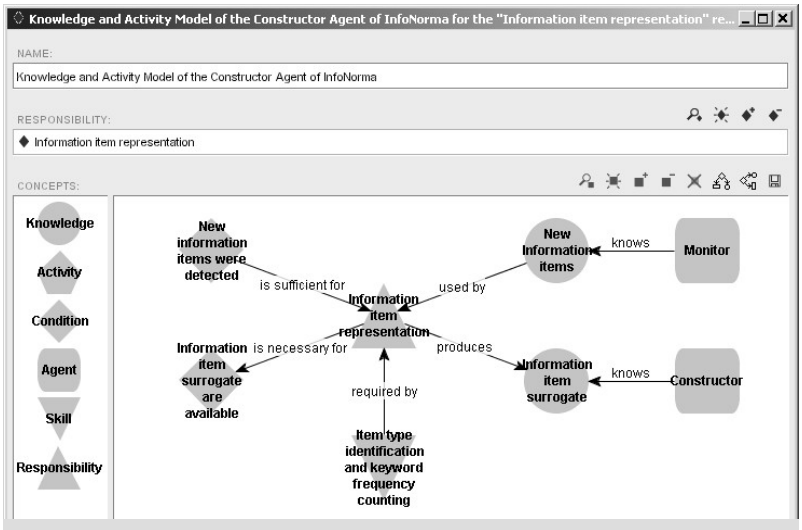


Figura 6.26: Modelo do Conhecimento e das Atividades do Agente *Construtor* para a responsabilidade *representação dos elementos de informação*

6.3.5.2 Modelagem dos estados do agente Construtor

A Figura 6.27 exibe o *Modelo dos Estados do Agente* Construtor da InfoNorma, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Desse jeito, o agente *Construtor* principia sua execução no estado *aguardando novos elementos de informação*, em que ele permanece enquanto nenhuma transição ocorre.

Se *novos elementos de informação tiverem sido detectados*, então o agente passa para o estado *representando elementos de informação*, no qual constrói as respectivas representações internas.

A volta para o estado de espera é determinada pelo fato de que *surrogates de elementos de informação estejam disponíveis*, quando prosseguem os passos prévios ao processo de filtragem de informação.

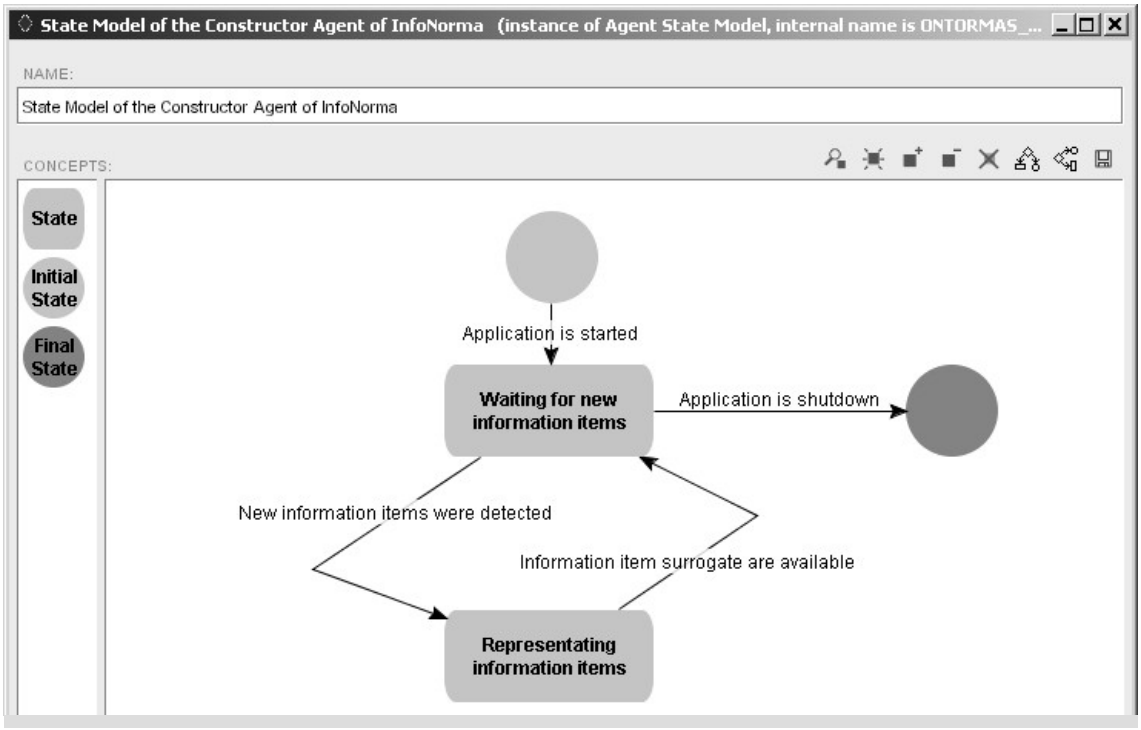


Figura 6.27: Modelo dos Estados do Agente *Construtor*

6.3.6 Projeto do agente *Filtrador*

Nesta quinta iteração da segunda subfase, tem-se como produto o *Modelo do Agente* Filtrador da InfoNorma (Figura 6.28), que é formado pelo *Modelo do Conhecimento e das Atividades do Agente* Filtrador e pelo *Modelo dos Estados do Agente* Filtrador.

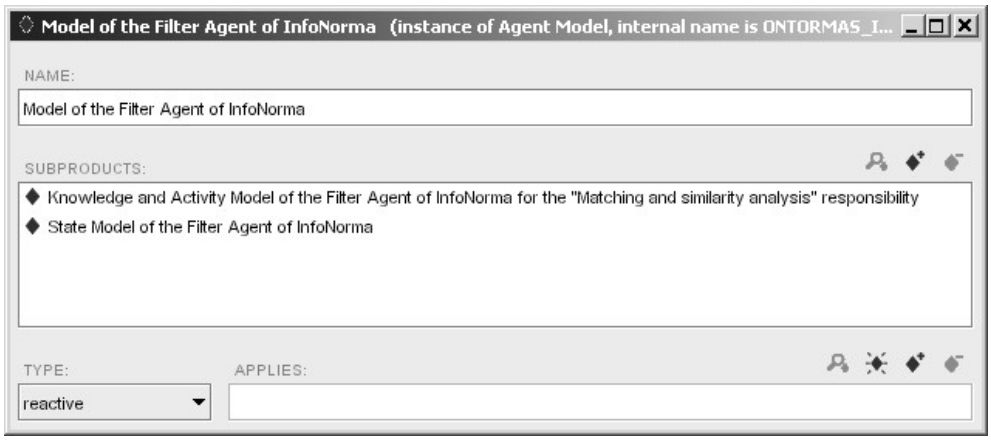


Figura 6.28: Modelo do Agente *Filtrador* da *InfoNorma*

6.3.6.1 Modelagem do conhecimento e das atividades do agente Filtrador

A Figura 6.29 mostra o *Modelo do Conhecimento e das Atividades do Agente* Filtrador da InfoNorma, referente à *comparação e análise de similaridade*, que é a única responsabilidade desse agente. Tal modelo representa e detalha de maneira isolada o contexto de execução dessa responsabilidade.

Assim, a *comparação e análise de similaridade* têm como pré-condições que *interesses de usuários tenham sido extraídos* e que *surrogates de elementos de informação estejam disponíveis*, e como pós-condição que *elementos de informação sejam filtrados*, sendo a destreza requerida para tanto a *comparação de palavras-chave e cálculo de percentual e peso*.

Na execução dessa responsabilidade, ocorre o uso de *surrogates de interesses de usuários* e de *surrogates de elementos de informação*, conhecidas, respectivamente, pelos agentes *Modelador* e *Construtor*, e a produção de *elementos de informação filtrados*, de conhecimento do agente *Filtrador*.

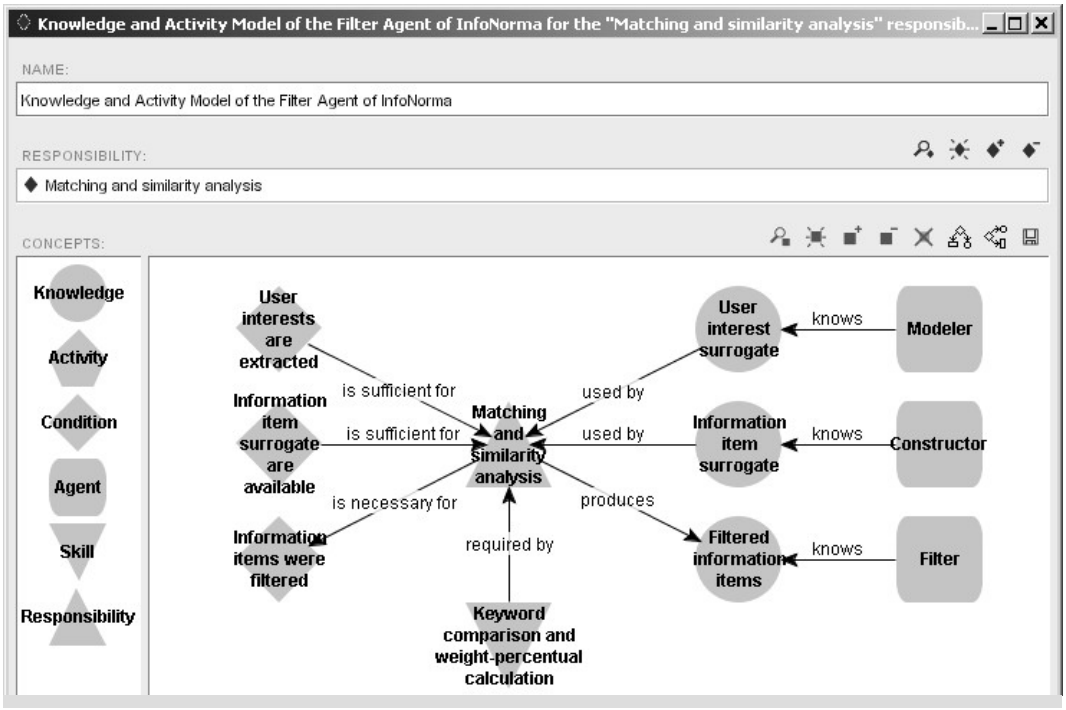


Figura 6.29: Modelo do Conhecimento e das Atividades do Agente *Filtrador* para a responsabilidade *comparação e análise de similaridade*

6.3.6.2 Modelagem dos estados do agente Filtrador

A Figura 6.30 exibe o *Modelo dos Estados do Agente* Filtrador da InfoNorma, que representa os estados pelos quais o agente passa durante seu funcionamento, bem como as transições que leva de um para outro.

Dessa forma, o agente *Filtrador* inicia seu ciclo de vida *aguardando surrogates de elementos de informação*, estado no qual ele fica até que uma transição aconteça.

Se *surrogates de elementos de informação estiverem disponíveis*, então o agente segue para o estado *esperando interesses de usuários*, em que aguarda que tais interesses sejam fornecidos pelo agente *Modelador*.

E, uma vez que esses *interesses de usuários sejam extraídos*, tem-se a transição para o estado *filtrando elementos de informação*, onde se sucede o processo de filtragem de informação propriamente.

Ao fim, com *elementos de informação tendo sido filtrados*, termina o processamento, transitando-se para o estado de aguardo.

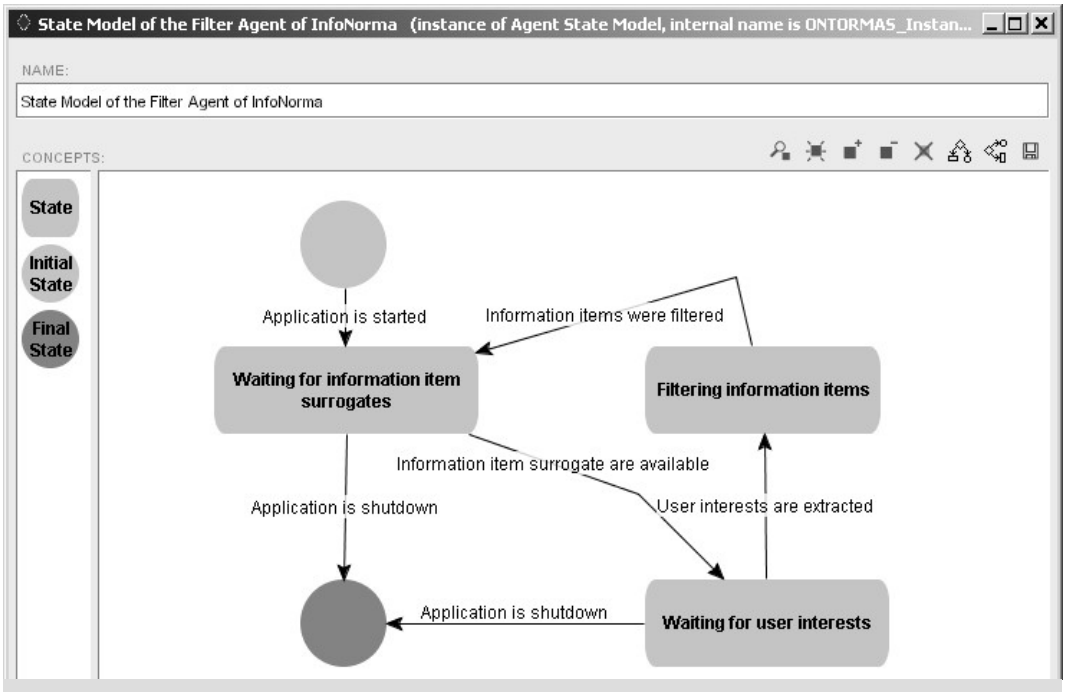


Figura 6.30: Modelo dos Estados do Agente *Filtrador*

6.3.7 Modelagem do conhecimento da sociedade multiagente

Nesta terceira subfase, é produzido o *Modelo do Conhecimento da Sociedade Multiagente* da InfoNorma (Figura 6.31), sendo nada mais que a base para a construção da ontologia de comunicação dos agentes, reunindo os conceitos carregados nas mensagens trocadas entre eles.

Então, no caso da filtragem de instrumentos jurídico-normativos, os cinco agentes irão se comunicar mediante a troca de quatro conhecimentos: *perfil*, *surrogate*, *interesse* e *satisfação*.

Um *perfil* é apresentado por um *usuário*, sendo composto pela sua *identificação*, que tem o *nome de usuário*, a *senha* e o *e-mail*, e pelos *interesses*, que têm um *tipo* e uma *categoria*.

Um *surrogate* representa um *item de informação*, tendo um *tipo*, um *número*, uma *data*, um *endereço* e *palavras-chave*, que se compõem de uma *palavra* e uma *freqüência*.

Uma *interesse* é extraído de um *modelo de usuário*, possuindo um *usuário*, o *identificador* do item de informação empregado na extração e *categorias* vinculadas ao tipo de tal item.

Uma *satisfação*, é baseada em *informações filtradas*, compondo-se do *usuário* a que se destina; do *tipo*, do *número*, da *data* e do *endereço* do item filtrado que lhe originou; da *categoria* em que houve o enquadramento; e do grau de *similaridade* calculado.

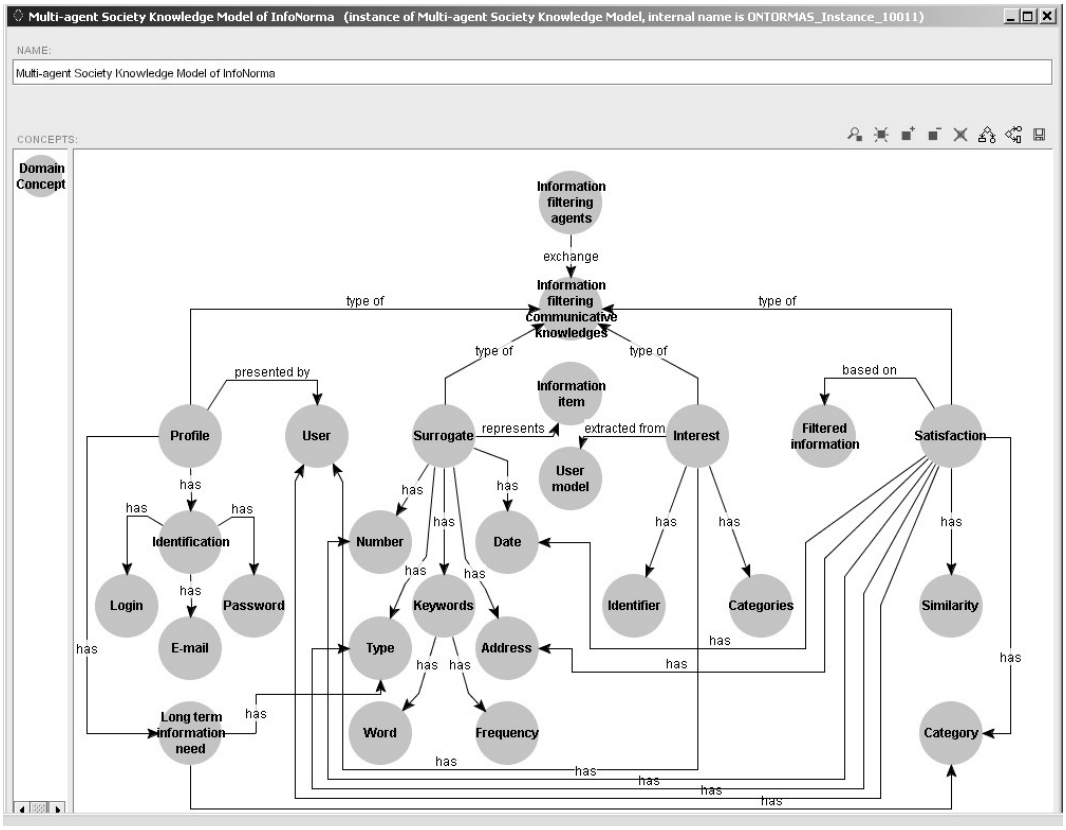


Figura 6.31: Modelo do Conhecimento da Sociedade Multiagente da *InfoNorma*

Uma vez concluídas todas as subfases e todos os passos da fase de projeto da aplicação, chega-se à implementação, em que haverá o mapeamento dos elementos aqui modelados para a elaboração de modelos relativos a uma tecnologia particular.

6.4 Implementação da aplicação *InfoNorma*

Na fase de Implementação da Aplicação, de acordo com a metodologia MAAEM, resta ao desenvolver elaborar o *Modelo de Implementação da Aplicação* (Figura 6.32), o qual é composto por dois subprodutos: o *Modelo de Agentes e Comportamentos* e o *Modelo de Atos de Comunicação entre Agentes*. Nas próximas

subseções, é pormenorizadamente demonstrada a confecção desses modelos na implementação da InfoNorma.

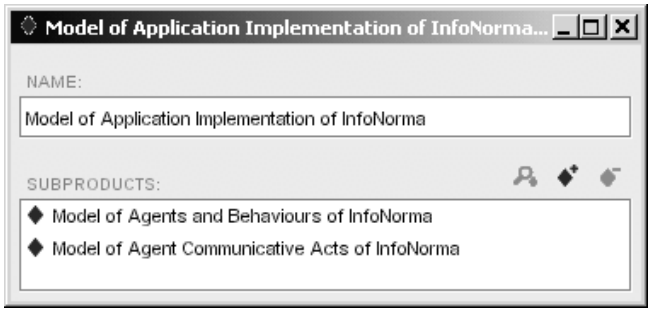


Figura 6.32: Modelo de Implementação da Aplicação *InfoNorma*

6.4.1 Mapeamento de agentes do projeto à implementação e de responsabilidades em comportamentos

A Figura 6.33 mostra o *Modelo de Agentes e Comportamentos* da InfoNorma, o qual relaciona instâncias das classes *agente* e *comportamento* da plataforma JADE, a tecnologia de implementação adotada, identificadas a partir dos *agentes* e *responsabilidades* projeto, respectivamente.

Assim, foram instanciados os agentes *Interfaceador*, *Modelador*, *Monitor*, *Construtor* e *Filtrador*, que compõem a arquitetura da aplicação. E, de acordo com as responsabilidades por eles assumidas, procedeu-se à instanciação dos comportamentos de cada um, os quais podem apresentar variados tipos, conforme exemplificado a seguir.

Inicialmente, o *Interfaceador* fica encarregado de dois comportamentos: *adquirir perfis de usuário*, que é *cíclico*, pois permanece em constante estado de espera por novos perfis especificados; e *entregar informações filtradas*, o qual é *simples*, já que é disparado cada vez que a filtragem produz resultados.

Em seguida, ao *Modelador* é atribuído o comportamento *manter modelos de usuário*, que é *composto*, tendo como subcomportamentos *criar modelos de usuário* e *extrair interesses de usuários*, ambos *atômicos*, posto que são executados de uma vez só, respectivamente, a cada perfil de usuário adquirido e a cada item de informação representado.

Logo após, o *Monitor* executa o comportamento *monitorar fonte de informação*, o qual também é *cíclico*, visto que sempre ocorrem mudanças na fonte a

serem detectadas e o *Construtor* cuida do comportamento *representar items de informação*, igualmente *atômico*, originado a cada novo item de informação identificado.

Finalmente, tem-se o *Filtrador* incumbido do comportamento *filtrar items de informação*, que é *cíclico*, sendo continuamente executado na espera de itens de informações a serem submetidos à filtragem.

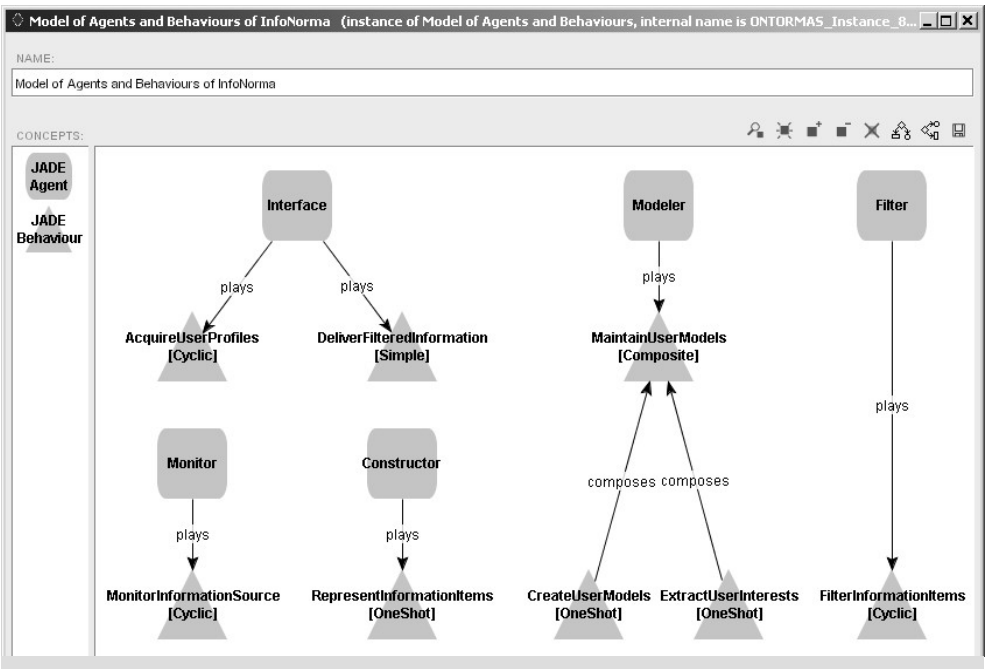


Figura 6.33: Modelo de Agentes e Comportamentos da *InfoNorma*

Realizada essa parte da implementação, tendo sido modelados todos os agentes e comportamento nos termos do JADE, resta apenas modelar os atos de comunicação entre os agentes.

6.4.2 Mapeamento de interações entre agentes em atos de comunicação

A Figura 6.34 traz o *Modelo de Atos de Comunicação entre Agentes* da InfoNorma, que exibe a forma como o conhecimento da sociedade multiagente é trocado entre os agentes por meio das mensagens que uns enviam para outros.

Em primeiro lugar, uma mensagem "*1-INFORM_REF (perfil)*" é originada pelo agente *Interfaceador* com destino ao *Modelador*, a qual carrega um objeto perfil de usuário, assim que é adquirido e validado por aquele, para que este crie e mantenha o respectivo modelo, a ser usado futuramente no processo de filtragem.

Já quando acontecem mudanças na fonte de informação, uma mensagem "*2-INFORM (item de informação)*" é remetida do agente *Monitor* ao *Construtor*, para cada novo item de informação, contendo seu endereço, para que seja construída a correspondente representação.

Em seguida, tendo sido executada a tarefa do agente *Construtor*, uma mensagem "*3-INFORM_REF (surrogate)*" é enviada por ela ao *Filtrador*, portando um objeto que representa um item de informação.

Feito isso, uma mensagem "*4-REQUEST (tipo do instrumento normativo, número do instrumento normativo)*" é encaminha pelo agente *Filtrador* ao *Modelador*, através da qual aquele especifica uma requisição para que este extraia e devolva interesses de informação de acordo com os parâmetros informados.

Em resposta, para cada usuário que possua interesses em conformidade com os mencionados parâmetros, uma mensagem "*5-INFORM_REF (interesse)*" é gerada pelo agente *Modelador* e passada para o *Filtrador*, permitindo o prosseguimento da filtragem de informação, suspensa à espera disso.

Por último, realizada a filtragem, uma mensagem "*6-INFORM_REF (satisfação)*" é produzida pelo agente *Filtrador* a cada item de informação filtrado segundo os interesses extraídos dos referidos modelos, sendo mandada ao *Interfaceador*, para então ser entregue ao destinatário final, que é o usuário indicado no próprio objeto portando a satisfação de informação.

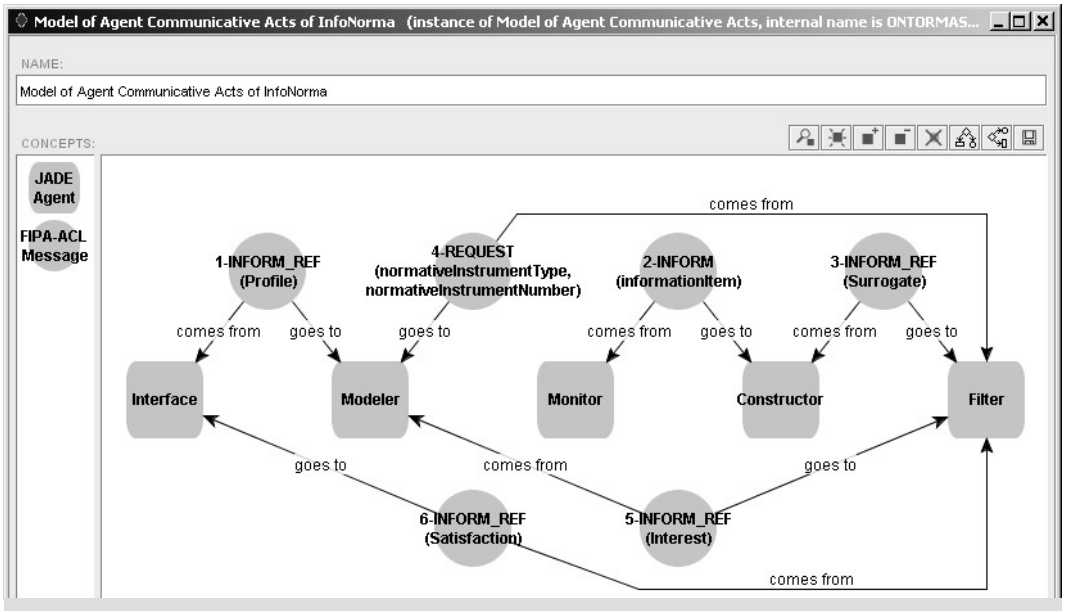


Figura 6.34: Modelo de Atos de Comunicação entre Agentes da *InfoNorma*

Nesse ponto, findam-se as três fases do desenvolvimento da InfoNorma, estando elaborados todos os modelos necessários para o término da construção da aplicação. Então, a partir daqui, resta somente empregar tais modelos no desenho de um protótipo e na escrita do código, como se acompanhará a seguir.

6.5 Prototipação e codificação da aplicação *InfoNorma*

Uma vez realizada toda a modelagem da aplicação *InfoNorma*, tendo-se concluído com êxito todos os passos das fases de análise, projeto e implementação previstos pela metodologia MAAEM, a presente seção cuidará de demonstrar a forma como a aplicação se apresentará quando pronta, sendo isso descrito a seguir e ilustrado com telas do seu protótipo e trechos do seu código.

Preliminarmente, deve-se considerar que a *InfoNorma* é uma aplicação cujo funcionamento acontece de maneira pretensamente ininterrupta, sendo executada em um servidor e não possuindo uma interface direta com usuários, senão a provida pela própria plataforma JADE para sua manutenção, expressando-se para eles, por conta disso, através de aplicações auxiliares com as quais troca informações.

Assim, tem-se em primeiro lugar uma aplicação suportada pela Web para aquisição de perfis de usuários (Figura 6.35), desenvolvida através das linguagens PHP (2006), que roda do lado do servidor, e JavaScript (2006), que roda do lado do cliente. Por meio do preenchimento dos formulários que compõem sua interface, os usuários tanto inserem quanto atualizam, removem e exibem os dados que caracterizam seus respectivos perfis de usuário, ficando estes armazenados em um banco de dados e acessíveis à *InfoNorma*, para que ela possa acessá-los e deles inferir interesses a serem confrontados com informações sobre instrumentos jurídico-normativos no processo de filtragem.

De fato, diretamente, tais registros de dados – bem como, indiretamente, tais páginas na Internet – representam uma face dos usuários, assim considerados como entidades que interagem com a *InfoNorma* para lhe fornecer dados de entrada ao seu processamento, posto que é por intermédio daqueles que esta pode receber a identificação e as necessidades de cada usuário, através de consultas periódicas à base de dados, cujo interfaceamento ocorre com ambos, aplicação e usuário, uma independentemente do outro.

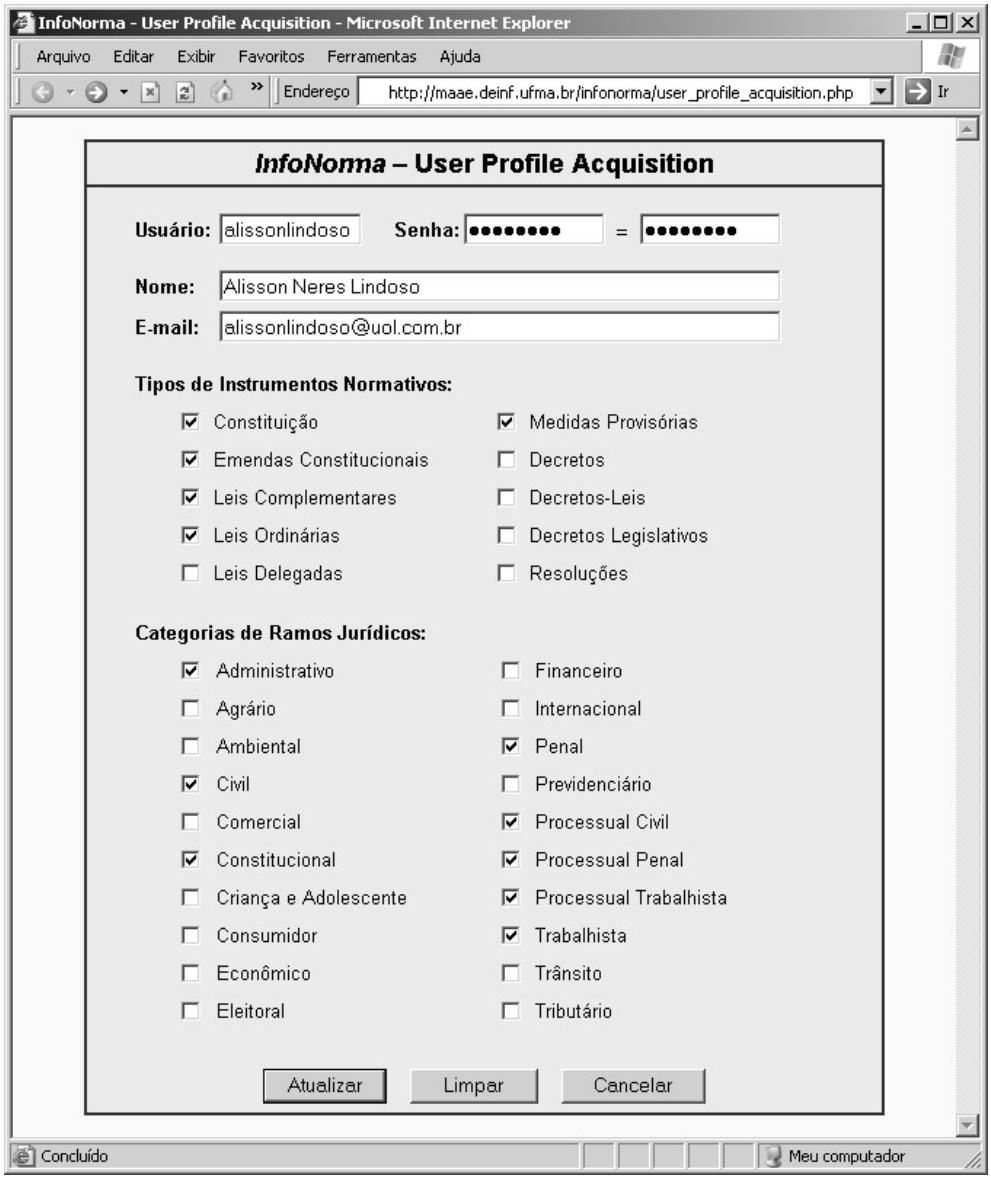


Figura 6.35: Protótipo da aplicação para aquisição de perfis de usuários da *InfoNorma*

Por outro lado, para entrega dos resultados do processo de filtragem aos usuários, será usado o correio eletrônico (Figura 6.36), o que retira do conjunto de preocupações concernentes à aplicação questões tais como segurança, autenticação etc, já contempladas pelo próprio serviço de e-mail.

Na verdade, as caixas de mensagens representam outra face dos usuários, assim encarados como sujeitos com que a aplicação interage para lhes oferecer dados de saída do seu processamento, visto que é através daquelas que esta é capaz de entregar as satisfações de informação a cada usuário, a qualquer instante, mesmo que ele não esteja acessando suas mensagens eletrônicas no momento.

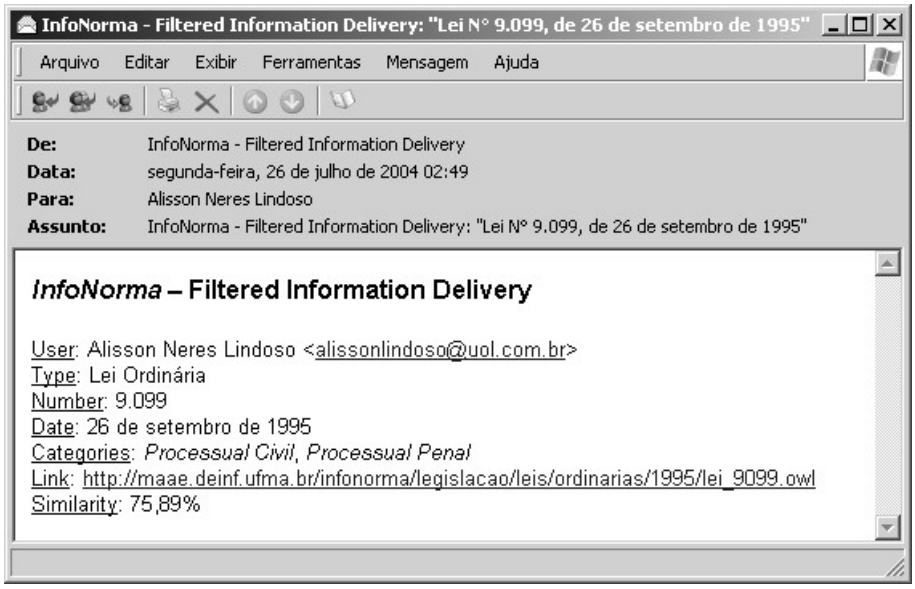


Figura 6.36: Exemplo de mensagem de entrega de informação filtrada pela *InfoNorma*

No tocante à fonte de informação, que consiste na seção de Legislação (Figura 6.37) do *website* da Presidência da República Federativa do Brasil (2005), têm-se problemas porque nela tanto os instrumentos jurídico-normativos quanto os índices que os listam são documentos da Web tradicional, escritos na linguagem HTML – *HyperText Markup Language* (2005), sendo exibidos de modo correto, mas completamente desestruturados, dificultando o seu processamento pela *InfoNorma*.

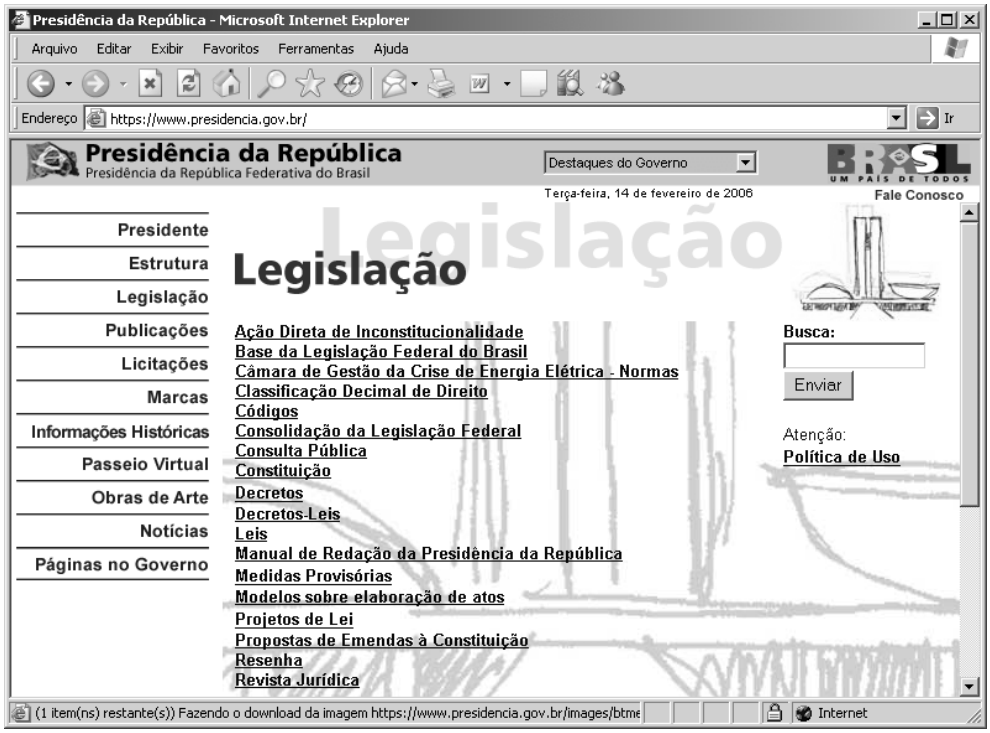


Figura 6.37: Seção de *Legislação* da página da *Presidência da República* na Internet

Então, para contornar essa dificuldade, tais elementos e índices de informação são convertidos para a Web semântica, através da construção de uma “fonte-espelho”, ficando os diversos tipos de documentos normativos representados na *OntoLegis* – Uma Ontologia de Normas Legais, que foi escrita na linguagem OWL - *Ontology Web Language* (2005), por sua vez suportada pelo modelo RDF - *Resource Description Framework* (2005) e pela linguagem XML - *Extensible Markup Language* (2005), resultando bem descritos e adequadamente estruturados tais documentos, o que facilita o seu tratamento pela aplicação.

A *OntoLegis* representa a estrutura genérica de um instrumento normativo, seja qual for seu tipo. Desse modo, a partir do conhecimento relativo às leis e aos demais tipos de normas jurídicas, construiu-se uma ontologia cujas instâncias são leis, decretos, resoluções medidas provisórias etc.

Conforme essa ontologia, um instrumento normativo é composto por uma identificação, uma ementa, um ou mais dispositivos e dados sobre sua publicação. A identificação, por sua vez, é composta por um tipo, um número e uma data, cujas partes são o dia, o mês e o ano. Os dispositivos podem ser grupos ou elementos. Títulos, capítulos e seções são tipos de grupos; artigos, parágrafos, incisos e alíneas são tipos de elementos. Seções fazem parte de capítulos, e estes, de títulos. Alíneas fazem parte de incisos, e estes, de parágrafos ou de artigos. A rede semântica (Figura 6.38) abaixo mostra os conceitos e os relacionamentos acima descritos.

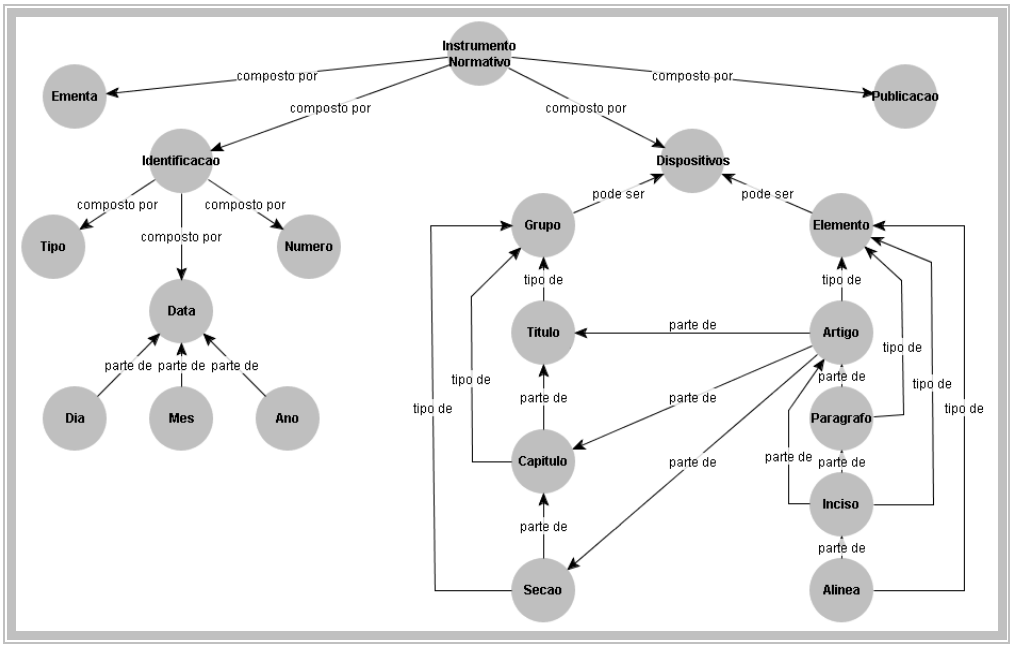


Figura 6.38: Rede semântica representando um instrumento normativo

Uma vez obtida a estrutura anterior, fez-se a sua transposição para uma hierarquia de *frames* e respectivos *slots* (Figura 6.39), através do editor Protégé.

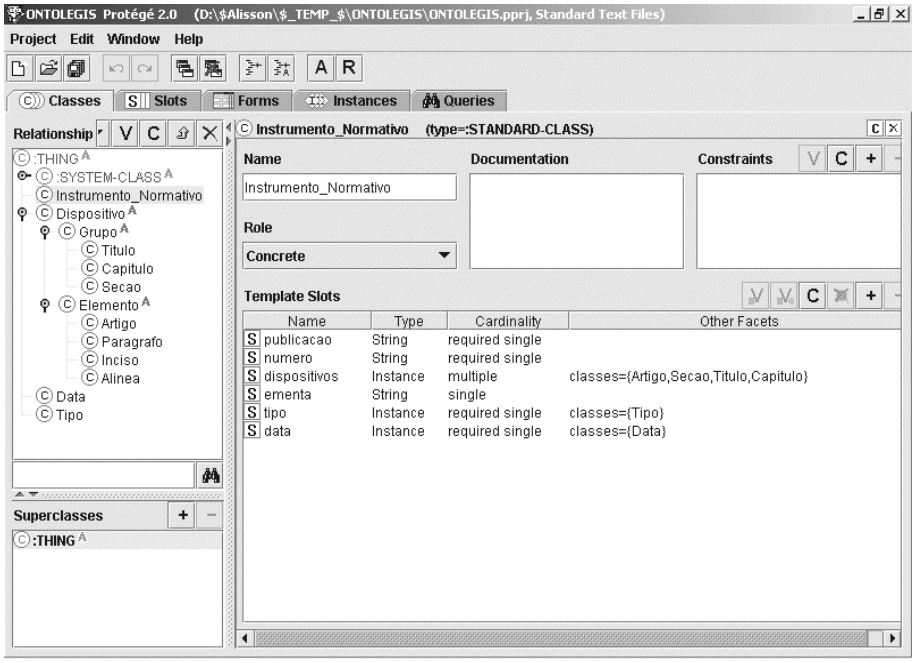


Figura 6.39: Hierarquia de *frames* e respectivos *slots* da *OntoLegis*

O primeiro passo para a representação de um instrumento normativo, através de uma instância da *OntoLegis*, é a escolha de seu tipo (Figura 6.40), dentre aqueles listados pelo especialista no domínio jurídico.

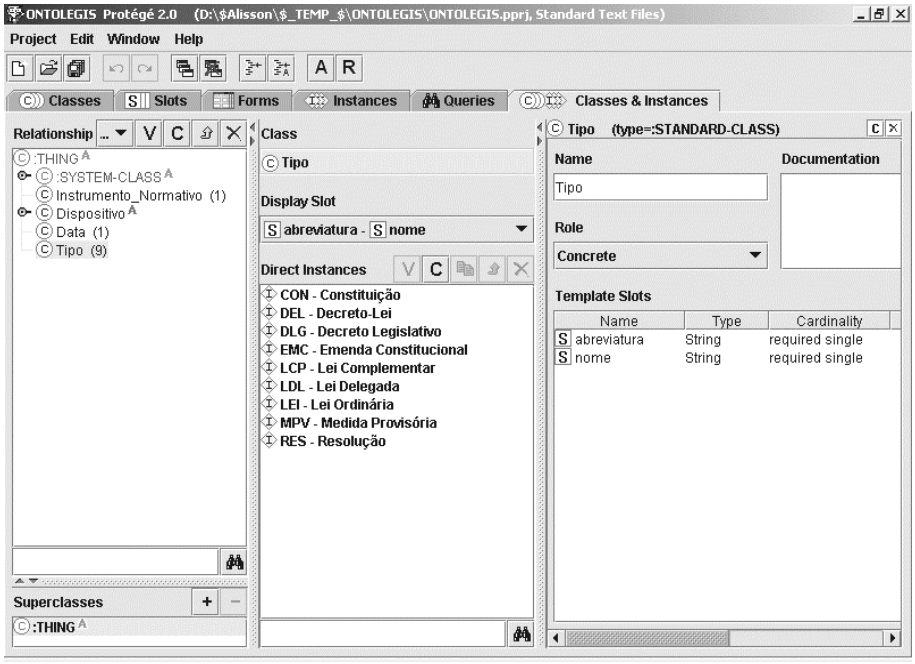


Figura 6.40: Tipos de instrumentos normativos listados na *OntoLegis*

Um exemplo de instanciação da *OntoLegis* é a representação da Lei Ordinária Nº 9.099, de 26 de setembro de 1999 (Figura 6.41), que dispõe sobre os Juizados Especiais Cíveis e Criminais. Alguns dos seus dispositivos são destacados, como o Capítulo II – Dos Juizados Especiais Cíveis e sua Seção I – Da Competência, e o Artigo 3º, juntamente com seu Parágrafo 1º e o Inciso I do último.

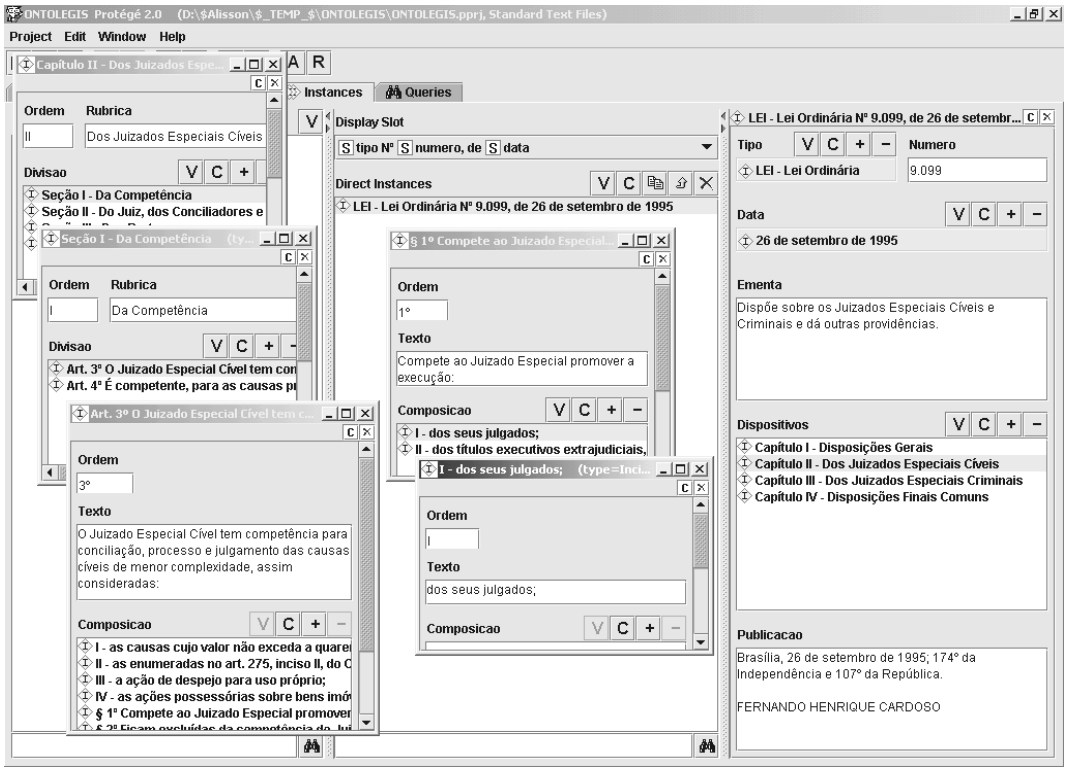


Figura 6.41: Exemplo de instanciação de um instrumento normativo na *OntoLegis*

Então, também de forma auxiliar, outra aplicação apóia o funcionamento da *InfoNorma*, com relação à manutenção da fonte de informação, a qual foi desenvolvida em Java (2005), tendo vinculação direta com a ontologia *OntoLegis* e possuindo três diferentes aspectos distintos: *Edição*, *Código fonte* e *Visualização*.

O cadastro de novos instrumentos jurídico-normativos na fonte de informação poderia muito bem ser realizado através da própria interface do Protégé, conforme ilustrado na figura anterior, mas com relativa dificuldade para usuários leigos, como os pretensos mantenedores da fonte de informação em foco.

Assim, dispõe-se de um formulário (Figura 6.42) que reflete as classes e os atributos da ontologia, o qual pode ser facilmente preenchido com os dados de um instrumento jurídico-normativo, que são, em seguida, automaticamente convertidos para o formato OWL (Figura 6.43) e armazenados como uma instância

da *OntoLegis*, sendo isso feito através do JENA – *Semantic Web Framework for Java* (2005). Tal instância pode ainda ser visualizada da forma como o usuário faria em seu navegador de Internet (Figura 6.44).

Figura 6.42: Parte de *Edição* da aplicação para manutenção da fonte de informação da *InfoNorma*

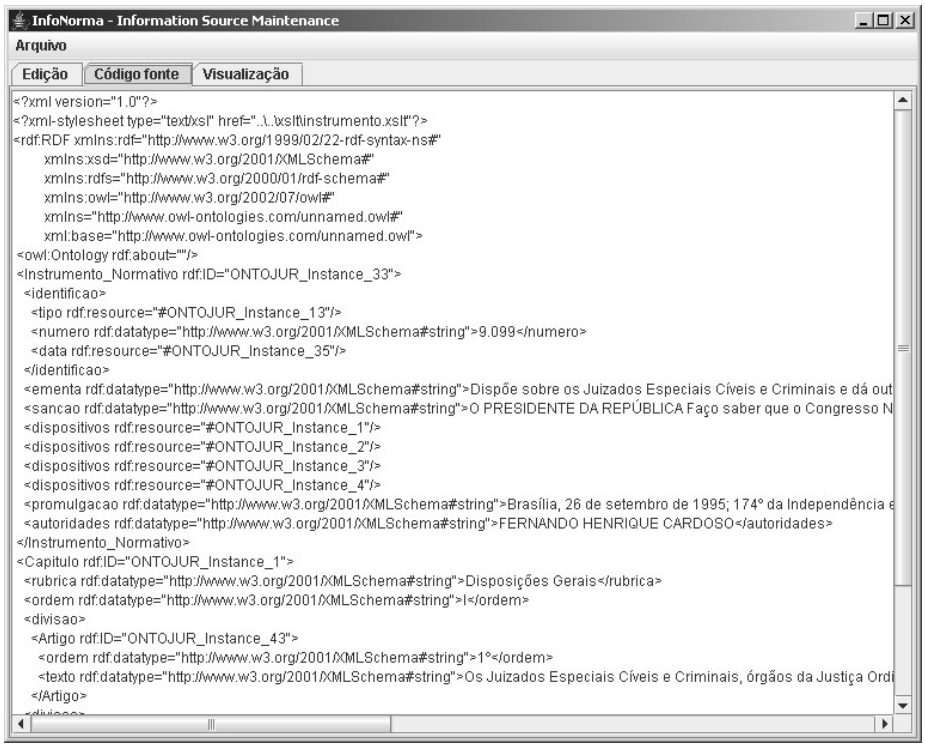


Figura 6.43: Parte de *Código fonte* da aplicação para manutenção da fonte de informação da *InfoNorma*

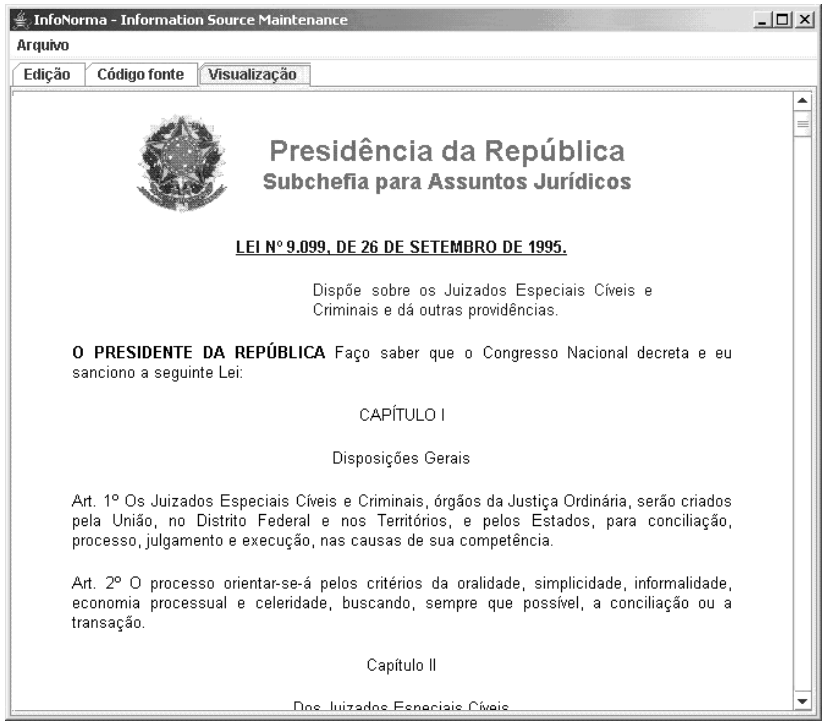


Figura 6.44: Parte de *Visualização* da aplicação para manutenção da fonte de informação da *InfoNorma*

Quanto à armazenagem dos arquivos OWL contendo os instrumentos jurídico-normativos e seus respectivos índices, foi preservada, na migração para a abordagem semântica, a organização tradicional dos diretórios. Assim, a decorrência dessa passagem é apenas o ganho quanto à manipulação automatizada de tais arquivos, com extensão ".owl", e diretórios, entre colchetes "[ ]", que são organizados de maneira hierárquica (Figura 6.45).

Portanto, de acordo com o andamento do processo legislativo, são variáveis somente os diretórios anuais e arquivos indexatórios e instrumentais, restando fixos todos os demais. E tal aplicação auxiliar cuida para que os correspondentes índices sejam automaticamente criados se não existirem ainda ou apenas atualizados, caso contrário. O mesmo ocorre em relação aos diretórios, sendo criados quando o documento for o primeiro do ano, ou somente utilizados quando já houver outros.

Além disso, para tornar os índices de legislação disponíveis para o monitoramento pela aplicação, há que se representar a fonte de informação como uma entidade – precisamente um objeto, cujo único atributo é um conjunto de índices, cada um composto por um endereço, uma data e um ponteiro – com a qual a *InfoNorma* possa interagir em busca de mudanças que disparem o processo de filtragem de informação.

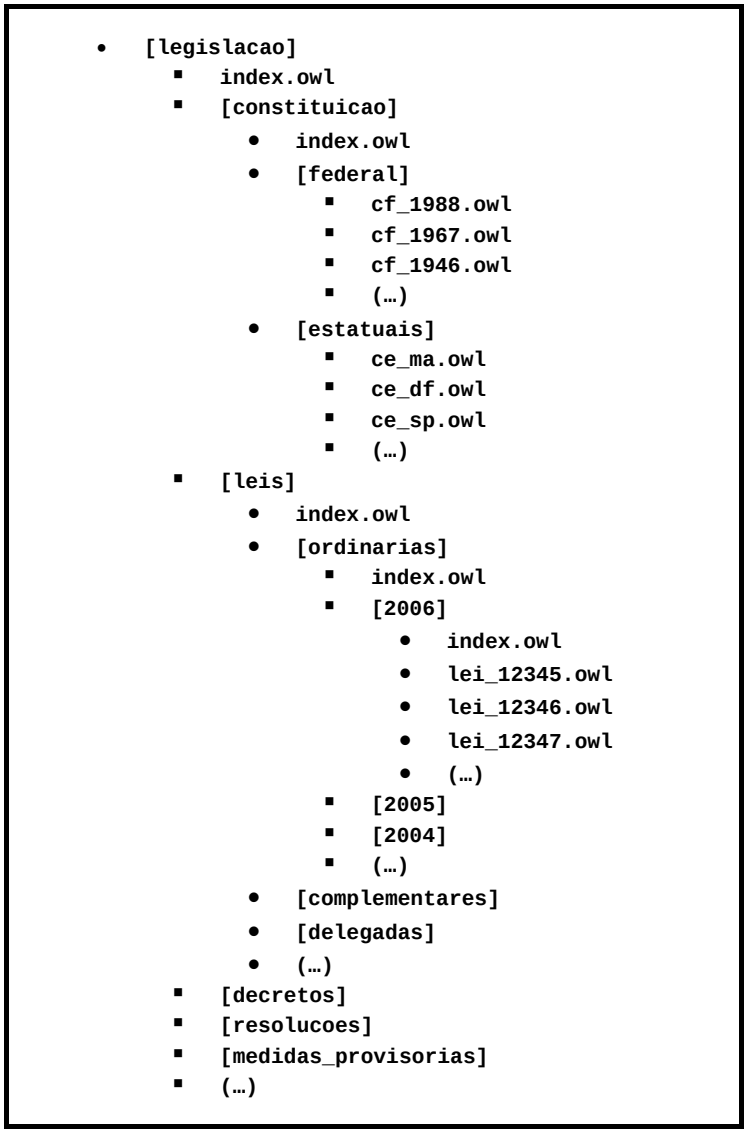


Figura 6.45: Exemplo da hierarquia de diretórios e arquivos da fonte de informação da *InfoNorma*

Um detalhe adicional é que tal objeto deve ser persistente, preservando seu estado atual, mesmo que a aplicação tenha sido finalizada, até que ela seja inicializada novamente. Para isso, será usada a serialização, através da qual um objeto pode ser armazenado em arquivo e recuperado no futuro, também por intermédio do JENA. No mais, é nesse objeto serializado que se propagarão as criações e as atualizações de índices de informação.

Há outra questão relativa ao gerenciamento da *InfoNorma*, dessa vez pelo especialista no domínio jurídico, o qual deve associar e manter atualizados conjuntos de termos-chave ponderados a cada uma das variadas categorias do Direito representadas na *OntoJuris* – Uma Ontologia de Ramos Jurídicos.

A *OntoJuris* representa as diversas categorias de ramos jurídicos, assim como os relacionamentos que cada um estabelece com os demais e os termos-chave associados a eles, tanto por ascendência quanto por descendência, segundo uma hierarquia de *frames* e seus respectivos *slots* (Figura 6.46).

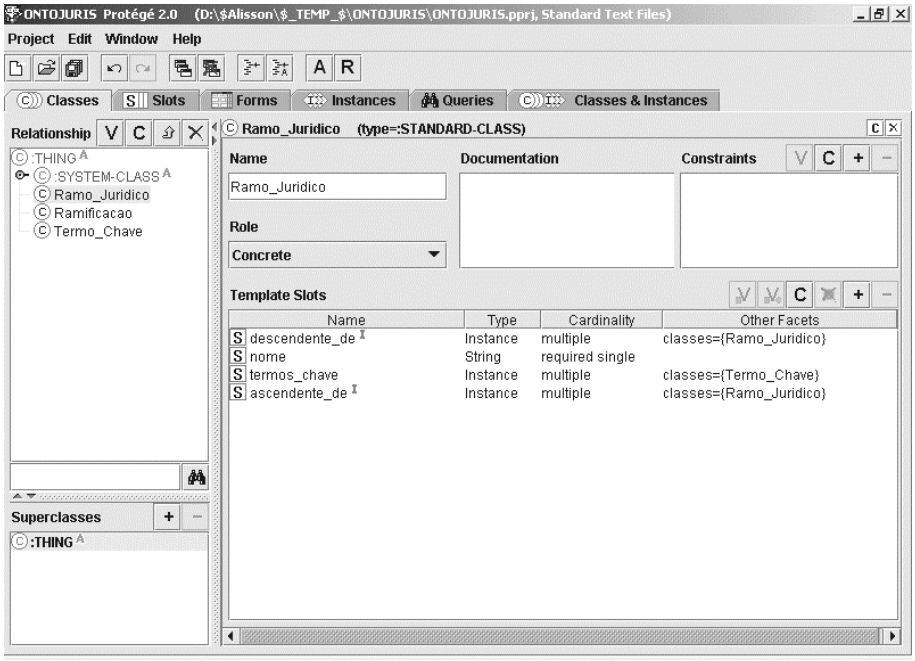


Figura 6.46: Hierarquia de *frames* e respectivos *slots* da *OntoJuris*

A instanciação da *OntoJuris* é realizada pelo especialista do domínio, o qual, de posse de seu conhecimento, define, para cada ramo jurídico, os ascendentes e descendentes relacionados, e os termos-chave associados, sendo um exemplo a categoria *Direito Penal* (Figura 6.47).

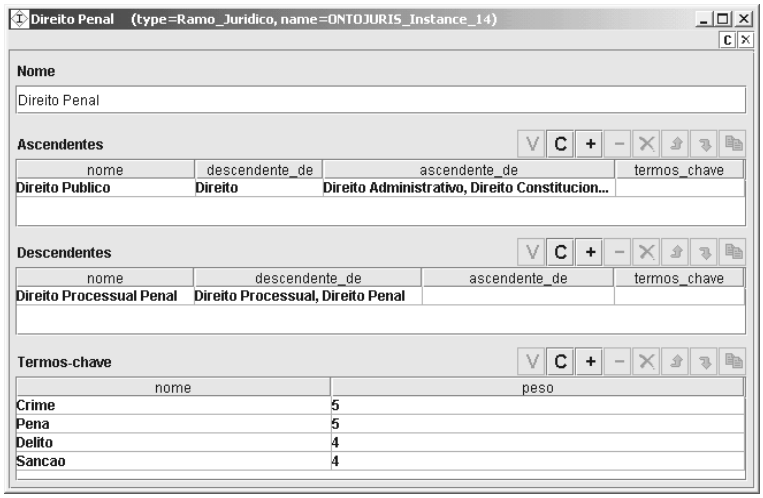


Figura 6.47: Exemplo de instanciação de um ramo jurídico na *OntoJuris*

Existe ainda na *OntoJuris* uma rede semântica (Figura 6.48) que mostra as ramificações do Direito, podendo-se nela observar como as diversas categorias jurídicas se relacionam.

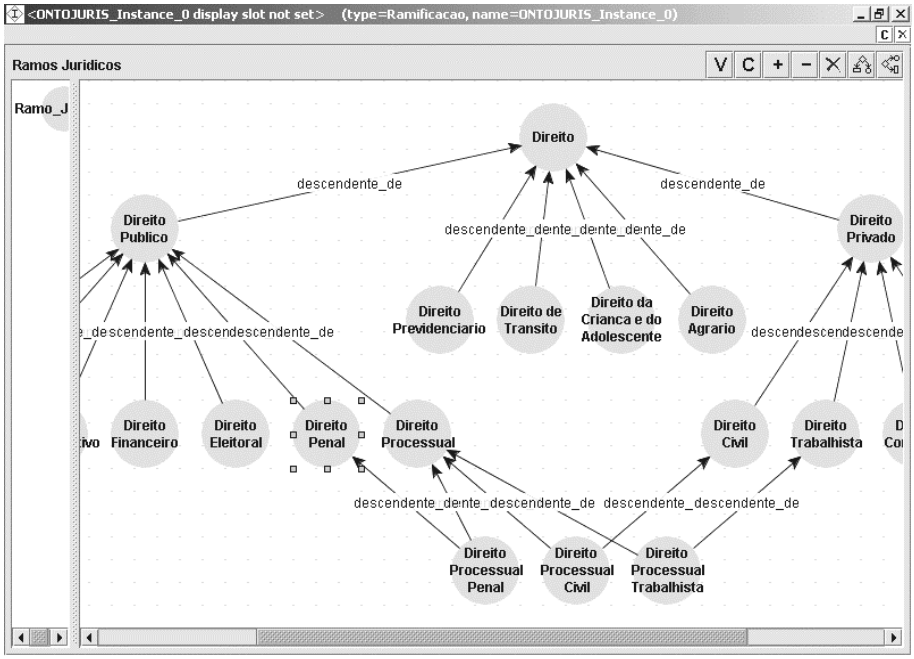


Figura 6.48: Rede semântica relacionando os ramos jurídicos na *OntoJuris*

Considerando que tais categorias jurídicas devem estar sempre acessíveis à aplicação para a filtragem, faz necessário que elas sejam representadas na forma de um objeto serializável, o qual não pode ser transiente, posto que precisa se conservar independentemente do funcionamento da aplicação. Tal é realizado através do JENA, da mesma forma já explicada anteriormente.

Por fim, no ANEXO 2 deste trabalho pode ser vista parte do código fonte da InfoNorma, com destaque para as classes dos agentes e dos comportamentos, desenvolvidas a partir do arcabouço provido pela plataforma JADE.

6.6 Considerações finais

Neste capítulo foi descrito o segundo estudo de caso elaborado com o intuito de avaliar a metodologia MAAEM e a ontologia ONTORMAS, dessa vez sem haver sido reutilizados quaisquer artefatos de software no domínio explorado, resultando modelada e construída a aplicação InfoNorma.

Assim, de início, delineou-se em que consistiria a mencionada aplicação, indo-se desde sua contextualização até um apanhado preliminar de seus requisitos, passando-se por uma breve fundamentação teórica.

Após isso, entrando-se propriamente na avaliação da referida metodologia, foram seguidas as orientações concernentes às fases de análise, projeto e implementação da aplicação, a partir das quais foram produzidos os respectivos modelos.

Enfim, construiram-se protótipos da InfoNorma e de suas aplicações auxiliares, as quais, uma vez codificadas, mostraram-se a princípio viáveis e funcionais, demonstrando potencial para atingir o fim esperado, tendo sido tudo devidamente relatado e ilustrado, conforme se pôde observar na seção imediatamente anterior.

Em geral, o desenvolvimento fluiu de maneira normal, sem grandes entraves, senão pela exigência de alguma familiariedade e prática com a parte gráfica do Protégé, para que ele seja empregado como ferramenta de suporte à modelagem de aplicações multiagente através da MAAEM e da ONTORMAS de forma realmente produtiva.

7 CONCLUSÕES

É crescente a demanda pela produção de software conciliando fatores muitas vezes conflitantes, tais como produtividade, custo e qualidade. Em face disso, não só é exigida a melhoria das técnicas e metodologias atuais, como também se faz necessária a elaboração de outras completamente novas, explorando abordagens distintas e inovadoras.

Nesse contexto, o paradigma de desenvolvimento orientado a agentes se mostra bastante atrativo. Isso porque 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, sendo as ontologias bastante adequadas para representar tais artefatos devido as suas peculiaridades (GIRARDI, 2004).

No presente trabalho, ao se propor a MAAEM, uma metodologia baseada em ontologias para a Engenharia de Aplicações Multiagente, tentou-se tirar proveito das características tanto dos agentes quanto das ontologias, elencadas no parágrafo anterior, revertendo-as em favor da Engenharia de Software, e ainda considerando a questão crucial da reutilização de software.

7.1 Principais contribuições

As principais características distintivas da MAAEM em relação às demais metodologias existentes são:

- primeiro, o fato de promover o reuso de abstrações de software de alto nível representando famílias de aplicações multiagente nos respectivos domínios;
- segundo, a adoção diversificada de ontologias para propósitos tais como:
  - na ONTORMAS, como base de conhecimento para representação dos artefatos de software reutilizáveis manipulados através da metodologia;
  - na *OntoLegis* e na *OntoJuris*, como estrutura de suporte ao desenvolvimento dos sistemas suplementares à *InfoNorma*;

- na comunicação dos agentes, para definição da semântica dos conceitos usados por eles nas trocas de mensagens entre si.

7.2 Resultados alcançados

O presente trabalho, além de ter permitido, de maneira geral, uma razoável compilação sobre o atual estado-da-arte dos temas que lhe embasam (Capítulos 2 e 3), apresentou como resultados especificamente relacionados com sua proposta os seguintes:

- revisão, atualização e sistematização dos processos da Engenharia de Domínio e Engenharia de Aplicações Multiagente (Seção 3.5);
- revisão, atualização e consolidação da metodologia MADEM (Seção 4.1), que se baseia em ontologias para a construção de artefatos de software reutilizáveis orientados a agentes, bem como da ontologia ONTOMADEM que a representa e lhe dá suporte;
- elaboração da metodologia MAAEM (Capítulo 4), que se baseia em ontologias para o desenvolvimento de aplicações multiagente através do reuso de tais artefatos, bem como da ontologia ONTOMAAEM que a representa e lhe dá suporte;
- integração das ontologias ONTOMADEM e ONTOMAEEM para formação da ONTORMAS (Seção 4.2 e ANEXO 1), de modo a simplificar o processo de reutilização dos produtos daquela como insumos desta;
- acompanhamento do processo de construção da aplicação multiagente *RecomTour* (Capítulo 5) e desenvolvimento pessoal da *InfoNorma* (Capítulo 6 e ANEXO 2), que exploram os casos de emprego da MAEEM e da ONTORMAS, respectivamente, *com* e *sem* reuso;
- revisão, atualização e reconstrução do modelo de domínio ONTOWUM-DM e do framework multiagente ONTOWUM-DD (APÊNDICE A).

7.3 Perspectivas futuras

No presente trabalho, empregou-se somente a abordagem composicional na reutilização de software para o desenvolvimento de sistemas multiagente. Nesse contexto, não há o que se esperar acerca de reuso automatizado através, por exemplo, de geradores de aplicações e linguagens específicas de domínio, que caracterizam a abordagem gerativa. De outro lado, tratou-se foi da composição manual de artefatos de software previamente elaborados, os quais, por ocasião da construção de uma aplicação multiagente qualquer, são devidamente selecionados e adaptados para esse fim.

Em face disso, planeja-se, futuramente, estender o escopo da atual pesquisa com o intuito de que a metodologia proposta passe a comportar também a reutilização gerativa, trazendo os benefícios a ela inerentes, tais como alta produtividade e desenvolvimento facilitado, dentre outros.

REFERÊNCIAS

ADOMAVICUS, G.; TUZHILIN, A. **Toward Next Generation of Recommender Systems**: a survey of the state-of-the-art and possible extensions. IEEE Transactions on knowledge and Data Engineering. V. 17, N. 6, 2005.

ALGERNON TAB. Rule-Based Programming. Disponível em: <http://algernon-j.sourceforge.net/doc/algernon-protege.html>. Acesso em: 07 fev. 2006.

BECKER, K.; VANZIN, M.. **Mineração do uso da Web**. In: Proceedings of the 19th Brazilian Symposium on Databases. Brasília, Brazil. October 18-22, 2004.

BELLIFEMINE, Fabio; CAIRE, Giovanni; TRUCCO; Tiziana; RIMASSA, Giovanni; MUNGENAST, Roland. **JADE Administrator's Guide**. Disponível em: <http://jade.tilab.com/doc/administratorsguide.pdf>. Acesso em: 24 nov. 2005.

BELLIFEMINE, Fabio; CAIRE, Giovanni; TRUCCO, Tiziana; RIMASSA, Giovanni. **JADE Programmer's Guide**. Disponível em: <http://jade.tilab.com/doc/programmersguide.pdf>. Acesso em: 24 nov. 2005.

BENI, Mário Carlos. **Análise Estrutural do Turismo**. 10. ed. São Paulo: SENAC, 428 p., 1998.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **Unified Modeling Language User Guide**. Reading, Addison Wesley. 1999.

BOULLOSA, J. R. F.. **Um Ambiente para Mineração de Utilização da Web**. Dissertação de Mestrado, Área de Banco de Dados. Universidade Federal do Rio de Janeiro (COPPE/UFRJ). 2002.

BRESCIANI, P.; *et al.*. **TROPOS: An Agent-Oriented Software Development Methodology**. In Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, Volume 8, Issue 3, pp. 203–236. May 2004.

BURKE, R.. **Knowledge-based Recommender Systems**. Encyclopedia of Library and Information Systems, Volume. 69, Supplement 32, Editor A. Kent, Marcel Dekker, 2000.

BURRAFATO, P.; COSSENTINO, M.. **Designing a Multi-agent Solution for a Bookstore with the PASSI Methodology**. In: Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002) at CAiSE'02. Toronto, Ontario, Canada. May 27 - 28, 2002.

CANTELE, Regina C.; *et al.*. **Reengenharia e Ontologias: Análise e Aplicação**. I Workshop de Web Semântica (WWS2004) no XIX Simpósio Brasileiro de Banco de Dados (SBBD'2004) e XVIII Simpósio Brasileiro de Engenharia de Software (SBES'2004). Brasília, DF, Brasil. 22 de Outubro de 2004.

CAIRE, Giovanni; *et al.*. **Agent Oriented Analysis using MESSAGE/UML**. Proceedings of Second International Workshop on Agent-oriented Software Engineering (AOSE 2001). Lecture Notes in Computer Science, Springer-Verlag GmbH, ISSN 0302-9743, V. 2222, p. 119. Montreal, Canada. May 2001.

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

CAZELA, Silvio César; REATEGUI, Eliseo Berni. **Sistemas de Recomendação**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul (UFRGS). Porto Alegre, 2004.

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

CHEN, M.; PARK, J. S.; YU, P. S.. **Efficient Data Mining for Path Traversal Patterns**. IEEE Transactions on Knowledge and Data Engineering. V. 10, N. 2, p. 209-221. March, 1998.

COPLIEN, J.; HOFFMAN, D.; WEISS, D.. **Commonality and variability in software engineering**. IEEE Software, pages 37-45, November/December 1998.

COSSENTINO, M.; POTTS, C.. **A CASE tool supported methodology for the design of multi-agent systems**. The 2002 International Conference on Software Engineering Research and Practice (SERP'02). Las Vegas, NV, USA. June 24 - 27, 2002.

COSSENTINO, M.; *et al.*. **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. Springer-Verlag, pp. 107-120. Efurt, Germany. October 9 -10, 2002.

COSSENTINO, M.; *et al.*. **Patterns reuse in the PASSI methodology**. Fourth International Workshop Engineering Societies in the Agents World (ESAW'03). Imperial College. London, UK. October 29-31, 2003.

CZARNECKI, Krzysztof; EISENECKER, Ulrich W.. **Generative Programming: Methods, Tools, and Applications**. ACM Press/Addison-Wesley Publishing Co., New York, NY, 2000.

DAM, Khanh Hoa; WINIKOFF, Michael. **Comparing Agent-Oriented Methodologies**, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS@AAMAS'03). July 2003.

DELOACH, Scott A.. **Analysis and Design using MaSE and agentTool**. Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001). Miami University, Oxford, Ohio, March 31 - April 1, 2001.

DELOACH, Scott A.; *et al.*. **Multiagent Systems Engineering**. The International Journal of Software Engineering and Knowledge Engineering, Volume 11 no. 3, June 2001.

DELOACH, Scott A.. **Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems**. In: Proceedings of Agent-Oriented Information Systems '99 (AOIS'99), pp 45-57 . Seattle WA, 1999.

DELOACH, Scott A.; WOOD, Mark. **Multiagent Systems Engineering: the Analysis Phase**. Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, June 2000.

DILEO, J.; JACOBS, T.; DELOACH, S.. **Integrating Ontologies into Multi-Agent Systems Engineering**, In: Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002), pp. 15-16. Bologna, Italy. July, 2002.

CMU-SEI. Domain Engineering. Carnegie Mellon University. Software Engineering Institute. Disponível em: < http://www.sei.cmu.edu/domain-engineering/domain_engineering.html >. Acesso em: 26 jul. 2004.

DOMAIN SPECIFIC REUSE WITH VHDL. Chapter 2. Disponível em: <http://soften.ktu.lt/~stuik/knyga/nchapter02.htm#Chapter2_3_4>. Acesso em: 17 mar. 2005.

FALBO, R. A.; GUIZZARDI G.; DUARTE, K. C.. **An Ontological Approach to Domain Engineering**. Proceedings of the XIV International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), ACM Press, pp. 351-358. Ischia, Italy. 2002.

FARIA, Carla Gomes de. **Uma Técnica para a Aquisição e Construção de Modelos de Domínio e Modelos de Usuários Baseados em Ontologias para a Engenharia de Domínio Multiagente**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2004.

FARIA, Carla Gomes de; OLIVEIRA, Ismênia Ribeiro de; GIRARDI, Rosario. **Especificação de uma Ontologia Genérica para a Construção de Modelos de Usuários**, Anais da 3ª Jornada Ibero-Americana de Engenharia de Software e Engenharia do Conhecimento (JIISIC 2003), Valdivia, Chile, Ed. LOM Ediciones Ltda., pp. 93-103, 2003.

FERREIRA, Steferson Lima Costa. **Uma Ferramenta e Técnica para o Projeto Global e Detalhado de Sistemas Multiagente**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2004.

FERREIRA, Steferson Lima Costa; GIRARDI, Rosario. **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 2003, Ed. Universidade del Bio-Bio. Chilán, Chile. 03 a 08 de novembro de 2003.

FIPA. Foundation for Intelligent Physical Agents. Disponível em: <http://www.fipa.org/>. Acesso em: 08 dez. 2005.

FIPA. ACL Message Structure Specification. Disponível em: <http://www.fipa.org/specs/fipa00061/SC00061G.html>. Acesso em: 15 dez. 2005.

FIPA. Communicative Act Library Specification. Disponível em: <http://www.fipa.org/specs/fipa00037/SC00037J.html>. Acesso em: 18 out. 2005.

GENNARI, J.; MUSEN, M. A.; FERGERSON, R. W.; *et al.*. **The Evolution of Protégé: An Environment for Knowledge-Based Systems Development**. Technical Report SMI-2002-0943. 2002.

GESEC. Grupo de pesquisa em Engenharia de Software e Engenharia do Conhecimento. Orientações. Disponível em: <http://maae.deinf.ufma.br/orientacoes/index.php>. Acesso em: 24 mai. 2005.

GEYER, Lars; BECKER, Martin. On the Influence of Variabilities on the Application-Engineering Process of a Product Family. In: Proceedings of the Second International Conference on Software Product Lines. Lecture Notes In Computer Science, V. 2379, p. 1-14. London: Springer-Verlag, 2002.

GILL, Nasib S.. **Reusability Issues in Component-based Development**. ACM SIGSOFT Software Engineering Notes, v. 28, n. 04, p. 4, ACM Press. New York, NY, USA. July 2003.

GIMENES, Itana Maria de Souza; TRAVASSOS, Guilherme Horta. **O Enfoque de Linha de Produto para Desenvolvimento de Software**. In: XXI Jornada de Atualização em Informática (JAI 2002), XXII Congresso da Sociedade Brasileira de Computação (CSBC 2002), <http://www.din.uem.br/~expsee/docs/artigos/Artigo-Itana_Travassos.pdf>. Florianópolis, Santa Catarina, Brasil. 15 a 19 de julho de 2002.

GIRARDI, Rosario. **Classification and Retrieval of Software through their Description in Natural Language**. PhD Thesis, University of Geneva, Switzerland, 1996.

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

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

GIRARDI, Rosario. **Engenharia de Software baseada em Agentes**. Anais do IV Congresso Brasileiro de Ciência da Computação (CBCOMP 2004), Ed. UNIVALI, pp. 913-937, Itajai, Santa Catarina, Brasil. 08 a 12 de outubro de 2004.

GIRARDI, Rosario, FARIA; Carla Gomes de. **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, Rosario; LINDOSO; Alisson Neres. **An Ontology-based Methodology for Multi-agent Domain Engineering**, In: 3rd Workshop on Multi-Agent Systems: Theory and Applications (MASTA 2005) at 12th Portuguese Conference on Artificial Intelligence (EPIA 2005), Ed. IEEE. Covilhã, Portugal. December 05-08, 2005.

GIRARDI, Rosario; LINDOSO, Alisson Neres. **An Ontology-driven Methodology and Tool for Domain Analysis and Design of Multi-agent Systems**. Artigo em processo de refinamento pelo jornal “Transactions on Software Engineering”. 2005.

GIRARDI, Rosario; LINDOSO, Alisson Neres. **Using Ontologies for the Representation and Reuse of Software Patterns**. Proceedings of 1st Workshop on Building A System Using Patterns (ECOOP 2005), Glasgow, Scotland, July 26, 2005.

GIRARDI, Rosario; LINDOSO, Alisson Neres. **A Domain Model of Web Recommender Systems based on Usage Mining and Collaborative Filtering**. Artigo submetido. 2005.

GIRARDI, Rosario; LINDOSO, Alisson Neres. **Architectural Modeling of Multi-agent Web Recommender Systems based on Usage Mining and Collaborative Filtering**. Artigo submetido. 2005.

GIRARDI, Rosario; MARINHO, Leandro Balby; LINDOSO, Alisson Neres. **WUMA – Miner: An Agent-based Design Pattern for Mining Users with Similar Navigational Behavior**, Anais da Quinta Conferência Latino-Americana em Linguagens de Padrões para Programação (SugarLoafPLoP`2005). Campos do Jordão, São Paulo, Brasil. 16 a 19 de agosto de 2005.

GIRARDI, Rosario; MARINHO, Leandro Balby; OLIVEIRA, Ismênia Ribeiro de. **A System of Agent-based Patterns for User Modeling based on Usage Mining**. Interacting with Computers Journal, v. 17, n. 7, p. 567-591, Elsevier. September 2005.

GIRARDI, Rosario; OLIVEIRA, Ismênia Ribeiro de; SILVA JUNIOR, Geovane Bezerra. **Towards a System of Patterns for the Design of Agent-based Systems**. Proceedings of The Second Nordic Conference on Pattern Languages of Programs (VikingPLoP 2003). Bergen, Norway, September 19th to 21st, 2003.

GIUNCHIGLIA, Fausto; MYLOPOULOS, John; PERINI, Anna. **The Tropos Software Development Methodology: Processes, Models and Diagrams**. Proceedings of Third International Workshop on Agent-oriented Software Engineering (AOSE 2002).

Lecture Notes in Computer Science, Springer-Verlag GmbH, ISSN 0302-9743, V. 2585, p. 162-173. Bologna, Italy. July 15, 2002.

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.

GÓMEZ-PÉREZ, A.; BENJAMINS, V.R, **Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods**. In: Proceedings of the International Joint Conference on Artificial Intelligence(IJCAI'99), Workshop on Ontologies and Problem-Solving Methods: Lesson learned and Future Trends (KRR5), V.R. Benjamins, *et al.*, Editors, CEUR Publications, Amsterdam, vol.18, pp.:1.1-1.15. Stockolm, Sweden. 1999.

GUANIRO, Nicola. **Formal Ontology in Information Systems**. In N.Guarino (ed.) Formal Ontology in Information Systems. Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998. IOS Press, Amsterdam: 3-15. 1998.

GUANIRO, Nicola. **Understanding, Building, and Using Ontologies: A Commentary to "Using Explicit Ontologies in KBS Development"**, by van Heijst, Schreiber, and Wielinga. International Journal of Human and Computer Studies (46): 293-310. 1997.

HARSU, Maarit. **A Survey of Domain Engineering**. Report 31, Institute of Software Systems, Tampere University of Technology. Dezembro de 2002.

HERLOCKER, J. L.; KONSTAN, J. A.; TERVEEN, L. G.; RIEDL, J. T.. **Evaluating collaborative filtering recommender systems**. In: ACM Transactions on Information Systems (TOIS), v. 22, n. 1, ACM Press. New York, NY, USA. January 2004.

HTML. HyperText Markup Language. Disponível em: <http://www.w3.org/MarkUp/>. Acesso em: 20 dez. 2005.

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

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

IGLESIAS, C.; GARIJO, M.; GONZÁLEZ, J. **A Survey of Agent-Oriented Methodologies**. In Intelligent Agents V. Agents Theories, Architectures, and Languages, Lecture Notes in Computer Science, vol. 1555, J. P. Müller, M. P. Singh, and A. S. Rao (Eds.), Springer-Verlag, 1998.

JADE. Java Agent DEvelopment Framework. Disponível em: <http://jade.tilab.com/>. Acesso em: 24 nov. 2005.

JAVASCRIPT. The Definitive JavaScript Resource. Disponível em: <http://www.javascript.com/>. Acesso em: 13 fev. 2006.

JAVA SOFTWARE. Disponível em: <http://www.java.com/ pt_BR/>. Acesso em: 05 dez. 2005.

JAVA TECHNOLOGY. Disponível em: < http://java.sun.com/>. Acesso em: 05 dez. 2005.

JENA. Semantic Web Framework for Java. Disponível em: <http://jena.sourceforge.net/>. Acesso em: 24 nov. 2005.

JENNINGS, N. R. **Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems**. The Knowledge Engineering Review, v. 8, n. 3, pp. 223-250. 1993.

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

KANG, K., COHEN, S.; HESS, J.; NOWAK, W.; PETERSON, S.. **Feature-Oriented Domain Analysis (FODA) Feasibility Study**. Technical Report, Software Engineering Institute, Carnegie Mellon University. Pittsburgh, Pennsylvania. November 1990.

KOBSA, Alfred. **User Modeling and User-Adapted Interaction**. In: Conference Companion on Human Factors in Computing Systems, p. 415-416. Boston, Massachusetts, United States. April 24 - 28, 1994.

KRUEGER, C. W.. **Software Reuse**. ACM Computing Surveys, 24(2): 131-183, June 1992.

LASSILA, O.; SWICK, R.. **Resouce Description Framework (RDF) Model and Syntax Specification**. In: W3C recommendation, World Wide Web Consortium. 1999.

LEGISLAÇÃO. Presidência da República Federativa do Brasil. Disponível em: <https://www.planalto.gov.br/legisla.htm>. Acesso em: 05 dez. 2005.

LEI Nº 9.099, DE 26 DE SETEMBRO DE 1995. Presidência da República. Subchefia para Assuntos Jurídicos. Disponível em: <http://www.presidencia.gov.br/ccivil_03/ Leis/L9099.htm>. Acesso em: 12 fev. 2006.

LINDOSO, Alisson Neres. **Desenvolvimento de um Modelo de Domínio Baseado em Ontologias para o Acesso à Informação Jurídica**. Monografia (Graduação em Ciência da Computação), Universidade Federal do Maranhão (CGCC/UFMA). São Luís-MA. 2003.

LINDOSO, Alisson Neres; GIRARDI, Rosario; OLIVEIRA, Ismênia Ribeiro de. Uma Experiência no Projeto de um Framework Multiagente para a Recuperação e Filtragem de Informação, In: Anais do IV Congresso Brasileiro de Computação (CBCOMP 2004), Ed. UNIVALI, pp. 124-129. Itajaí, Santa Catarina, Brasil. 08 a 12 de outubro de 2004.

MARINHO, Leandro Balby. **Um Framework Multiagente para a Personalização da Web Baseada na Modelagem de Usuários e na Mineração de Uso**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2005.

MASSONET, Philippe; DEVILLE, Yves; NÈVE, Cédric. **From AOSE Methodology to Agent Implementation**. Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1, pp. 27-34, 2002.

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

MYLOPOULOS J.; CASTRO, J.; KOLP M.. **Tropos: Towards Agent-Oriented Information Systems Engineering**. Position Paper at the Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2000). Stockolm, Sweden. June 5-6, 2000.

NOVELLO, Taisa Carla. **Ontologias, sistemas baseados em conhecimento e modelos de banco de dados**. Disponível em: <http://www.inf.ufrgs.br/~clesio/cmp151/cmp15120021/artigo_taisa.pdf>. Acesso em: 10 mar. 2003.

ODELL, J.; PARUNAK, H.; BAUER, B. **Extending UML for Agents**. In: Proceedings of the Agent-Oriented Information Systems Workshop (AOIS 2000) at the 17th National Conference on Artificial Intelligence (AAAI), accepted role, pp. 3-17. July 2000.

OLIVEIRA, Ismênia Ribeiro de. **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), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2004.

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

ONTOVIZ TAB. Visualizing Protégé Ontologies. Disponível em: <http://protege.stanford.edu/plugins/ontoviz/ontoviz.html>. Acesso em: 07 fev. 2006.

OWL. Ontology Web Language. Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em: 20 dez. 2005.

QUEIROZ, Sergio Ricardo de Melo. **Recomendação para Grupos através de Filtragem Colaborativa**. Trabalho de Graduação em Inteligência Artificial. Universidade Federal de Pernambuco (UFPE). Recife, 2002.

PHP. Hypertext Preprocessor. Disponível em: <http://www.php.net/>. Acesso em: 13 fev. 2006.

PIERRAKOS, D.; PALIOURAS, G.; PAPATHEODOROU, C.; SPYROPOLOUS, C. D.. **Web Usage Mining as a Tool for Personalization**: a survey, user modeling and user-adapted interaction. 13: 311-372. 2003.

PONT, M. J.; MOREALE, E.. **Towards a Practical Methodology for Agent-Oriented Software Engineering with C++ and Java**. Technical Report, TR 96-33, Department of Engineering, Leicester University. 1996.

RDF. Resource Description Framework. Disponível em: <http://www.w3.org/RDF/>. Acesso em: 20 dez. 2005.

RUSSELL, Stuart; NORVIG, Peter. Inteligência Artificial. 2. ed. São Paulo: Campus, 2004. 1040 p.

SERRA, Ivo José da Cunha. **Uma Técnica para o Desenvolvimento de Linguagens Específicas de Domínio**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2004.

SERRA JUNIOR, Gentil Cutrim. **Agente de Modelagem do Aprendiz para o sistema MATHNET de Ensino Inteligente Cooperativo Computadorizado**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2001.

SERRA, Ivo José da Cunha; GIRARDI, Rosario; SILVA FILHO, José Henrique da Silva. **Um Modelo de Domínio baseado em Ontologias para a Recuperação e Filtragem de Informação**. Anais do IV Congresso Brasileiro de Computação (CBCOMP 2004), Ed. UNIVALI, pp. 124-129. Itajaí, Santa Catarina, Brasil. 08 a 12 de outubro de 2004.

SHAHABI, C.; BANAEI-KASHANI, F.. **Efficient and Anonymous Web Usage Mining for Web Personalization**. In: Informs Journal on Computing, Special Issue on Data Mining, v. 15, n. 2, Spring. 2003.

SILVA JUNIOR, Geovane Bezerra. **Padrões Arquiteturais para o Desenvolvimento de Aplicações Multiagente**. Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação), Universidade Federal do Maranhão (CPGEE/UFMA). São Luís-MA. 2003.

SPARLING, Michael. **Lessons learned through six years of component-based development**. Communications of the ACM, v. 43, n. 10, p. 47-53, ACM Press. New York, NY, USA. October 2000.

STURM, Arnon; SHEHORY, Onn. **A Framework for Evaluating Agent-Oriented Methodologies**, In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS@AAMAS'03). July 2003.

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

THE PROTÉGÉ PROJECT. Disponível em: <http://protege.stanford.edu/>. Acesso em: 26 jul. 2005.

TORRES, Roberto. **Personalização na Internet**. São Paulo: Novatec, 2004.

UEMURA, Keiji; OHORI, Miki.. **A cooperative approach to software development by application engineers and software engineers**. In: Proceedings of the 7th International Conference on Software Engineering. Orlando, Florida, United States, 1984.

VITHARANA, Padmal. **Risks and challenges of component-based software development**. Communications of the ACM, v. 46, n. 08, p. 67-72, ACM Press. New York, NY, USA. August 2003.

VITHARANA, Padmal; ZAHEDI, Fatemah "Mariam", JAIN, Hemant. **Design, retrieval, and assembly in component-based software development**. Communications of the ACM, v. 46, n. 11, p. 97-102, ACM Press. New York, NY, USA. November 2003.

WANG, Ju An. **Towards component-based software engineering**. In: Proceedings of the eighth annual consortium on Computing in Small Colleges Rocky Mountain conference, Consortium for Computing Sciences in Colleges, p. 177-189. Orem, Utah, United States. 2000.

WITHEY, James V. **Implementing Model-Based Software Engineering in Your Organization: An Approach to Domain Engineering. Pittsburgh**, PA: Software Engineering Institute, Carnegie Mellon University, April 1994.

WOOD, Mark; DELOACH, Scott A.. **An Overview of the Multiagent Systems Engineering Methodology**. In Agent-Oriented Software Engineering - Proceedings of the First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), 10th June 2000, Limerick, Ireland. P Cicarini, M. Woodridge, editors. Lecture Notes in Computer Science. Vol. 1957, Springer Verlag, Berlin, January, 2001.

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

WOOLDRIDGE M. J.; JENNINGS, N. R.; KINNY, D.. **The Gaia methodology for agent-oriented analysis and design**. Autonomous Agents and Multi-Agent Systems, 3(3): 285–312, September 2000.

XML. Extensible Markup Language. Disponível em: <http://www.w3.org/XML/>. Acesso em: 20 dez. 2005.

YAMAGUCHI, Takahira; KUREMATSU, Masaki. **A Legal Ontology Rapid Development Environment Using a Machine Readable Dictionary**. In: 1st International Workshop on Legal Ontologies. School of Information Shizuoka University. Japão, 1997.

ZAMBONELLI, Franco; et al.. **Developing Multiagent systems: The Gaia methodology**. ACM Transactions on Software Engineering and Methodology, 12(3), pages 417-470. 2003.

ANEXO 1 – Hierarquia de classes da ontologia *ONTORMAS*

- THING
  - Modeling Concepts
    - Basic Concepts
      - Architectural Style
        - Blackboard
        - Broker
        - Facilitator
        - Federation
        - Layer
        - Master
        - Negotiator
        - Slave
      - Bibliographical Reference
      - Life Line
      - State
        - Final State
        - Initial State
    - Implementation Concepts
      - FIPA-ACL Message
      - JADE Agent
      - JADE Behaviour
        - JADE CyclicBehaviour
        - JADE OneShotBehaviour
        - JADE SimpleBehaviour
    - Interrelated Concepts
      - Activity
      - Condition
      - Domain Concept
      - External Entity
      - Goal
        - General Goal
        - Specific Goal
      - Internal Entity
        - Agent
        - Role
      - Knowledge
      - Responsibility
      - Skill
    - Variable Concepts
      - Activity
      - Domain Concept
      - Responsibility
      - Skill
      - Specific Goal
  - Modeling Products
    - Composed Products
      - Agent Model
      - Application Engineering Products
        - Application Architecture
        - Application Specification
      - Architectural Model
      - Domain Engineering Products
        - Domain Model
        - Multi-agent Framework
        - Software Patterns and Pattern Systems

        - Simple Products
            - Agent Interaction Model
            - Agent Knowledge and Activity Model
            - Agent State Model
            - Concept Model
            - Coordination and Cooperation Mechanisms Model
            - Goal Model
            - Multi-agent Society Knowledge Model
            - Multi-agent Society Model
            - Pattern System
            - Role Interaction Model
            - Role Model
            - Software Pattern
                - Analysis Pattern
                - Architectural Pattern
                - Design Pattern
                - Idiom
                - Organizational and Process Pattern
    - Modeling Tasks
        - Composed Tasks
            - Agent Design
            - Application Engineering Tasks
                - Application Analysis
                - Application Design
                - Application Implementation
            - Architectural Design
            - Domain Engineering Tasks
                - Domain Analysis
                - Domain Design
                - Pattern Extraction and Representation
        - Simple Tasks
            - Agent Interaction Modeling
            - Agent Knowledge and Activity Modeling
            - Agent State Modeling
            - Concept Modeling
            - Cooperation and Coordination Mechanisms Modeling
            - Goal Modeling
            - Multi-agent Society Knowledge Modeling
            - Multi-agent Society Modeling
            - Pattern System Modeling
            - Role Interaction Modeling
            - Role Modeling
            - Software Pattern Modeling
    - :SYSTEM-CLASS
        - :ANNOTATION
            - :INSTANCE-ANNOTATION
        - :CONSTRAINT
            - :PAL-CONSTRAINT
        - :META-CLASS
            - :CLASS
                - :STANDARD-CLASS
            - :FACET
                - :STANDARD-FACET
            - :SLOT
                - :STANDARD-SLOT
        - :RELATION
            - :DIRECTED-BINARY-RELATION
                - Modeling Relationships
                    - Communication Relationships
                        - Incoming Communication

- Outgoing Communication
- Concept Relationships
- Condition Relationships
  - Post-condition
  - Pre-condition
- Interaction Relationships
  - Agent Interaction Performative
  - External Event Notification
  - Role Interaction
- Knowledge Relationships
  - Produced Knowledge
  - Used Knowledge
- Other Relationships
  - Activity Flow
  - Gets Information
  - Has Knowledge
  - State Transition
- Pattern Relationships
  - Refines
  - Replaces
  - Requires
  - Uses
- Solution Relationships

Project: ONTORMAS
Class Specific Goal

Concrete Class Extends
Goal, Variable Concepts

Direct Instances:

1. [Satisfy dynamic information need through retrieval techniques]
2. [Satisfy long-term user information need through filtering techniques]
3. [Satisfy long-term user information need through content-based filtering]
4. [Satisfy long-term user information need through collaborative filtering]
5. [Model users through Usage Mining]
6. [Model adaptation through Usage Mining]
7. [Model explicitly users]
8. [Filter information based on the content]

Direct Subclasses:
None

Template Slots					
Slot name	**Documentation**	**Type**	**Allowed Values/Classes**	**Cardinality**	**Default**
reached from		Instance	Specific Goal	0:*	
non-functional requirements		String		0:*	
synonym of		Instance	Modeling Concepts	0:*	
specialization of		Instance	Modeling Concepts	0:*	

generalization of		Instance	Modeling Concepts	0:*	
description		String		0:1	
name		String		1:1	
variability type		Symbol	mandatory, alternative, optional	0:1	
alternative or optional		Instance	Specific Goal	0:*	
leads to		Instance	General Goal, Specific Goal	1:1	
achieved by		Instance	Responsibility	0:*	

Return to class hierarchy

ANEXO 2 – Listagem de código-fonte da aplicação *InfoNorma*

```
package entidadeUsuario;

public class Usuario {

}
```

```
package camadaProcessamentoUsuario.agenteInterfaceador;

import jade.core.Agent;


public class Interfaceador extends Agent {

	protected void setup() {
		addBehaviour(new AdquirirPerfisUsuario(this));
		addBehaviour(new EntregarInformacoesFiltradas(this));
	}

}
```

```
package camadaProcessamentoUsuario.agenteInterfaceador;

import jade.core.behaviours.CyclicBehaviour;


public class AdquirirPerfisUsuario extends CyclicBehaviour {


	public AdquirirPerfisUsuario(Interfaceador agenteInterfaceador) {
		super(agenteInterfaceador);
	}

	public void action() {
		//VALIDAR PERFIS DE USUÁRIO

		//ENVIAR PERFIS DE USUÁRIO PARA O AGENTE MODELADOR
	}
}
```

```
package camadaProcessamentoUsuario.agenteInterfaceador;
```

```
import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import recursos.InstrumentoNormativo;
import recursos.RamoJuridico;
import recursos.satisfacao.Satisfacao;
import recursos.satisfacao.SatisfacaoOntology;


public class EntregarInformacoesFiltradas extends CyclicBehaviour {


    public EntregarInformacoesFiltradas(Interfaceador agenteInterfaceador) {
          super(agenteInterfaceador);
    }

       public void action() {
              //RECEBER SATISFAÇÕES POR USUÁRIO DO AGENTE FILTRADOR
              Satisfacao satisfacao = receberSatisfacaoUsuario();

              if(satisfacao != null) {
                     //ENCAMINHAR SATISFAÇÕES PARA O USUÁRIO
//                   encaminharSatisfacaoUsuario(satisfacao);
              }
       }


       private Satisfacao receberSatisfacaoUsuario() {
              Satisfacao satisfacao = null;

              Codec codec = new SLCodec();
              Ontology ontology = SatisfacaoOntology.getInstance();

              myAgent.getContentManager().registerLanguage(codec);
              myAgent.getContentManager().registerOntology(ontology);

              MessageTemplate modeloInformacao =
MessageTemplate.and(MessageTemplate.MatchLanguage(codec.getName()),MessageTemplate.MatchOntology(ontology.getName()));

              ACLMessage mensagem = myAgent.receive(modeloInformacao);
```

```
		ContentElement conteudo = null;
		if(mensagem != null)
			try {
				conteudo = myAgent.getContentManager().extractContent(mensagem);
			} catch (UngroundedException e) {
				e.printStackTrace();
			} catch (CodecException e) {
				e.printStackTrace();
			} catch (OntologyException e) {
				e.printStackTrace();
			}

		if(conteudo instanceof Satisfacao) {
			satisfacao = (Satisfacao) conteudo;

			String  usuario 		= satisfacao.getUsuario(),
				tipo 			= InstrumentoNormativo.obterDescricao(satisfacao.getTipo()),
				categoria 		= RamoJuridico.obterDescricao(satisfacao.getCategoria()),
				numero 			= satisfacao.getNumero(),
				data 			= satisfacao.getData(),
				endereco 		= satisfacao.getEndereco(),
				similaridade	= satisfacao.getSimilaridade();

			System.out.println("\n# Filtrador => ==========================================================================================#");
			System.out.println("Tipo: " + tipo + " Nº " + numero + ", de " + data);
			System.out.println("Categoria: " + categoria);
			System.out.println("Usuário: " + usuario);
			System.out.println("Similaridade: " + similaridade);
			System.out.println("Vínculo: " + endereco);
			System.out.println("#==========================================================================================#\n");
		}

		return satisfacao;
	}
}
```

```
package camadaProcessamentoUsuario.agenteModelador;

import jade.core.*;


public class Modelador extends Agent {

	protected void setup() {
		addBehaviour(new ManterModelosUsuarios(this));
	}

}
```

```
package camadaProcessamentoUsuario.agenteModelador;

import jade.content.lang.Codec;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.util.Iterator;
import java.util.Vector;

import recursos.RamoJuridico;
import recursos.necessidade.Necessidade;
import recursos.necessidade.NecessidadeOntology;


public class ManterModelosUsuarios extends CyclicBehaviour {


	public ManterModelosUsuarios(Modelador agenteModelador) {
		super(agenteModelador);
	}

	public void action() {
		//RECEBER DADOS DE INSTRUMENTO NORMATIVO DO AGENTE FILTRADOR
		String dadosInstrumentoNormativo = receberDadosInstrumentoNormativo();

		if(dadosInstrumentoNormativo != null) {
			String[] string = dadosInstrumentoNormativo.split("_");
			int      tipoInstrumentoNormativo   = new Integer(string[0]).intValue();
			String   numeroInstrumentoNormativo = string[1];

			Iterator necessidades = extrairNecessidadesUsuario(tipoInstrumentoNormativo,numeroInstrumentoNormativo);

			while(necessidades.hasNext()) {
				//ENVIAR NECESSIDADES POR USUÁRIO PARA O AGENTE FILTRADOR
				enviarNecessidadeUsuario((Necessidade) necessidades.next());
			}
		}
	}


	private String receberDadosInstrumentoNormativo() {
		String dadosInstrumentoNormativo = null;
```

```
        MessageTemplate modeloMensagem = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        ACLMessage mensagem = myAgent.receive(modeloMensagem);

        String conteudo = null;
        if(mensagem != null)
                conteudo = mensagem.getContent();

        if(conteudo != null) {
                dadosInstrumentoNormativo = conteudo;
                //System.out.println("Filtrador => : " + dadosInstrumentoNormativo);
        }

        return dadosInstrumentoNormativo;
}

private Iterator extrairNecessidadesUsuario(int tipoInstrumentoNormativo, String numeroInstrumentoNormativo) {
        Vector necessidades = new Vector();

        Necessidade necessidade;

        necessidade = new Necessidade();
        necessidade.setUsuario("Alisson Neres Lindoso");
        necessidade.setIdentificador(numeroInstrumentoNormativo);
        necessidade.addCategorias(RamoJuridico.PENAL);
        necessidades.add(necessidade);

        necessidade = new Necessidade();
        necessidade.setUsuario("Rafael Robinson de Sousa Neto");
        necessidade.setIdentificador(numeroInstrumentoNormativo);
        necessidade.addCategorias(RamoJuridico.CIVIL);
        necessidade.addCategorias(RamoJuridico.AMBIENTAL);
        necessidades.add(necessidade);

        necessidade = new Necessidade();
        necessidade.setUsuario("Leandro Balby Marinho");
        necessidade.setIdentificador(numeroInstrumentoNormativo);
        necessidade.addCategorias(RamoJuridico.TRABALHISTA);
        necessidades.add(necessidade);

        return necessidades.iterator();
}

private void enviarNecessidadeUsuario(Necessidade necessidade) {
        Codec codec = new SLCodec();
        Ontology ontology = NecessidadeOntology.getInstance();

        myAgent.getContentManager().registerLanguage(codec);
        myAgent.getContentManager().registerOntology(ontology);

        ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM_REF);
```

```
                mensagem.addReceiver(new AID("filtrador",AID.ISLOCALNAME));
                mensagem.setLanguage(codec.getName());
                mensagem.setOntology(ontology.getName());

                try {
                        myAgent.getContentManager().fillContent(mensagem,necessidade);
                        myAgent.send(mensagem);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                //System.out.println("=> Filtrador : " + necessidade.getUsuario());
                /*
                Iterator categorias = necessidade.getAllCategorias();
                int i = 1;
                while(categorias.hasNext()) {
                        Integer categoria = (Integer) categorias.next();
                        System.out.println("[" + i++ + "] Categoria: " + RamoJuridico.obterDescricao(categoria.intValue()));
                }
                */
        }
}
```

```
package entidadeFonteInformacao;

import java.io.IOException;
import java.net.*;
import java.util.*;


public class FonteInformacao {

        private Vector indices;

        public FonteInformacao() {
                indices = new Vector();
        }

        public int obterTamanhoIndices() {
                return indices.size();
        }

        public void adicionarIndice(String endereco) throws IOException {
                Vector temp = new Vector(3);
                temp.add(endereco);
                temp.add(new Long(new URL(endereco).openConnection().getLastModified()));
                temp.add(new String(""));
                indices.add(temp);
```

```
	}

	public void adicionarIndice(String endereco, String ponteiro) throws IOException {

		Vector temp = new Vector(3);
		temp.add(endereco);
		temp.add(new Long(new URL(endereco).openConnection().getLastModified()));
		temp.add(ponteiro);
		indices.add(temp);
	}

	public void removerIndice(int i) {
		indices.remove(i);
	}

	public String obterEndereco(int i) {
		Vector temp = (Vector) indices.get(i);
		return (String) temp.get(0);
	}

	public Long obterData(int i) {
		Vector temp = (Vector) indices.get(i);
		return (Long) temp.get(1);
	}

	public String obterPonteiro(int i) {
		Vector temp = (Vector) indices.get(i);
		return (String) temp.get(2);
	}

	public void ajustarEndereco(int i, URL endereco) {
		Vector temp = (Vector) indices.get(i);
		temp.set(0, endereco);
		indices.set(i,temp);
	}

	public void ajustarData(int i) throws IOException {
		Vector temp = (Vector) indices.get(i);
		String endereco = (String) temp.get(0);
		temp.set(1, new Long(new URL(endereco).openConnection().getLastModified()));
		indices.set(i,temp);
	}

	public void ajustarPonteiro(int i, String ponteiro) {
		Vector temp = (Vector) indices.get(i);
		temp.set(2, ponteiro);
		indices.set(i,temp);
	}
}
```

```
package camadaProcessamentoInformacao.agenteMonitor;

import jade.core.*;


public class Monitor extends Agent {

	protected void setup() {
		addBehaviour(new MonitorarFontesInformacao(this));
	}
}
```

```
package camadaProcessamentoInformacao.agenteMonitor;

import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.Iterator;
import java.util.Vector;
import recursos.Parser;
import entidadeFonteInformacao.FonteInformacao;


public class MonitorarFontesInformacao extends CyclicBehaviour {

	private FonteInformacao fonteInformacao = new FonteInformacao();
	private String ponteiro,
				caminho = "file:\\C:\\Arquivos de programas\\eclipse\\workspace\\infonorma\\legislacao\\lei\\2004\\";
//				caminho = "http://www.planalto.gov.br/ccivil_03/Leis/2003/";
//				caminho = "http://www.planalto.gov.br/ccivil_03/MPV/Quadro/";
//				caminho = "http://www.planalto.gov.br/ccivil_03/decreto-lei/1965-1988/";
//				caminho = "http://www.planalto.gov.br/ccivil_03/_Ato2004-2006/2004/Decreto/";

	public MonitorarFontesInformacao(Monitor agenteMonitor) {
		super(agenteMonitor);
		try {
			fonteInformacao.adicionarIndice(caminho + "index.html","10.859");
//			fonteInformacao.adicionarIndice("file:\\C:\\Arquivos de
programas\\eclipse\\workspace\\infonorma\\legislacao\\Leis_Ordinarias_de_2003.htm","10.827");
//			fonteInformacao.adicionarIndice("file:\\C:\\Arquivos de
programas\\eclipse\\workspace\\infonorma\\legislacao\\MP_Tramitacao_2.htm","191");
//			fonteInformacao.adicionarIndice("file:\\C:\\Arquivos de
programas\\eclipse\\workspace\\infonorma\\legislacao\\Quadro1965_1988.htm","2.476");
```

```
//                    fonteInformacao.adicionarIndice("file:\\C:\\Arquivos de
programas\\eclipse\\workspace\\infonorma\\legislacao\\Quadro_decreto_2004.htm","5.118");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }

        public void action() {
                try {
                        //DETECTAR ATUALIZAÇÕES NOS ÍNDICES DE INFORMAÇÃO
                        for (int indice = 0; indice < fonteInformacao.obterTamanhoIndices(); indice++){
                                String  endereco        = fonteInformacao.obterEndereco(indice);
                                Long    dataAnterior = fonteInformacao.obterData(indice),
                                                dataAtual       = new Long(new URL(endereco).openConnection().getLastModified());

                                ponteiro = fonteInformacao.obterPonteiro(indice);

                                if(alteracaoDetectada(dataAtual,dataAnterior)) {
                                        Vector  novosElementosReversos = identificarNovosElementos(endereco,ponteiro),
                                                        novosElementosCorretos = new Vector();
                                        for(int i=novosElementosReversos.size()-1; i>=0; i--)
                                                novosElementosCorretos.add(novosElementosReversos.get(i));

                                        Iterator novosElementos = novosElementosCorretos.iterator();
                                        fonteInformacao.ajustarPonteiro(indice,ponteiro);
                                        fonteInformacao.ajustarData(indice);

                                        while(novosElementos.hasNext()) {
                                                //ENVIAR ELEMENTO DE INFORMAÇÃO PARA O AGENTE CONSTRUTOR
                                                enviarElementoInformacao((String) novosElementos.next());
                                        }
                                }
                        }

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }


        private boolean alteracaoDetectada(Long dataAtual, Long dataAnterior) {
                        if(dataAtual.longValue() > dataAnterior.longValue())
                                return true;
                        else
                                return false;
        }

        private Vector identificarNovosElementos(String endereco, String ponteiroAnterior) {
                Vector novosElementos = new Vector();
```

```
		try {
			BufferedReader indice = new BufferedReader(new InputStreamReader(new URL(endereco).openStream()));

			String  linha,
				segmento = "",
				ponteiroAtual = null;

			boolean detectandoElemento = false;

			while ((linha = indice.readLine()) != null) {
				String copia = linha;
				linha = linha.toLowerCase();

				if(linha.indexOf("<tr") != -1) {
					detectandoElemento = true;
				}

				if(detectandoElemento) {
					if(linha.indexOf("<a") != -1) {
						int ini = linha.indexOf("<a");
						if(linha.indexOf("a>") != -1) {
							int fim = linha.indexOf("a>") + 2;
							segmento = copia.substring(ini,fim);
							detectandoElemento = false;
						}else {
							segmento = copia.substring(ini);
						}
					}else if(linha.indexOf("a>") != -1) {
						int fim = linha.indexOf("a>") + 2;
						segmento += copia.substring(0,fim);
						detectandoElemento = false;
					}else if(segmento.length() != 0) {
						segmento += copia.substring(0);
					}

					if(!detectandoElemento) {
						String novoElemento = Parser.stripHTMLTagsString(segmento.substring(segmento.indexOf("\">") + 2,
segmento.indexOf(",")));

						if(!novoElemento.equals(ponteiroAnterior)) {
							novosElementos.add(caminho + segmento.substring(segmento.indexOf("href=\"") + 6,
segmento.indexOf("\">")));
							if(novosElementos.size() == 1)
								ponteiroAtual = novoElemento;
							segmento = "";
						} else {
							break;
						}
					}
				}
```

```
                        }

                        if(ponteiroAtual != null)
                                ponteiro = ponteiroAtual;
                        else
                                ponteiro = ponteiroAnterior;

                }catch (IOException e) {

                }

                return novosElementos;
        }

        private void enviarElementoInformacao(String elementoInformacao) {
                //ACLMessage mensagem = new ACLMessage();
                //mensagem.setPerformative(ACLMessage.INFORM);
                ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM);
                mensagem.addReceiver(new AID("construtor",AID.ISLOCALNAME));
                mensagem.setContent(elementoInformacao);
                myAgent.send(mensagem);

                System.out.println("=> Construtor : " + elementoInformacao);
        }
}
```

```
package camadaProcessamentoInformacao.agenteConstrutor;

import jade.core.Agent;


public class Construtor extends Agent {

        protected void setup() {
                addBehaviour(new  RepresentarElementosInformacao(this));
        }
}
```

```
package camadaProcessamentoInformacao.agenteConstrutor;

import recursos.*;
import recursos.surrogate.*;
import java.io.IOException;
import java.util.*;
import jade.content.lang.Codec;
```

```
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.*;
import jade.core.*;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.*;


public class RepresentarElementosInformacao extends CyclicBehaviour {

	public RepresentarElementosInformacao(Construtor agenteConstrutor) {
		super(agenteConstrutor);
	}

	public void action() {
		//RECEBER ELEMENTO DE INFORMAÇÃO DO AGENTE MONITOR
		String elementoInformacao = receberElementoInformacao();

		if(elementoInformacao != null)
			try {
				//ENVIAR REPRESENTAÇÃO INTERNA PARA O AGENTE FILTRADOR
				enviarRepresentacaoInterna(criarRepresentacaoInterna(elementoInformacao));
			} catch (IOException e) {
				e.printStackTrace();
			}
	}


	private String receberElementoInformacao() {
		String elementoInformacao = null;

		MessageTemplate modeloMensagem = MessageTemplate.MatchPerformative(ACLMessage.INFORM);
		ACLMessage mensagem = myAgent.receive(modeloMensagem);

		String conteudo = null;
		if(mensagem != null)
			conteudo = mensagem.getContent();

		if(conteudo != null) {
			elementoInformacao = conteudo;
			//System.out.println("Monitor => : " + elementoInformacao);
		}

		return elementoInformacao;
	}

	private Surrogate criarRepresentacaoInterna(String elementoInformacao) throws IOException {
		//CONSTRUIR SURROGATE DO NOVO ELEMENTO DE INFORMAÇÃO
		Surrogate representacaoInterna = new Surrogate();
```

```
		String  identification = Parser.extractIdentification(elementoInformacao);

		representacaoInterna.setTipo(Parser.getType(identification));
		representacaoInterna.setNumero(Parser.getNumber(identification));
		representacaoInterna.setData(Parser.getDate(identification));
		representacaoInterna.setEndereco(elementoInformacao);

		Parser.stripHTMLTagsFile(elementoInformacao);
		for(Enumeration e = Parser.extractKeywords().elements(); e.hasMoreElements(); ){
			Termo termo = (Termo) e.nextElement();
			PalavraChave palavraChave = new PalavraChave();
			palavraChave.setPalavra(termo.Term);
			palavraChave.setFrequencia(termo.Freq);
			representacaoInterna.addPalavrasChave(palavraChave);
		}

		return representacaoInterna;
	}

	private void enviarRepresentacaoInterna(Surrogate representacaoInterna) {

		Codec codec = new SLCodec();
		Ontology ontology = SurrogateOntology.getInstance();

		myAgent.getContentManager().registerLanguage(codec);
		myAgent.getContentManager().registerOntology(ontology);

		ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM_REF);
		mensagem.addReceiver(new AID("filtrador",AID.ISLOCALNAME));
		mensagem.setLanguage(codec.getName());
		mensagem.setOntology(ontology.getName());

		try {
			myAgent.getContentManager().fillContent(mensagem,representacaoInterna);
			myAgent.send(mensagem);
		} catch (CodecException e) {
			e.printStackTrace();
		} catch (OntologyException e) {
			e.printStackTrace();
		}

		//System.out.println("=> Filtrador : " + representacaoInterna.getEndereco());
		//System.out.println("=> Filtrador : " + representacaoInterna.getTipo());
		//System.out.println("=> Filtrador : " + representacaoInterna.getNumero());
		//System.out.println("=> Filtrador : " + representacaoInterna.getData());
/*
		for(int i=1;i<10;i++){
			PalavraChave   pc     = (PalavraChave) representacaoInterna.getPalavrasChave().get(i);
			String                p      = pc.getPalavra();
			int                        f      = pc.getFrequencia();
```

```
                        System.out.println("[" + i + "] Palavra: '" + p + "' <==> Frequência: " + f);
                }
*/
        }
}
```

```
package camadaProcessamentoInformacao.agenteFiltrador;

import jade.core.Agent;


public class Filtrador extends Agent {

        protected void setup() {
                addBehaviour(new FiltrarElementosInformacao(this));
        }
}
```

```
package camadaProcessamentoInformacao.agenteFiltrador;

import jade.content.ContentElement;
import jade.content.lang.Codec;
import jade.content.lang.Codec.CodecException;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.onto.UngroundedException;
import jade.core.AID;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

import java.text.DecimalFormat;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Vector;

import recursos.RamoJuridico;
import recursos.TermoChave;
import recursos.necessidade.Necessidade;
import recursos.necessidade.NecessidadeOntology;
import recursos.satisfacao.Satisfacao;
import recursos.satisfacao.SatisfacaoOntology;
import recursos.surrogate.PalavraChave;
import recursos.surrogate.Surrogate;
import recursos.surrogate.SurrogateOntology;
```

```
public class FiltrarElementosInformacao extends CyclicBehaviour {

	private LinkedList representacoesInternas = new LinkedList();

	public FiltrarElementosInformacao(Filtrador agenteFiltrador) {
		super(agenteFiltrador);
	}

	public void action() {
		//RECEBER REPRESENTAÇÃO INTERNA DO AGENTE CONSTRUTOR
		Surrogate representacaoInterna = receberRepresentacaoInterna();

		if(representacaoInterna != null) {
			//ENVIAR DADOS DE INSTRUMENTO NORMATIVO PARA O AGENTE MODELADOR
			representacoesInternas.addLast(representacaoInterna);
			enviarDadosInstrumentoNormativo(representacaoInterna.getTipo(),representacaoInterna.getNumero());
		}

		//RECEBER NECESSIDADES POR USUÁRIO DO AGENTE MODELADOR
		Necessidade necessidade = receberNecessidadeUsuario();

		if(necessidade != null && !representacoesInternas.isEmpty()) {
			representacaoInterna = (Surrogate) representacoesInternas.getFirst();

			if(!necessidade.getIdentificador().equals(representacaoInterna.getNumero())) {
				representacoesInternas.removeFirst();
				representacaoInterna = (Surrogate) representacoesInternas.getFirst();
			}

			Iterator satisfacoes = realizarComparacao(representacaoInterna,necessidade);

			while(satisfacoes.hasNext()) {
				//ENVIAR SATISFAÇÕES POR USUÁRIO PARA O AGENTE INTERFACEADOR
				enviarSatisfacaoUsuario((Satisfacao) satisfacoes.next());
			}
		}
	}


	private Surrogate receberRepresentacaoInterna() {
		Surrogate representacaoInterna = null;

		Codec codec = new SLCodec();
		Ontology ontology = SurrogateOntology.getInstance();

		myAgent.getContentManager().registerLanguage(codec);
		myAgent.getContentManager().registerOntology(ontology);
```

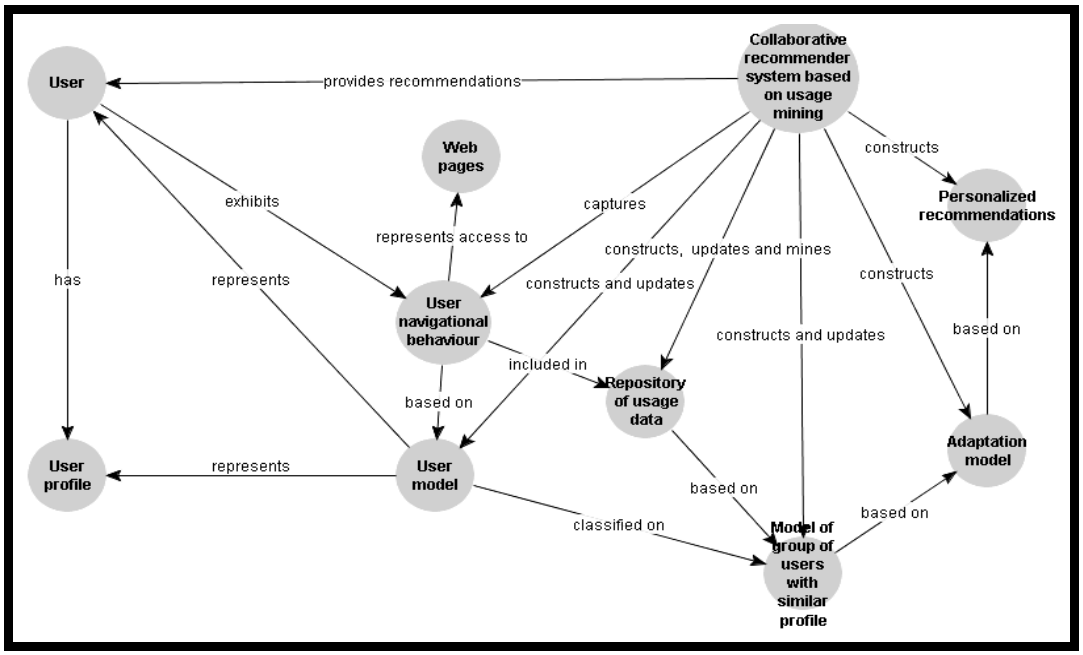
```
		MessageTemplate modeloInformacao =
MessageTemplate.and(MessageTemplate.MatchLanguage(codec.getName()),MessageTemplate.MatchOntology(ontology.getName()));
		ACLMessage mensagem = myAgent.receive(modeloInformacao);

		ContentElement conteudo = null;
		if(mensagem != null)
			try {
				conteudo = myAgent.getContentManager().extractContent(mensagem);
			} catch (UngroundedException e) {
				e.printStackTrace();
			} catch (CodecException e) {
				e.printStackTrace();
			} catch (OntologyException e) {
				e.printStackTrace();
			}

		if(conteudo instanceof Surrogate) {
			representacaoInterna = (Surrogate) conteudo;
			//System.out.println("Construtor => : " + representacaoInterna.getEndereco());
			//System.out.println("Construtor => : " + InstrumentoNormativo.obterDescricao(representacaoInterna.getTipo()));
			//System.out.println("Construtor => : " + representacaoInterna.getNumero());
			//System.out.println("Construtor => : " + representacaoInterna.getData());
			/*
			Iterator palavrasChave = representacaoInterna.getAllPalavrasChave();
			int i = 1;
			while(palavrasChave.hasNext() && i < 4) {
				PalavraChave palavraChave = (PalavraChave) palavrasChave.next();
				System.out.println("[" + i++ + "] Palavra: '" + palavraChave.getPalavra() + "' <==> Frequência: " +
palavraChave.getFrequencia());
			}
			*/
		}

		return representacaoInterna;
	}

	private void enviarDadosInstrumentoNormativo(int tipoInstrumentoNormativo, String numeroInstrumentoNormativo) {
		ACLMessage mensagem = new ACLMessage(ACLMessage.QUERY_REF);
		mensagem.setContent(new Integer(tipoInstrumentoNormativo).toString() + "_" + numeroInstrumentoNormativo);
		mensagem.addReceiver(new AID("modelador",AID.ISLOCALNAME));
		myAgent.send(mensagem);

		//System.out.println("=> Modelador : " + InstrumentoNormativo.obterDescricao(tipoInstrumentoNormativo));
	}

	private Necessidade receberNecessidadeUsuario() {
		Necessidade necessidade = null;

		Codec codec = new SLCodec();
		Ontology ontology = NecessidadeOntology.getInstance();
```

```
		myAgent.getContentManager().registerLanguage(codec);
		myAgent.getContentManager().registerOntology(ontology);

		MessageTemplate modeloInformacao =
MessageTemplate.and(MessageTemplate.MatchLanguage(codec.getName()),MessageTemplate.MatchOntology(ontology.getName()));

		ACLMessage mensagem = myAgent.receive(modeloInformacao);

		ContentElement conteudo = null;
		if(mensagem != null)
			try {
				conteudo = myAgent.getContentManager().extractContent(mensagem);
			} catch (UngroundedException e) {
				e.printStackTrace();
			} catch (CodecException e) {
				e.printStackTrace();
			} catch (OntologyException e) {
				e.printStackTrace();
			}

		if(conteudo instanceof Necessidade) {
			necessidade = (Necessidade) conteudo;
			//System.out.println("Modelador => : " + necessidade.getUsuario());
			/*
			Iterator categorias = necessidade.getAllCategorias();
			int i = 1;
			while(categorias.hasNext()) {
				Integer categoria = (Integer) categorias.next();
				System.out.println("[" + i++ + "] Categoria: " + RamoJuridico.obterDescricao(categoria.intValue()));
			}
			*/
		}

		return necessidade;
	}

	private Iterator realizarComparacao(Surrogate representacaoInterna, Necessidade necessidade) {
		Vector satisfacoes = new Vector();

		Hashtable palavrasChave = new Hashtable();
		Iterator aux = representacaoInterna.getAllPalavrasChave();
		while(aux.hasNext()) {
			PalavraChave palavraChave = (PalavraChave) aux.next();
			palavrasChave.put(palavraChave.getPalavra(),new Integer(palavraChave.getFrequencia()));
		}

		Iterator categorias = necessidade.getAllCategorias();
		while(categorias.hasNext()) {
			Integer integer = (Integer) categorias.next();
```

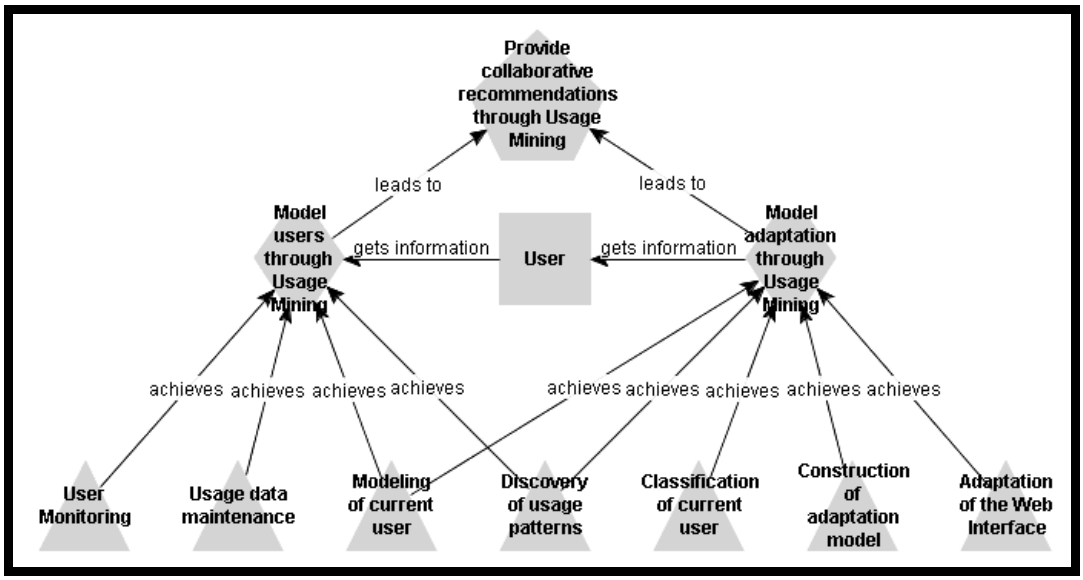
```
                int categoria = integer.intValue();
                Iterator termosChave = RamoJuridico.obterTermosChave(categoria);

                int contador = 0,
                        total   = 0;
                while(termosChave.hasNext()) {
                        TermoChave termoChave = (TermoChave) termosChave.next();
                        if(palavrasChave.get(termoChave.obterTermo()) != null)
                                contador++;
                        total++;
                }

                if(contador > 0) {
                        Satisfacao satisfacao = new Satisfacao();

                        satisfacao.setUsuario(necessidade.getUsuario());
                        satisfacao.setTipo(representacaoInterna.getTipo());
                        satisfacao.setCategoria(categoria);
                        satisfacao.setNumero(representacaoInterna.getNumero());
                        satisfacao.setData(representacaoInterna.getData());
                        satisfacao.setEndereco(representacaoInterna.getEndereco());
                        satisfacao.setSimilaridade(analisarSimilaridade(contador,total));

                        satisfacoes.add(satisfacao);
                }
        }
        System.out.println("MATCHING " + representacaoInterna.getNumero());
        return satisfacoes.iterator();
}

private String analisarSimilaridade(double contador, double total) {
        String similaridade = "0,00%";

        if(total != 0.0) {
                double percentual = 100*(contador/total);
                DecimalFormat formatadorDecimal = new DecimalFormat("###.00");
                similaridade = formatadorDecimal.format(percentual) + "%";
        }

        return similaridade;
}

private void enviarSatisfacaoUsuario(Satisfacao satisfacao) {

        Codec codec = new SLCodec();
        Ontology ontology = SatisfacaoOntology.getInstance();

        myAgent.getContentManager().registerLanguage(codec);
        myAgent.getContentManager().registerOntology(ontology);
```

```
		ACLMessage mensagem = new ACLMessage(ACLMessage.INFORM_REF);
		mensagem.addReceiver(new AID("interfaceador",AID.ISLOCALNAME));
		mensagem.setLanguage(codec.getName());
		mensagem.setOntology(ontology.getName());

		try {
			myAgent.getContentManager().fillContent(mensagem,satisfacao);
			myAgent.send(mensagem);
		} catch (CodecException e) {
			e.printStackTrace();
		} catch (OntologyException e) {
			e.printStackTrace();
		}
		/*
		String  usuario 		= satisfacao.getUsuario(),
				tipo 			= InstrumentoNormativo.obterDescricao(satisfacao.getTipo()),
				categoria 		= RamoJuridico.obterDescricao(satisfacao.getCategoria()),
				numero 			= satisfacao.getNumero(),
				data 			= satisfacao.getData(),
				endereco 		= satisfacao.getEndereco(),
				similaridade	= satisfacao.getSimilaridade();

		System.out.println("\n#=> Interfaceador ==================================================================#");
		System.out.println("Tipo: " + tipo + " Nº " + numero + ", de " + data);
		System.out.println("Categoria: " + categoria);
		System.out.println("Usuário: " + usuario);
		System.out.println("Similaridade: " + similaridade);
		System.out.println("Vínculo: " + endereco);
		System.out.println("#=========================================================================================#\n");
		*/
	}
}
```

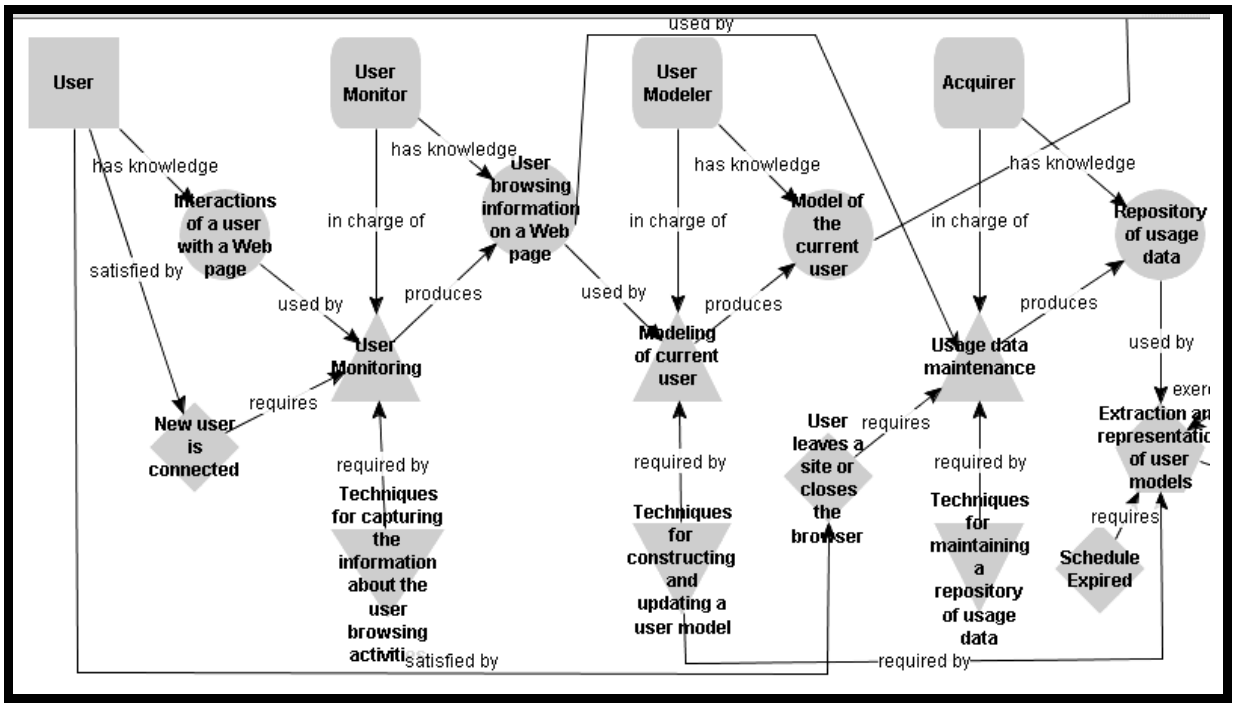
APÊNDICE A – Instâncias da modelagem da *ONTOWUM*:
modelo de domínio (*DM*) e framework multiagente (*DD*)



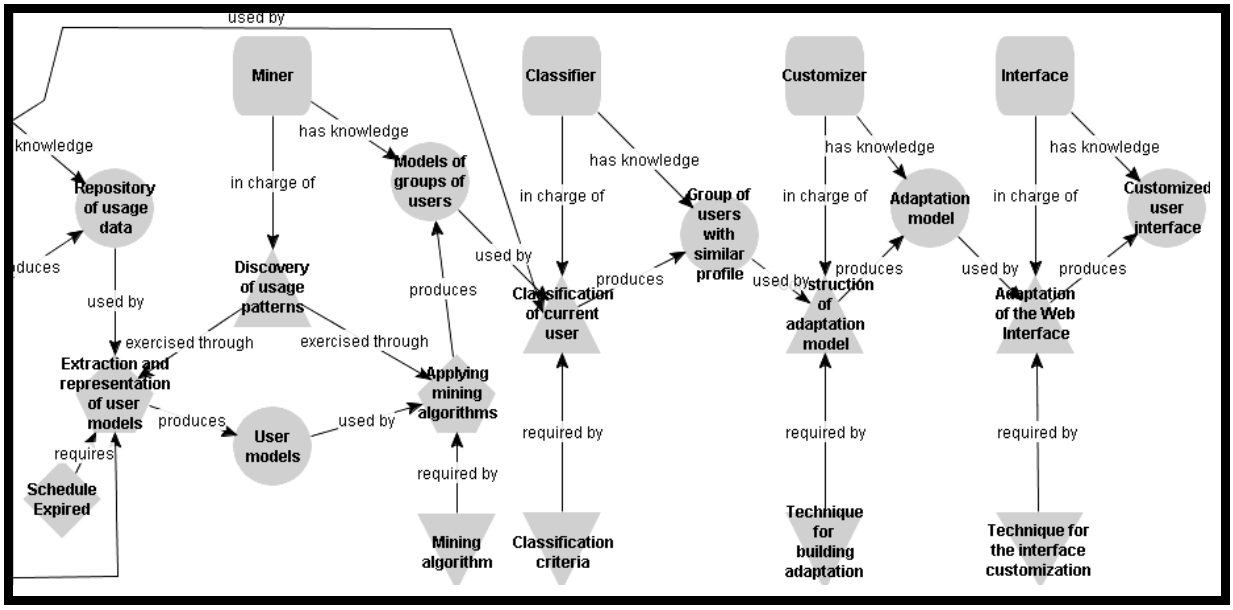
Modelo de Conceitos da ONTOWUM-DM (GIRARDI, LINDOSO, 2005d)



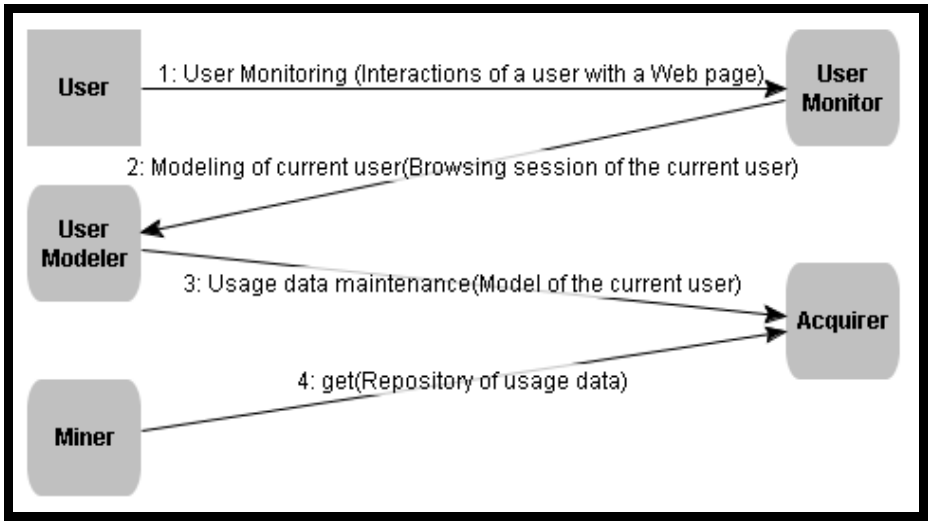
Modelo de Objetivos da ONTOWUM-DM (GIRARDI, LINDOSO, 2005d)



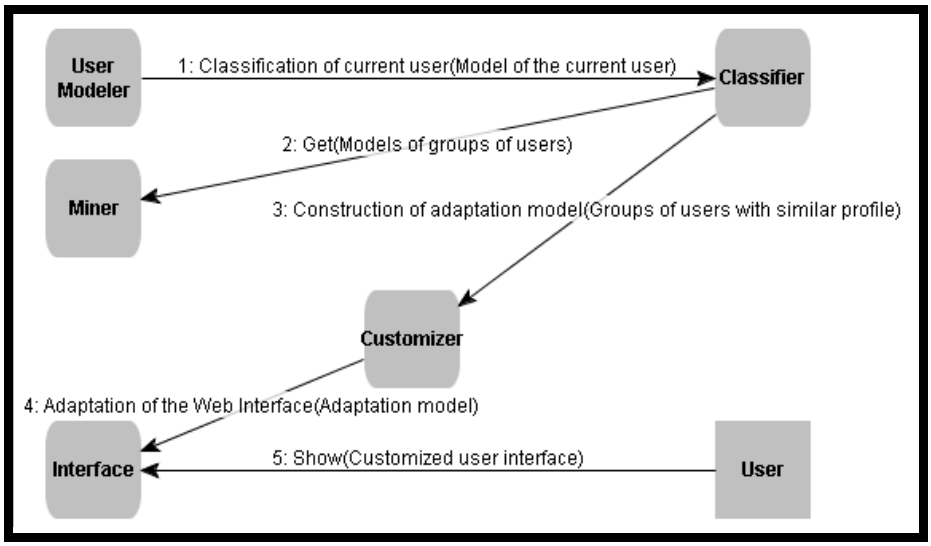
Modelo de Papéis da ONTOWUM-DM mostrando os papéis
Monitor de Usuário, *Modelador de Usuário* e *Aquisitor* (GIRARDI, LINDOSO, 2005d)



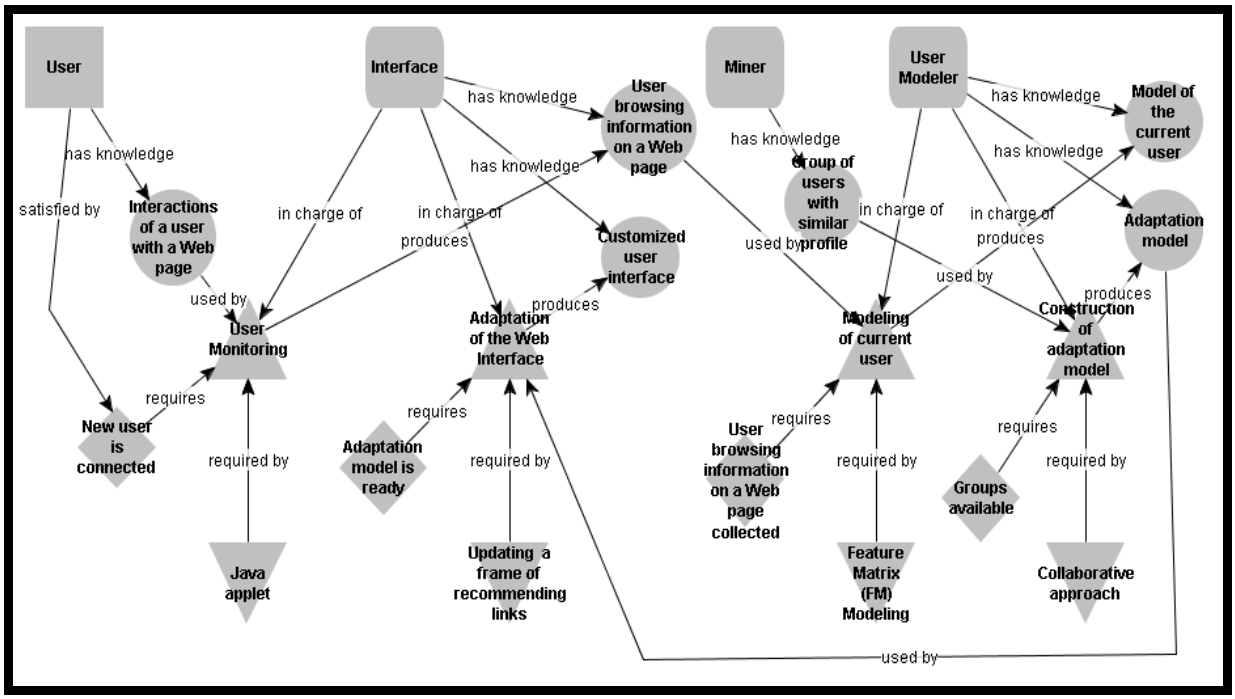
Modelo de Papéis da ONTOWUM-DM mostrando os papéis
Minerador, *Classificador*, *Personalizador* e *Interfaceador* (GIRARDI, LINDOSO, 2005d)



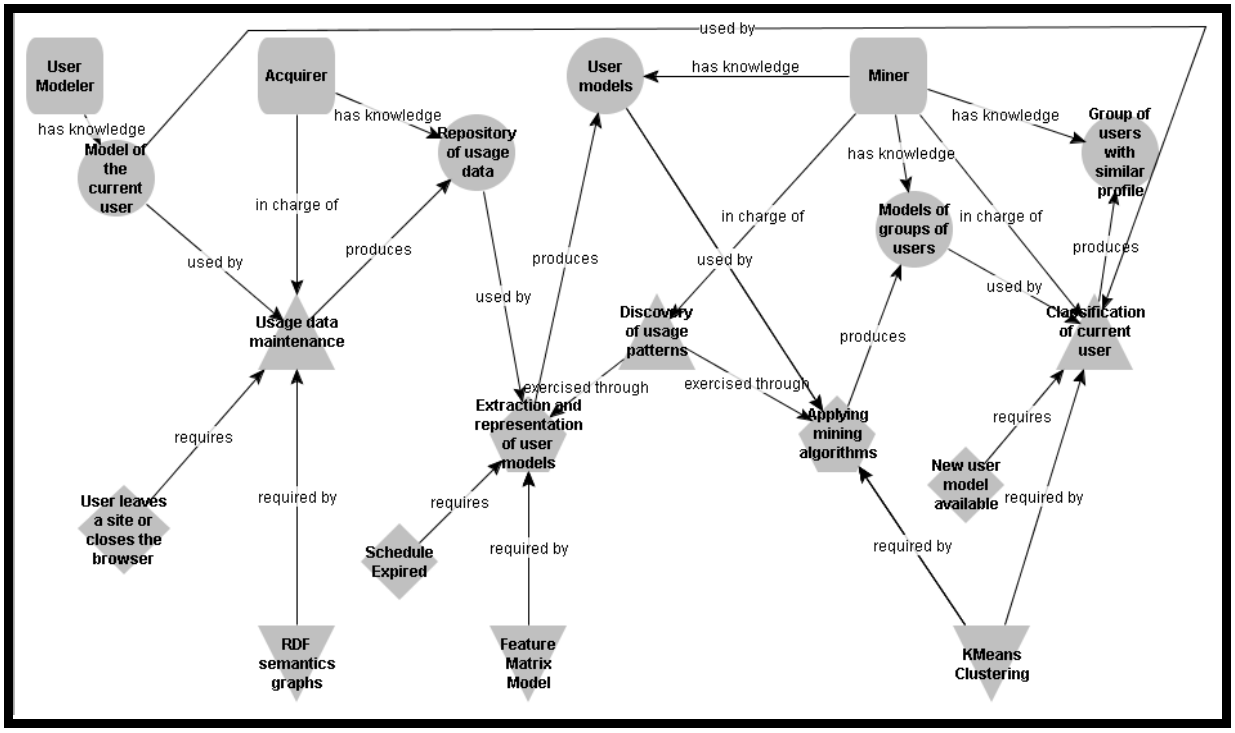
Modelo de Interações entre Papéis da ONTOWUM-DM relativo ao objetivo específico *Modelar usuários através de mineração de uso* (GIRARDI, LINDOSO, 2005d)



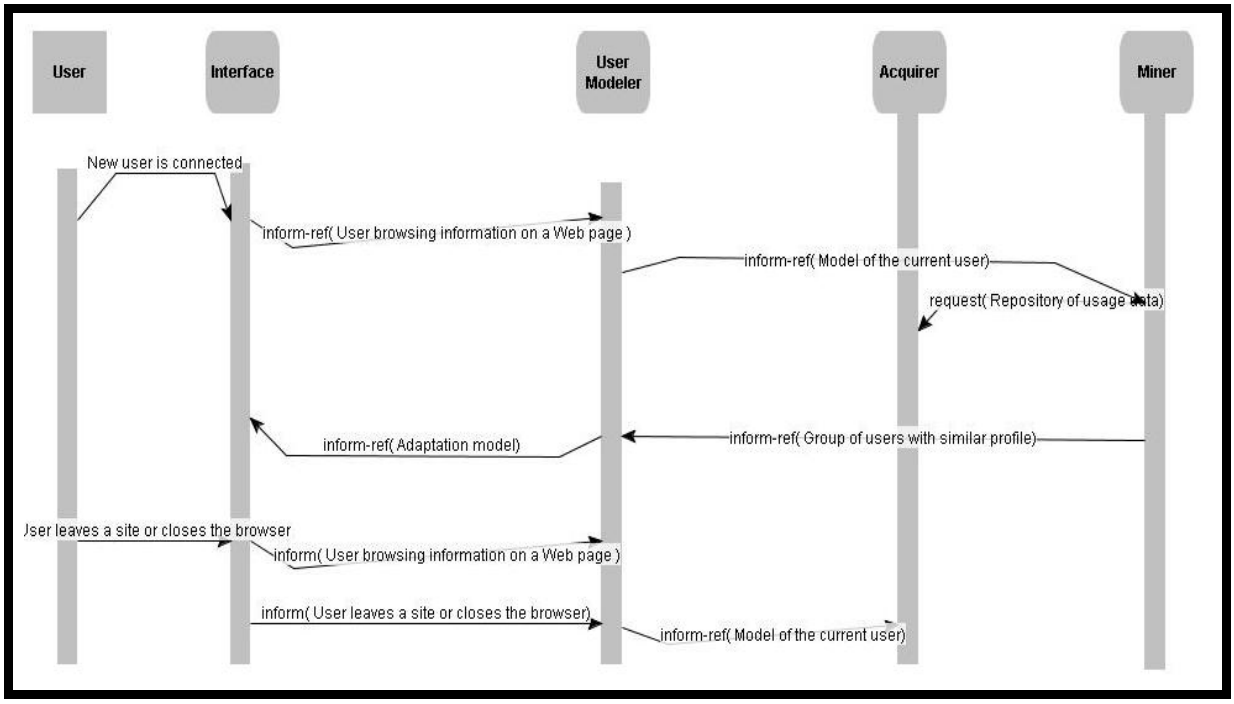
Modelo de Interações entre Papéis da ONTOWUM-DM relativo ao objetivo específico *Modelar adaptação através de mineração de uso* (GIRARDI, LINDOSO, 2005d)



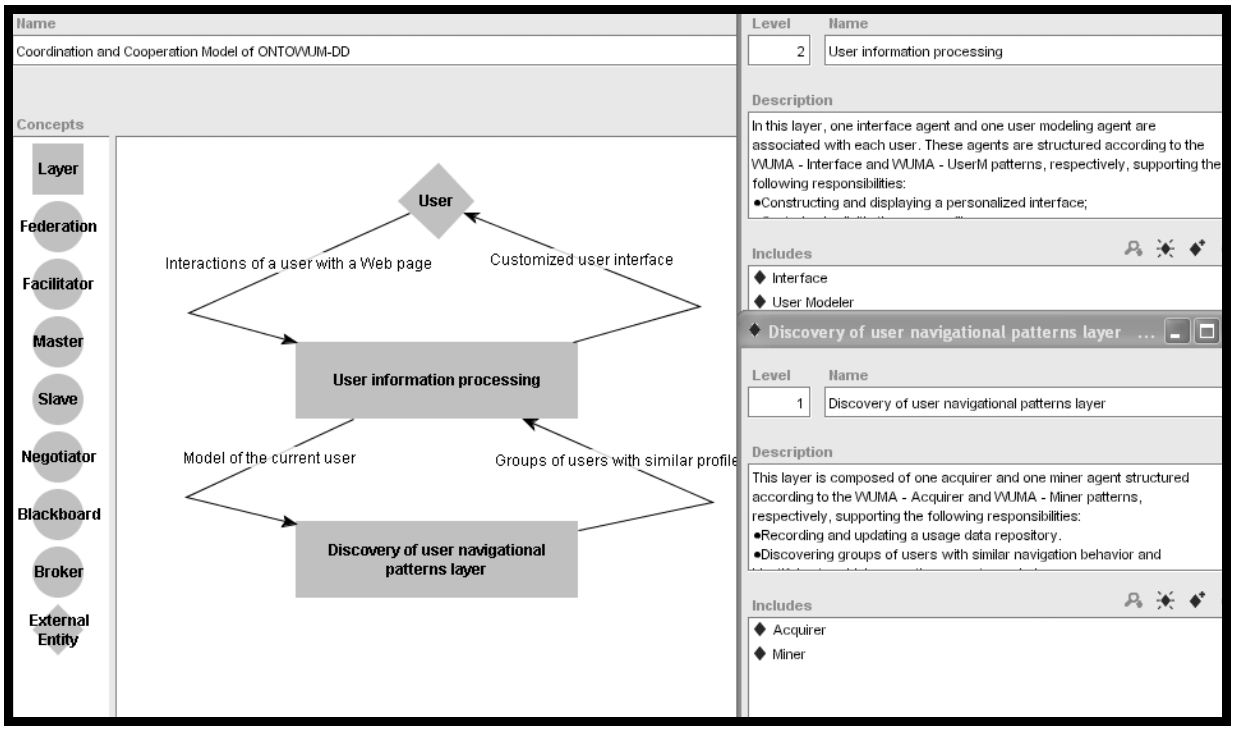
Modelo da Sociedade Multiagente da ONTOWUM-DD mostrando os agentes *Interfaceador* e *Modelador* de *Usuário* (GIRARDI, LINDOSO, 2005e)



Modelo da Sociedade Multiagente da ONTOWUM-DD mostrando os agentes *Aquisitor* e *Minerador* (GIRARDI, LINDOSO, 2005e)



Modelo de Interações entre Agentes da ONTOWUM-DD (GIRARDI, LINDOSO, 2005e)



Modelo dos Mecanismos de Cooperação e Coordenação da ONTOWUM-DD (GIRARDI, LINDOSO, 2005e)