UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA DE CIÊNCIA DA COMPUTAÇÃO

UIRATAN ALVES DE SOUSA CAVALCANTE

MADAE-IDE: Um ambiente de desenvolvimento de software baseado no conhecimento para o reuso composicional no desenvolvimento de sistemas multiagente

São Luís
2009

UIRATAN ALVES DE SOUSA CAVALCANTE

MADAE-IDE: Um ambiente de desenvolvimento de software baseado no conhecimento para o reuso composicional no desenvolvimento de sistemas multiagente

Dissertação de Mestrado apresentada ao Curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão, para obtenção do título de Mestre em Engenharia de Eletricidade, na área de Ciência da Computação.

Orientadora: Profª Drª Rosario Girardi

São Luís

2009

Cavalcante, Uiratan Alves de Sousa
MADAE-IDE: Um ambiente de desenvolvimento de software baseado no conhecimento para o reuso composicional no desenvolvimento de sistemas multiagente / Uiratan Alves de Sousa Cavalcante. – São Luís, 2009.
176f.
Orientadora: Rosário Girardi.
Impresso por computador (Fotocópia).
Dissertação (Mestrado) – Universidade Federal do Maranhão, Curso de Engenharia de Eletricidade na área de Ciência da Computação. São Luís, 2009.

1. Software - Desenvolvimento. 2. Sistemas Multiagente. 3. Ontologias. 4. Sistemas baseados em conhecimento. 5. Software - Engenharia. I Título.

CDU 004.4'2

MADAE-IDE: UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE BASEADO NO CONHECIMENTO PARA O REUSO COMPOSICIONAL NO DESENVOLVIMENTO DE SISTEMAS MULTIAGENTE

Uiratan Alves de Sousa Cavalcante

Dissertação aprovada em 01 de dezembro de 2009.

Profa. Maria del Rosário Girardi, Ph. D.
(Orientadora)

Prof. Giancarlo Guizzardi, Ph. D.
(Membro da Banca Examinadora)

Prof. Zair Abdelouahab, Ph. D.
(Membro da Banca Examinadora)

Ao meu filho João Manoel.

À minha mãe Maria de Lourdes e meu irmão Luciano Klaus.

AGRADECIMENTOS

Aos meus pais, em especial minha mãe Lourdes por todo o auxílio e exemplo que sempre foi para mim.

À Profª Girardi pelas orientações fundamentais e toda a paciência despedida no decorrer do mestrado e principalmente no desenvolvimento desta dissertação.

Aos colegas e ex-colegas do grupo GESEC, em especial à Adriana por suas revisões e sugestões.

Ao Programa de Pós-Graduação em Engenharia de Eletricidade pela oportunidade de realizar este trabalho e ao Alcides, que sempre prestou um trabalho cordial na secretaria.

A todos que acreditaram na conclusão deste trabalho e que, de alguma forma, foram importantes na sua conclusão.

“If I have seen a little further it is by standing on the shoulders of giants.”

Sir Isaac Newton

RESUMO

O paradigma de desenvolvimento orientado a agentes tem atingido um alto nível de maturidade na última década. Entretanto, ainda faltam ambientes integrados de desenvolvimento de software que suportem todas as fases do desenvolvimento orientado a agentes. MADAE-Pro é um processo para o desenvolvimento e reuso de famílias de sistemas multiagente e integra dois subprocessos complementares. Um é baseado nos conceitos da Engenharia de Domínio, isto é, visa construir artefatos reutilizáveis que representem uma família de aplicações e o outro, baseado na Engenharia de Aplicações, guia o desenvolvimento de uma aplicação específica reutilizando os produtos do primeiro subprocesso. Este trabalho propõe MADAE-IDE, um ambiente integrado de desenvolvimento baseado no conhecimento que automatiza o processo desenvolvimento de software multiagente MADAE-Pro. O ambiente agrega a ontologia ONTORMAS e, através de regras de inferência e buscas semânticas diretamente sobre suas instâncias, automatiza as tarefas de modelagem do processo MADAE-Pro. Estas características levam a um ganho de produtividade por parte do desenvolvedor, além de manter uma base de conhecimento consistente, garantido o sucesso do reuso em futuras aplicações.

Palavras-chave: Ambiente de desenvolvimento de software. Sistemas multiagente. Ontologias. Sistemas baseados em conhecimento. Engenharia de software. Engenharia de conhecimento.

ABSTRACT

The agent-oriented development paradigm has achieved a high maturity level over the last decade. However, integrated development environments supporting all phases of agent-oriented development are still missing. MADAE-Pro is a process for the development and reuse of family of multi-agent systems and integrates two complementary process. One is based on Domain Engineering concepts, aiming in creating artifacts of reusable software in the development of an application family in a particular domain problem, and the other is based on Application Engineering, which guides the construction of applications based on reusable software artifacts previously produced in the Domain Engineering process. The environment includes the ONTORMAS ontology and, through inference rules and semantic search over its instances, automates the modeling tasks of the MADAE-Pro process. These features allow for the developer productivity gains and for maintaining the consistence of the knowledge-base, ensuring the success of the reuse in future applications.

Keywords: Software development environments. Multi-Agent systems. Ontologies. Knowledge-based systems. Software engineering. Knowledge engineering.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACL	Agent Communication Language
ADEMAS	Aplication Design technique for Multi-Agent Systems
ADS	Ambientes de Desenvolvimento de Software
AIMAS	Application Implementation Technique for Multi-Agent Systems
CASE	Computer-Aided Software Engineering
DDEMAS	Domain Design technique for Multi-Agent Systems
DIMAS	Domain Implementation technique for Multi-Agent Systems
FIPA	Foundation for Intelligent Physical Agents
GESEC	Grupo de pesquisa em Engenharia de Software e Engenharia do Conhecimento
GRAMO	Generic Requiremente Analysis Method based on Ontologies
HL7	Health Level Seven
JADE	Java Agent Development Framework
JESS	Java Expert System Shell
MAAEM	Multi-agent Aplication Engineering Methodology
MADAE-IDE	Multi-agent Domain and Application Engineering – Integrated Development Environment
MADAE-ML	Multi-agent Domain and Aplication Engineering – Modeling Language
MADAE-Pro	Multiagent Domain and Application Engineering Process
MADEM	Multi-agent Domain Engineering Methodology
ODE	Ontology-based software Development Environment
OME	Organization Modeling Environment
ONTOMADEM	Ontology for Multi-Agent Domain Engineering Methodology
ONTORMAS	Ontology for Reusing Multiagent Software
ONTOWUN-DM	Ontolology-driven Model of Recommender Systems

	based on Web Usage Mining
OO	Orientado a Objetos
OSEE	Ontology-driven Software Engineering Environment
OWL	Web Ontology Language
PSEE	Process-centered Software Engineering Environment
RDQL	RDF Data Query Language
SBC	Sistemas Baseados em Conhecimento
SE	Sistemas Especialistas
SMA	Sistema Multiagente
SRAMO	Specific Requirement Analysis Method based on Ontologies
URI	Unique Resource Identifier
W3C	World Wide Web Consortium

SUMÁRIO

1 INTRODUÇÃO

Com o surgimento do paradigma orientado a agentes, foram e ainda estão sendo propostas diversas metodologias e processos para guiar o desenvolvimento de software seguindo esse paradigma. Para que possamos tirar máximo proveito dessas novas tecnologias, a utilização de ferramentas que auxiliem o desenvolvedor torna-se indispensável.

O surgimento da tecnologia CASE (Computer Aided Software Engineering) - Engenharia de Software Auxiliada por Computador exerceu um enorme impacto sobre a área. As ferramentas CASE proporcionam suporte às metodologias e métodos de desenvolvimento de software. Os ambientes integrados de desenvolvimento de software, ou simplesmente Ambientes de Desenvolvimento de Software (ADSs) representam uma evolução do conceito de CASE, definindo mecanismos de integração entre as ferramentas e oferecendo suporte a todas as etapas do ciclo de vida (FALBO, 1998).

Técnicas e ferramentas são integradas para permitir o controle de cada uma das etapas que são necessárias para desenvolver um produto de software. Assim, os ambientes de desenvolvimento de software orientados ao processo (do inglês Process-centered Software Engineering Environment - PSEE) (GIMENES, 1994) (REIS, 2000) surgiram com o objetivo de proporcionar uma estrutura computacional que gerencie o intercâmbio de informações entre os desenvolvedores, controlando as atividades realizadas, envolvendo assim os recursos consumidos, os prazos determinados, e as datas de início e término de cada atividade.

Assim, faz-se necessário o desenvolvimento de ferramentas de software especialmente projetadas para este fim e que possam agregar melhorias na qualidade de desenvolvimento, como o reuso, no qual se obtenha aproveitamento máximo (redução de custos, tempo de desenvolvimento, minimização de erros) através da reutilização de artefatos de software anteriormente criados.

A criação de uma ferramenta destinada ao desenvolvimento de artefatos e de aplicações baseado no reuso é de extrema relevância, pois incorpora o reuso de forma sistemática no processo de desenvolvimento de sistemas multiagente. A produtividade e qualidade são dois fatores desejáveis no desenvolvimento de um

produto de software e uma das formas de obtê-los é através do reuso. O reuso de software é um processo de criação de sistemas de software a partir de software existente, ao invés de desenvolvê-los a partir do zero (KRUEGER, 1992).

A Engenharia de Domínio é um processo para criação de abstrações de software reusáveis no desenvolvimento de uma família de aplicações em um domínio particular de problema. A Engenharia de Aplicações é um processo para construção de aplicações baseadas no reuso de artefatos de software previamente produzidos no processo da Engenharia de Domínio.

Ontologias provêem uma terminologia não-ambígua que pode ser compartilhada por todos os envolvidos no processo de desenvolvimento de software. Elas podem ser tão genéricas quanto necessário permitindo o seu reuso e fácil extensão. Estas características tornam a ontologias úteis para representar o conhecimento das técnicas e metodologias para a Engenharia de Software e é um mecanismo de abstração apropriado para especificação de artefatos de software de alto nível como modelos de domínio, *frameworks* e padrões de software (GIRARDI et al., 2004).

O MADAE-Pro (LEITE, 2009), (LEITE e GIRARDI, 2009) é um processo para o desenvolvimento e reuso de famílias de sistemas multiagente. Ele integra dois subprocessos complementares. Um é baseado nos conceitos da Engenharia de Domínio, isto é, visa construir artefatos reutilizáveis que representem uma família de aplicações e o outro, baseado na Engenharia de Aplicações, guia o desenvolvimento de uma aplicação específica reutilizando os produtos do primeiro subprocesso. Os conceitos utilizados pelo MADAE-Pro são expressos na ontologia ONTORMAS (LEITE et al., 2008), parte da ferramenta ONTORMAS-T para modelagem de famílias de aplicações e aplicações específicas seguindo as diretrizes do processo MADAE-Pro, e os seus produtos são representados como instâncias desta. Isso permite incorporar ao MADAE-Pro as vantagens fornecidas pelas ontologias como a estruturação do conhecimento, compartilhamento de conhecimento e facilidade de reuso.

O processo MADAE-Pro é complexo e exige um grande esforço e tempo do desenvolvedor para a correta instanciação das fases e produtos de modelagem na ONTORMAS-T. Além disso, para se efetivar o reuso é necessário recuperar, adaptar e integrar instâncias adequadamente. Estes fatores provocam o surgimento de erros de modelagem, que podem tornar a base de conhecimento inconsistente.

As tarefas de modelagem do MADAE-Pro têm sido executadas manualmente apenas apoiadas pela linguagem gráfica e o editor de modelos da ontologia, o que se mostrou trabalhoso e suscetível a erros tornando a base de conhecimento ONTORMAS geralmente inconsistente.

1.1 Objetivos

O objetivo geral desse trabalho é contribuir com a qualidade e produtividade do desenvolvimento de software através da construção de ferramentas baseadas no conhecimento que promovam o reuso de software.

Para alcançar esse objetivo geral, planeja-se atingir o objetivo específico de modelar e desenvolver um ambiente integrado de desenvolvimento de software baseado no conhecimento que auxilie o desenvolvedor na aplicação do processo MADAE-Pro.

1.2 Estruturação

Este trabalho, incluindo esta introdução, está estruturado em seis capítulos. No capítulo 2 é apresentada uma visão geral da Engenharia de Conhecimento aplicada à Engenharia de Software, o processo de desenvolvimento baseado no reuso de software e das ontologias. Ferramentas baseadas em conhecimento são discutidas e é feito um comparativo entre elas.

No capítulo 3 é apresentado o processo MADAE-Pro, para o desenvolvimento de famílias de sistemas e de aplicações multiagente.

O ambiente MADAE-IDE, tema central deste trabalho, é apresentado no capítulo 4, detalhando-se o projeto do ambiente e sua implementação.

Um estudo de caso para avaliação do ambiente é desenvolvido no capítulo 5, através da comparação de uma família de aplicações já desenvolvida manualmente com outra modelada através do ambiente MADAE-IDE.

No último capítulo, as conclusões do trabalho são apresentadas, incluindo as contribuições do ambiente MADAE-IDE, as limitações que o mesmo ainda possui e que precisam ser superadas e as perspectivas de trabalhos que estão sendo ou que serão desenvolvidos no futuro.

2 ENGENHARIA DO CONHECIMENTO E ENGENHARIA DE SOFTWARE

As metodologias para o desenvolvimento de Sistemas Baseados em Conhecimento (SBC) prescrevem o desenvolvimento de abstrações para solução de problemas e modelos de domínio. Abstrações, ou modelos, são necessárias para permitir a seleção e configuração de componentes reutilizáveis para uma aplicação específica. As tecnologias de modelagem e representação de conhecimento, em suas várias formas (por exemplo, as ontologias), estão tornando-se onipresente.

Este capítulo apresenta uma introdução à Engenharia de Conhecimento, seguido de sua utilização na Engenharia de Software através de sistemas baseados em conhecimento. Ao final são apresentadas as considerações finais desta abordagem.

2.1 Introdução

No início das pesquisas sobre Inteligência Artificial, a idéia principal era a transferência do conhecimento de um especialista em um sistema através da mineração do conhecimento deste especialista. Esta visão assume que “gemas de especialização” podem ser elicitadas uma a uma do domínio do perito e codificado em um sistema, onde o objetivo é criar um perito virtual, ou seja, um sistema que possa emular o comportamento do perito na solução do problema, baseando-se no mesmo corpo de conhecimento (MOTTA, 2001).

Os Sistemas Baseados em Conhecimento (SBC) são capazes de resolver problemas usando conhecimento específico sobre o domínio da aplicação, enquanto os Sistemas Especialistas (SE), concebidos para reproduzir o comportamento de especialistas humanos na resolução de problemas do mundo real, mas possuindo um domínio de problemas restritos (BITTENCOURT, 1998), são SBC que resolvem problemas ordinariamente resolvidos por um especialista humano (REZENDE et al., 2003).

A Engenharia de Software fornece tecnologias e paradigmas de desenvolvimento que possibilitam, de um lado, a produção de componentes reutilizáveis com potencial e, de outro, a construção de sistemas a partir da

integração de forma racional desses componentes. A chave do sucesso dessa prática na engenharia de desenvolvimento de sistemas a partir de partes previamente construídas, adaptáveis e provadas corretas está na habilidade de construir componentes baseados em abstrações simples, fáceis de serem encontrados, passíveis de uso em várias aplicações com estrutura e comportamento encapsulados.

2.2 Engenharia do Conhecimento

O processo de desenvolvimento de um Sistema Especialista é chamado de Engenharia do Conhecimento. Este processo consiste na tarefa do engenheiro do conhecimento extrair de um ou mais especialistas a metodologia de solução de determinado problema e transferir este conhecimento para o sistema. O resultado é um programa de computador capaz de resolver problemas como especialistas humanos. (BITTENCOURT, 1998).

Os SEs são utilizados quando o problema a ser resolvido não se molda a um algoritmo padrão sendo necessário aplicar conhecimento ao invés de um algoritmo. Desta forma, os SEs são Sistemas Baseados em Conhecimento (SBC). A relação entre sistemas baseados em conhecimento e sistemas especialistas é apresentada na Figura 1 (WATERMAN, 1986).

Um SBC apresenta, em geral, uma arquitetura com três módulos: uma base de regras, uma memória de trabalho e um motor de inferência. A base de regras e memória de trabalho formam a base de conhecimento do SBC, onde está representado o conhecimento sobre o domínio. O motor de inferência é o mecanismo de controle do sistema que avalia e aplica as regras de acordo com as informações da memória de trabalho. O conhecimento deve ser obtido junto a um especialista humano do domínio e representado de acordo com regras formais definidas para a codificação de regras no SBC em questão.

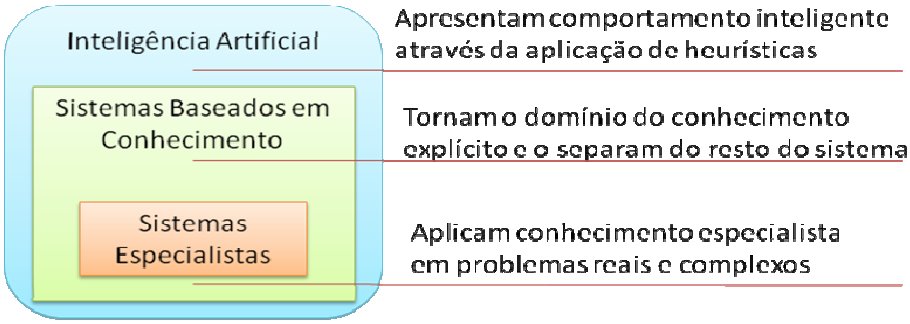

Figura 1 - Sistemas especialistas e sistemas baseados em conhecimento

A modelagem de conhecimento visa representar sistemas num nível de abstração sem considerações de implementação, modelando o que os sistemas conhecem e como eles utilizam este conhecimento.

Como o conhecimento existe em todo lugar (agentes de software, pessoas, dispositivos), são necessários métodos e técnicas para sua aquisição, representação, compartilhamento e manutenção, independentemente de onde esteja: em um banco de dados, numa pessoa ou em um robô.

As fontes potenciais de conhecimento são os especialistas humanos (principal fonte), livros-texto, bancos de dados, documentos com relatos de experiências e estudos, a experiência pessoal do engenheiro do conhecimento. Quando a fonte de conhecimento é uma pessoa, a atividade é essencialmente um empreendimento social que requer cooperação entre o elicitante (geralmente o engenheiro de conhecimento) e o provedor de conhecimento. Ambas as partes interpretam a situação como sua progressão e ajustam suas respostas para tornarem-se apropriadas.

O foco da Engenharia de Conhecimento está na modelagem de sistemas e não na tentativa de reproduzir a maneira como os especialistas raciocinam, defendendo a visão de que uma base de conhecimento deve ser vista como um produto de uma atividade de modelagem e não um repositório de conhecimento especializado. A modelagem passa a ser o aspecto central da Engenharia de Conhecimento e a aquisição de conhecimento passa a ser essencialmente um processo construtivo, no qual o engenheiro de conhecimento usa todos os tipos de informação disponíveis e estabelece as decisões finais de modelagem (CLANCEY, 1993).

A aquisição de conhecimento é a transferência e transformação da habilidade ou perícia para resolver problemas contidos em alguma fonte de

conhecimento para um programa. A aquisição do conhecimento não deve ser caracterizada como um processo de mapeamento entre o conhecimento do especialista em uma representação computacional, mas sim como um processo de construção de modelos (MOTTA, 2001). A modelagem do conhecimento permite criar modelos para reutilização, sendo utilizados pela engenharia baseada no reuso, quando da criação de um sistema específico da família de aplicação (GRUBER, 1995).

Para resolver os problemas mais complexos encontrados na Inteligência Artificial, é necessário uma grande quantidade de conhecimentos e certos mecanismos para manipulá-los. Para implementar estes problemas é necessário contar com uma forma adequada de representação do conhecimento. Quando discutimos representação do conhecimento lidamos com dois tipos diferentes de entidades: os fatos, verdades em algum mundo relevante, as coisas que queremos representar e a representação destes fatos com algum formalismo escolhido, são as coisas que efetivamente seremos capazes de manipular.

Uma ontologia (CHANDRASEKARAN, 1999) (FALBO R., 1998) (GÓMEZ PÉREZ, 2004) (GUARINO, 1998) é uma especificação parcial de um vocabulário sobre um domínio de discurso (GUARINO, 1998). O termo ontologia tem sido usado como: (i) artefato concreto de engenharia, projetado com um propósito específico e sem prestar nenhuma ou quase nenhuma atenção a aspectos teóricos de fundamentação; (ii) modelo de um domínio específico do conhecimento (ex., biologia molecular, finanças, logística, doenças infecciosas) expresso em uma linguagem de representação do conhecimento (ex., RDF, OWL, F-Logic) ou modelagem conceitual (ex., UML, EER, ORM) (GUIZZARDI et al., 2009).

As áreas de compartilhamento e reuso na modelagem de conhecimento podem utilizar alguma forma de abstração para os artefatos de software (GRUBER, 1995). Abstração é a principal característica em qualquer técnica de reuso. Sem abstrações, os desenvolvedores de software deixariam de lado o uso de coleções de artefatos reutilizáveis em troca da tentativa de descobrir o que faz cada artefato, quando ele pode ser reutilizado e quando usá-lo. As abstrações no caso de componentes para SBC são os modelos de nível de conhecimento. Assim, o reuso neste paradigma implica em modelagem: não podemos descobrir o que um artefato faz, a menos que tenhamos um modelo de suas competências (MOTTA, 2001).

2.3 Paradigma de Desenvolvimento Orientado a Agentes

Existe uma demanda por sistemas que interajam de maneira mais sofisticada com o usuário, que sejam capazes de executar suas funções eficientemente mesmo em ambientes dinâmicos, que possam perceber e adequar-se às mudanças no ambiente, e que saibam lidar com adversidades.

Os Sistemas Multiagente (WOOLDRIDGE e JENNINGS, 1999) (RUSSEL e NORVIG, 2003) são sistemas compostos por múltiplos agentes, que interagem e cooperam para realizar atividades que estariam fora da sua capacidade individual ou em alguns casos, como nos jogos, competem entre si para atingir um objetivo.

2.3.1 Agentes

Agente é definido em (WOOLDRIDGE, 1997) como “um sistema de computação que é capaz de agir independentemente em nome do seu usuário ou do seu proprietário”. Um agente pode descobrir por ele mesmo o que ele precisa fazer para satisfazer seus objetivos de projeto em vez de ter que ser dito explicitamente a ele o que fazer a qualquer momento dado”. (RUSSEL e NORVIG, 2003) definem agentes como “tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e agir sobre ele por intermédio de atuadores”. Essa é uma definição bem aceita na comunidade de pesquisa e desenvolvimento de sistemas multiagente. Com ela, o trabalho de construir um agente se resume a definir o mapeamento entre o que é percebido do ambiente (entradas) e as ações que o agente efetua no ambiente (saída).

Assim, um agente é um sistema de computador encapsulado que está situado em algum ambiente e que é capaz de ser flexível e de ter ações autônomas que permitem atingir os seus objetivos (BRENNER et. al., 1998). Na Tabela 1 encontram-se algumas dessas características principais (WOOLDRIDGE, 1997).

De modo diferente de objetos, que são definidos em termos de atributos e métodos, um agente é definido pelos seus comportamentos, ou seja, como ele percebe o ambiente e age sobre o referido ambiente.

Tabela 1 - Características de Agentes

Característica	Descrição
Autonomia	Um agente pode agir de modo autônomo, sem que outro agente ou o usuário solicite deliberadamente que ele o faça.
Pró-atividade	Um agente não só reage a estímulos do ambiente, mas também age pró-ativamente visando cumprir os objetivos do sistema.
Habilidade Social ou Comunicação	Um agente possui a capacidade de se comunicar com outros agentes, trocando informações, solicitando a execução de tarefas, retornando resultados das execuções, entre outras.
Reatividade	Um agente é capaz de receber estímulos do seu ambiente via seus sensores e decidir que ações tomarem com base nessa informação.
Adaptabilidade	Um agente pode mudar o seu comportamento com base em experiências anteriores.

2.3.2 Sistemas Multiagente

Basicamente, um Sistema Multiagente (SMA) é um sistema composto por vários agentes que se comunicam e são coletivamente capazes de atingir objetivos que não seriam capazes de satisfazer separadamente.

Em tais sistemas, a complexidade deve emergir das interações entre os agentes. Ou seja, cada agente pode executar, dentro de suas limitações, uma tarefa simples, mas a boa coordenação da execução dessas tarefas simples por cada agente torna o sistema capaz de executar tarefas de grande complexidade.

Uma característica de um sistema multiagente bem modelado é justamente esta: a de que cada agente execute uma tarefa simples para que o sistema como um todo seja capaz de tratar tarefas complexas.

Para que se atinja tal objetivo, é necessário que se definam bons mecanismos de cooperação (auxílio) e coordenação (organização) no sistema multiagente (GIRARDI, 2004).

A FIPA (Foundation for Intelligent Physical Agents) (FIPA, 2008) direciona esforços para a produção de especificações para a tecnologia de agentes. Um conjunto de especificações FIPA muito importante é o da Linguagem de Comunicação entre Agentes (ACL). Esse conjunto inclui as especificações de uma série de protocolos de interação entre os agentes (como o Contract-Net, Query, Request, entre outros). As especificações FIPA ACL incluem a definição do formato das mensagens trocadas entre os agentes.

O Java Agent DEvelopment framwork (JADE, 2008) é um ambiente open source para desenvolvimento de aplicações baseado em agentes, conforme as especificações da FIPA. O objetivo de JADE é garantir um padrão de integração de SMA através de um conjunto de agentes e de serviços de sistema.

JADE oferece facilidades para o desenvolvimento de sistemas multiagente que dispensam o desenvolvedor de se preocupar com a implementação de uma plataforma eficiente de agentes, ou com a comunicação, troca de mensagens e muitos outros atributos que um sistema multiagente necessita. Além disso, JADE oferece uma variedade de ferramentas de monitoração, gerenciamento e depuração que ajudam tanto no desenvolvimento quanto na manutenção e suporte a sistemas multiagente. Vale ressaltar a grande preocupação que JADE tem em sempre manter os padrões especificados pela FIPA. Fato que aumenta o grau de interoperabilidade e escalabilidade do ambiente em relação a outros SMA.

Jess (FRIEDMAN-HILL et al., 2003) é um motor de inferência e um ambiente de scripting totalmente escrito em linguagem Java. Usando Jess, pode-se construir software em Java que tenha a capacidade de raciocinar utilizando algum conhecimento que possamos fornecer, sob a forma de regras declarativas. Jess é pequeno, leve, e um dos mais rápidos motores de inferência disponíveis. A sua linguagem de scripting dá acesso a todas as APIs do Java's. Jess ainda possui um plugin para a plataforma Eclipse.

2.4 Reuso de software baseado no conhecimento

Nesta seção são definidos os principais conceitos utilizados neste trabalho para o reuso de software baseado no conhecimento: Engenharia de Domínio, Engenharia de Aplicações e ontologias.

2.4.1 A Engenharia de Domínio

A Engenharia de Domínio, também chamada de desenvolvimento *para* o reuso, é uma abordagem sistemática para a construção de abstrações de software reutilizáveis (ARANGO, 1988). Desenvolver software *para* o reuso significa que a intenção do desenvolvedor não é a construção de uma aplicação específica, mas

desenvolver um conjunto de artefatos que são comuns a uma família de aplicações. A Engenharia de Domínio inclui três fases básicas: Análise de Domínio, Projeto de Domínio e Implementação de Domínio. Na Análise de Domínio, são definidos os requisitos comuns e variáveis pertencentes à família de aplicações. Esses requisitos são representados em um modelo de domínio. Um modelo de domínio é uma representação explícita das propriedades comuns e variáveis do sistema em um domínio e das dependências entre as propriedades variáveis (CZARNECKI, 1998).

No Projeto de Domínio, a arquitetura comum é estabelecida para a família de aplicações de acordo com o modelo de domínio. O projeto possui duas partes principais, o projeto global e detalhado. No projeto global é definida a arquitetura global da família de aplicações, isto é, como os componentes estão estruturados. No projeto detalhado, é definida a estrutura de cada componente da arquitetura global. Nessa fase, também é feita a especificação das características comuns e variáveis da família de aplicações de cada componente da arquitetura. O produto dessa fase é um modelo arquitetural contendo a arquitetura global e detalhada bem como a especificação de características comuns e variáveis da família de aplicações. Esse modelo é usado para guiar a implementação do domínio.

Na Implementação de Domínio os artefatos são implementados de acordo com o projeto global e detalhado e com a especificação de variabilidades definidos no Projeto de Domínio. Esses artefatos podem ser, por exemplo, componentes reusáveis e linguagens especificas de domínio.

2.4.2 A Engenharia de Aplicações

A Engenharia de Aplicações, também chamada de desenvolvimento *com* reuso, é o processo de desenvolvimento de aplicações de software individuais a partir de artefatos de software criados pelo processo da Engenharia de Domínio. Na falta de artefatos para serem reusados estes podem ser desenvolvidos especificamente para a aplicação. A Engenharia de Aplicações possui três fases básicas: Engenharia dos Requisitos da Aplicação, Projeto da Aplicação e Implementação da Aplicação. Na Engenharia dos Requisitos da Aplicação, os requisitos de uma aplicação específica da família são definidos através da seleção

dos requisitos comuns e variáveis a uma família de aplicações definidos no modelo de domínio da fase de Análise de Domínio.

No Projeto da Aplicação, a arquitetura da família de aplicações produzida na fase de Projeto de Domínio é reusada e são realizadas as adaptações necessárias para a aplicação que se estiver desenvolvendo.

Na Implementação da Aplicação, os componentes de software produzidos na fase de Implementação do Domínio são reusados para compor a aplicação de acordo com as características comuns e variáveis. Os componentes reusados podem precisar sofrer algum tipo de adaptação para atender algum requisito especifico da aplicação que estiver sendo desenvolvida. Depois da seleção e adaptação os componentes são integrados para compor a aplicação.

2.4.3 Ontologias

Uma ontologia é freqüentemente definida como a especificação de uma conceitualização (GRUBER, 1995). A conceitualização refere-se à abstração de uma parte do mundo (um domínio) onde são representados os conceitos relevantes e seus relacionamentos.

As ontologias provêem uma terminologia não-ambígua que pode ser compartilhada por todos os envolvidos no processo de desenvolvimento de software. Elas podem ser tão genéricas quanto necessário permitindo o seu reuso e fácil extensão. Estas características tornam a ontologias úteis para representar o conhecimento das técnicas e metodologias para a Engenharia de Software e é um mecanismo de abstração apropriado para especificação de artefatos de software de alto nível como modelos de domínio, frameworks e padrões de software (GIRARDI et al., 2004).

As ontologias podem ser classificadas em: ontologias genéricas, ontologias de domínio, ontologias de tarefa e ontologias de aplicação (GUARINO, 1998).

Uma ontologia genérica descreve conceitos gerais, tais como, espaço, tempo, matéria, objeto, sendo independentes de domínio. Uma ontologia de domínio reúne conceitos de um domínio específico e seus relacionamentos, definindo restrições na estrutura e conteúdo do conhecimento desse domínio. Uma ontologia

de tarefa expressa conceitos sobre a resolução de problemas, independentemente do domínio em que ocorram, ou seja, descreve o vocabulário relacionado a uma atividade ou tarefa. Uma ontologia de aplicação descreve conceitos dependentes ao mesmo tempo de um domínio particular e de um conjunto de tarefas específicas. Estes conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio enquanto realizam certas atividades.

Os conceitos de uma ontologia de domínio ou de uma ontologia de tarefa podem ser especializados dos termos introduzidos por uma ontologia genérica. Os conceitos de uma ontologia de aplicação, por sua vez, podem ser especializações dos termos das ontologias de tarefas e de ontologias de domínio (Figura 2). As setas expressam relacionamentos de especialização.

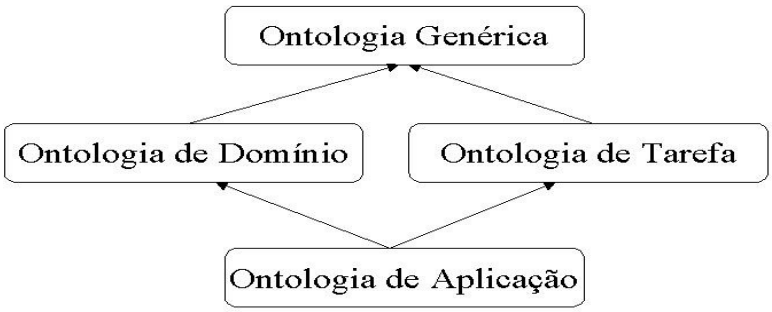

Figura 2 - Classificação das ontologias segundo o seu nível de generalidade. Fonte: (GUARINO, 1998)

As ontologias também podem ser diferenciadas segundo seu grau de formalidade. (USCHOLD, 1996) classifica uma ontologia como altamente informal, informal estruturada, semi-formal e rigorosamente formal, onde:

- **Altamente informal:** é expressa imprecisamente em linguagem natural como, por exemplo, um glossário;
- **Informal estruturada:** é expressa em linguagem natural, mas de forma estruturada e com o escopo do vocabulário bem-definido. Diminui consideravelmente o nível de ambigüidade em relação às ontologias do tipo altamente informal;
- **Semi-formal:** é expressa em uma linguagem artificial formalmente definida;
- **Rigorosamente formal:** os conceitos da ontologia são definidos em semântica formal, com teoremas e provas de suas propriedades.

As ontologias oferecem vantagens ao desenvolvimento de software, especialmente pela sua característica de compartilhamento de conhecimento independente de linguagem de representação, o que facilita o reuso de artefatos. Em (HAPPEL e SEEDOF, 2006) as vantagens do uso de ontologias são discutidas em vários aspectos do desenvolvimento de software:

- **Engenharia de Requisitos:** Na Engenharia de Software tradicional o conhecimento a respeito do domínio da aplicação fica separado do conhecimento acerca dos requisitos da aplicação, podendo trazer inconsistências entre os conceitos do domínio e o dos requisitos. Além disso, os requisitos muitas vezes são descritos em linguagem natural o que deixa margem para ambigüidades. Através do uso de ontologias todo esse conhecimento pode ser integrado evitando tanto o problema da ambigüidade quanto da inconsistência;
- **Reuso de Componentes:** as ontologias são úteis para selecionar componentes por causa da sua expressividade semântica permitindo representar, por exemplo, relacionamentos entre partes (“part of”) e de herança (“is a”), enquanto que a maioria dos repositórios tradicionais são limitados a buscas sintáticas por palavras chaves o que traz baixa precisão por causa das palavras homônimas e baixa recuperação por causa das palavras sinônimas;
- **Implementação:** as ontologias facilitam a documentação e reuso através da descrição dos artefatos de forma semântica. Se os produtos das fases de análise e projeto forem realizados utilizando ontologias como o código pode ser gerado automaticamente definindo-se regras de mapeamento de análise/projeto para implementação;

As ontologias de domínio de aplicação de sistemas de software podem ser utilizadas para evitar problemas e erros durante o ciclo de vida do produto de software: da análise dos requisitos inicias (facilitando a interação entre o cliente e o analista), passando pelas fases desenvolvimento e construção, e finalmente com o

estágio de manutenção (assegurando maior compreensão da modificação de requisitos, melhor entendimento do sistema de manutenção, etc.) (RUIZ, 2006).

2.5 Ferramentas Baseadas em Conhecimento para Engenharia de Software

Esta seção apresenta alguns sistemas baseados em conhecimento que de alguma forma contribuem na pesquisa da Engenharia de Software: o framework OSEE, para compartilhamento de conhecimento em processos de software entre desenvolvedores pela web; ARTICULATOR, um ambiente baseado em conhecimento para simulação de processos de desenvolvimento de software; o Ambiente ODE, um ambiente de desenvolvimento de software centrado em processo; OME3, um ambiente para análise e desenvolvimento de modelos e ONTORMAS-T, uma ferramenta para modelagem de sistemas multiagente. Ao final é apresentado um estudo comparativo entre as ferramentas.

2.5.1 Ontology-driven Software Engineering Environment – OSEE

Ontology-driven Software Engineering Environment (OSEE) (THADDEUS, 2006) é um framework baseado em ontologias para engenharia de software baseada em conhecimento. Tem por objetivo utilizar ontologias de produto e software para construir uma base de conhecimento e usá-la nas atividades de engenharia de software. Uma ferramenta para web semântica baseado em arquitetura multiagente foi construída como protótipo, utilizando ontologias como esquema formal de representação de conhecimento de produtos e processo de domínio.

A base de conhecimento é construída pela acumulação de conhecimento em engenharia de software como instâncias da ontologia e dissemina conhecimento para os desenvolvedores de software, através de um mecanismo de pesquisa semântico, fornecendo informações para aplicações automatizarem a geração de código a partir de modelos de domínio estruturados que estejam armazenados na ontologia ou automatiza a execução de processos baseados na estrutura de processos definidos na ontologia de processo de software.

OSEE atua sobre itens de conhecimento de uma organização que é um recurso humano e de difícil articulação. Este conhecimento é flexível e é estabilizado

gradualmente em conhecimento explícito (documentos de projeto, planos, estruturas de processo e pacotes, diretrizes técnicas, manuais), que por sua vez é facilmente adicionado como instâncias nas respectivas categorias ontológicas, através do entrelaçamento de muitos itens relativos ao mesmo conceito sob categorias ontológicas distintas. Este conhecimento é obtido de diversas atividades de engenharia de software:

- Especificações de software podem fornecer conhecimento que atualiza conceitos e seus relacionamentos sobre um domínio de aplicação em sua respectiva ontologia;
- Casos de uso da especificação de requisitos, que atualizam as características do sistema na ontologia de domínio;
- Registro da resolução do problema de execução de projetos ou relatórios de fechamento de projeto apontando casos de problema/solução acrescenta conhecimento às ontologias de processo.

Uma base de conhecimento embutida num framework da web semântica, com documentos de projeto explícitos ou com itens conhecimento implícito, é uma fonte útil para que desenvolvedores tenham acesso a informação instantânea de qualquer lugar e a qualquer momento.

A arquitetura OSEE é projetada em três camadas (Figura 3). A camada de dados inclui as ontologias de domínio do ambiente relativas a produtos e processos. Cada projeto é representado por uma ontologia distinta. Isso define o ambiente para um projeto pela adaptação e vinculação de ontologias de produtos e processos com conceitos e terminologias relativos a um projeto específico.

A camada de meio, semântica, é montada utilizando uma arquitetura multiagente para web semântica (DAI, 2005) e consiste de um conjunto de agentes realizando tarefas independentes como cooperação, concorrência ou coexistência entre si. Estes agentes são agrupados em três categorias:

- **Agente de coordenação** que recebe todas as solicitações da camada de aplicação e processa os pedidos identificando a tarefa, dividindo-as em subtarefas e alocando-as para os agentes de domínio. Após a execução ou termino de cada tarefa, ele envia o resultado para a camada de aplicação.

- **Agentes de domínio** que efetuam tarefas especializadas de acordo com a ontologia de domínio como formulação de consultas, uma solução especifica de domínio ou comunicação com outros agentes. Agentes de domínios diferentes representam produto, processo e projeto.
- **Agente Jena** que cria modelos de ontologia e interage com o ambiente JENA para modelagem de ontologias, inferindo ou atualizando dados instanciados.

A camada de aplicação consiste das ferramentas de interface com o usuário. Cada aplicação é uma entidade independente, que interage com a camada central, requisitando um serviço e obtendo uma resposta. A entrada do usuário é analisada gramaticalmente pelo mecanismo de busca, onde acontece a identificação de *tokens* chaves, com base nos conceitos e propriedades do domínio selecionado. Baseado no número de tokens e possível combinação de termos com respeito aos conceitos e propriedades, consultas são formuladas em RDQL e são executadas com a ajuda do motor de consultas do Jena. Como parte da técnica utilizada para a geração de código (KALYANPUR, 2004), técnicas de mapeamento são usadas para converter classes OWL em classes orientadas a objetos.

O protótipo do OSEE foi construído no contexto de aplicações de cuidados com a saúde, nomeado sistema de Administração de Pacientes. Foi utilizado como ontologia de processo o RUP e a ontologia de domínio da aplicação foi desenvolvida baseada no Modelo de Informação de Referência HL7 (Health Level Seven, um padrão ANSI, que é usado para padronização da linguagem médica).

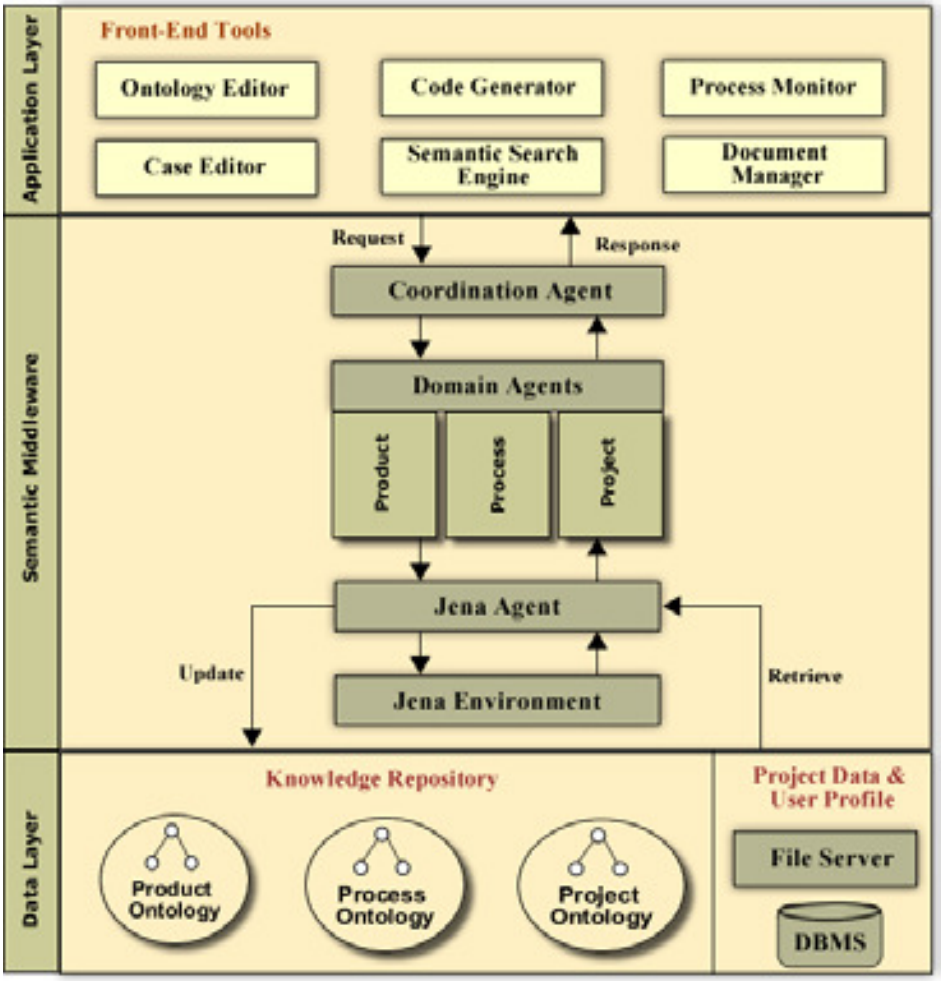

Figura 3 - Arquitetura da Aplicação OSEE

2.5.2 ARTICULATOR

Articulator (MI, 1990) é um ambiente baseado em conhecimento para simulação de processos de desenvolvimento de software. Ele provê um meta-modelo de processo de desenvolvimento de software, uma linguagem baseada em objetos e um mecanismo de simulação automática. Para simular a execução de processos, o Articulator usa uma abordagem multiagente onde os desenvolvedores são modelados como agentes cognitivos. A arquitetura do ambiente contém cinco subsistemas: base de conhecimento, simulador de comportamento, mecanismo de consulta, gerenciador de instanciação e gerenciador de aquisição de conhecimento.

A base de conhecimento implementa o meta-modelo do Aticulator através de uma abordagem orientada a objetos e consiste de recursos, agentes e tarefas. Os recursos são objetos nas tarefas dos agentes. As tarefas consomem e produzem recursos. Um agente apresenta uma coleção de comportamentos e atributos

associados. O comportamento do agente é visto durante a execução das tarefas (incluindo comunicação, acomodação e negociação) do modelo de desenvolvimento de software. Os agentes são modelos gerais de desenvolvedores, times de desenvolvimento e organizações. As tarefas são representadas através de uma rede de ações que os agentes realizam. Como não existe execução de processos neste ambiente, os agentes são agentes cognitivos que simulam a execução de tarefas e várias situações para auxiliar a geração da descrição do processo. A simulação é importante para detectar falhas como a alocação de recursos, cronograma, dentre outras.

O gerenciador de instanciação gerencia os relacionamentos entre o meta-modelo, os modelos de processo de software customizado e suas instâncias. Ele mantém todos estes relacionamentos de acordo com o tempo de criação e linhas de herança e recupera a instância correta quando requisitado. Em contraste com sistemas tradicionais de sistemas de banco de dados, não existe uma fronteira explícita entre o meta-modelo, o modelo de processo de software e suas instâncias. Todos eles podem ser manipulados e modificados como uma instância associada ao antigo sempre que necessário.

O simulador de comportamento controla a simulação de um dado modelo de processo de software e cria uma trajetória de processo, uma seqüencia de *snapshots* do desenvolvimento de software em um dado período de tempo, isto é, uma execução simbólica do modelo de processo de software.

O mecanismo de consulta fornece o meio para o usuário acessar a informação no Articulator. A fonte de informação é a base de conhecimento: o meta-modelo, os modelos de processo de software e suas instâncias. Usando técnicas de herança de classes e mecanismos de inferência, o mecanismo de consulta infere sobre a informação e conhecimento para determinar respostas a diversos tipos de questões.

O gerenciador de aquisição de conhecimento é uma interface para o Aticulator adquirir um modelo de processo de software e dados associados. O gerenciador de aquisição de conhecimento utiliza uma descrição de agentes estruturada, tarefas e recursos como entrada, então os traduz num modelo de processo de software configurado. Ele também avalia a informação adquirida dos projetos de desenvolvimento de software controlados pelo Articulator e guarda-os para uso posterior. Na Figura 4 é apresentada a arquitetura do Articulator.

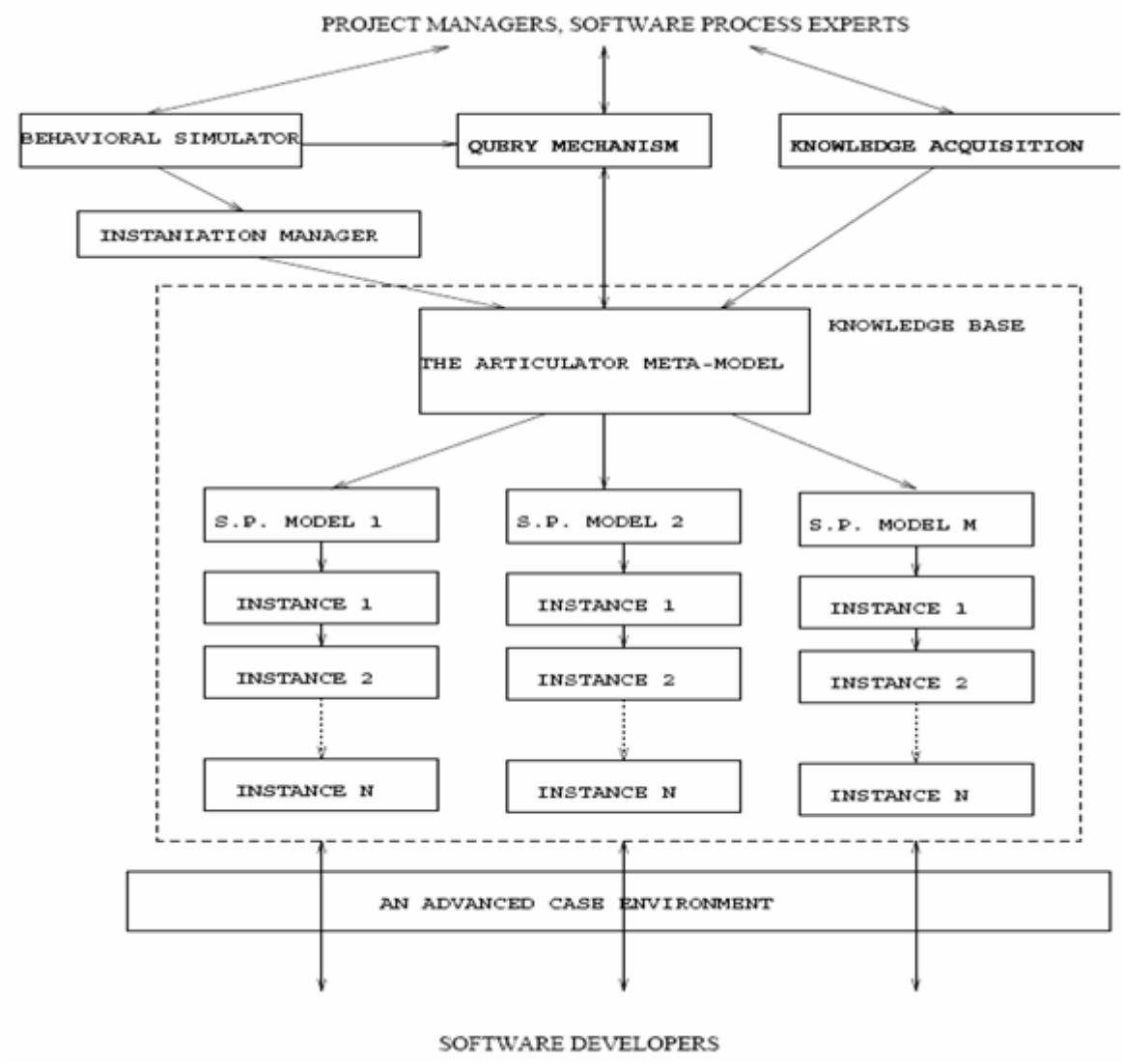

Figura 4 - Arquitetura do Articulator

2.5.3 Ambiente ODE

Ontology-based software Development Environment (ODE) (BERTOLLO et. al., 2002) (FALBO et. al., 2003) é um ambiente de desenvolvimento de software (ADS) centrado em processo, tendo sua fundamentação na ontologia de processo de software descrita em (FALBO, 1998) e recentemente foi avaliada e reformulada tendo por base uma ontologia de fundamentação (UFO) (GUIZZARDI et al., 2008), criando repositório de conhecimento, que proporciona ao ambiente uma uniformidade de conceitos, primordial na integração de ferramentas. Esta base ontológica permite a integração de ferramentas, pois os conceitos das ontologias,

tais como Processo, Atividade, Artefato, Recurso e Procedimento, estão bem definidos e são compartilhados pelas ferramentas do ambiente.

Se as ferramentas de um ADS são construídas com base em ontologias, a integração dessas ferramentas pode ser facilitada, pois os conceitos envolvidos estão bem definidos. Assim, a mesma ontologia pode ser usada para construir diferentes ferramentas, apoiando atividades relacionadas à engenharia de software (FALBO et. al., 2002).

Como forma de permitir a descrição do conhecimento de domínio no ambiente, ODE provê suporte ao desenvolvimento de ontologias de domínio através do editor de ontologias ODEd. A partir de uma ontologia, ODEd gera sistematicamente frameworks de objetos e permite a instanciação das ontologias através da criação automática de bases de dados e interfaces customizadas (MIAN e FALBO, 2002).

Também foi desenvolvida AgeODE (PEZZIN e FALBO, 2004), uma infra-estrutura para apoiar a construção de agentes para atuarem no ambiente ODE. agentes têm acesso aos objetos existentes em ODE, que podem ser vistos como parte de suas bases de conhecimento, e cooperam e negociam com outros agentes para alcançarem seus objetivos.

A arquitetura de possui dois níveis. O Metanível (Pacote Conhecimento), que define classes que descrevem o conhecimento sobre os objetos no Nível Base (Pacote Controle). As classes do pacote conhecimento são derivadas de ontologias, utilizando a abordagem de (GUIZZARDI et al., 2006). Seus objetos são itens da instanciação de uma ontologia. As classes do Pacote Controle não derivam diretamente de ontologias, mas são baseadas nelas. Na Figura 5, é apresentada parte do Pacote Controle. As classes que possuem um conhecimento correspondente têm essa associação representada na forma de atributos.

O controle de projetos de software e seus respectivos processos pode ser feito em ODE, consistindo basicamente de um conjunto de atividades e os elementos envolvidos na sua realização. É um ambiente configurável, pois se configura de acordo com o perfil do usuário, disponibilizando apenas as funcionalidades às quais o usuário tem acesso e as informações do processo que são úteis a ele, de acordo com os projetos aos quais está alocado. Um desenvolvedor deve estar alocado a um projeto de software que tenha um processo definido.

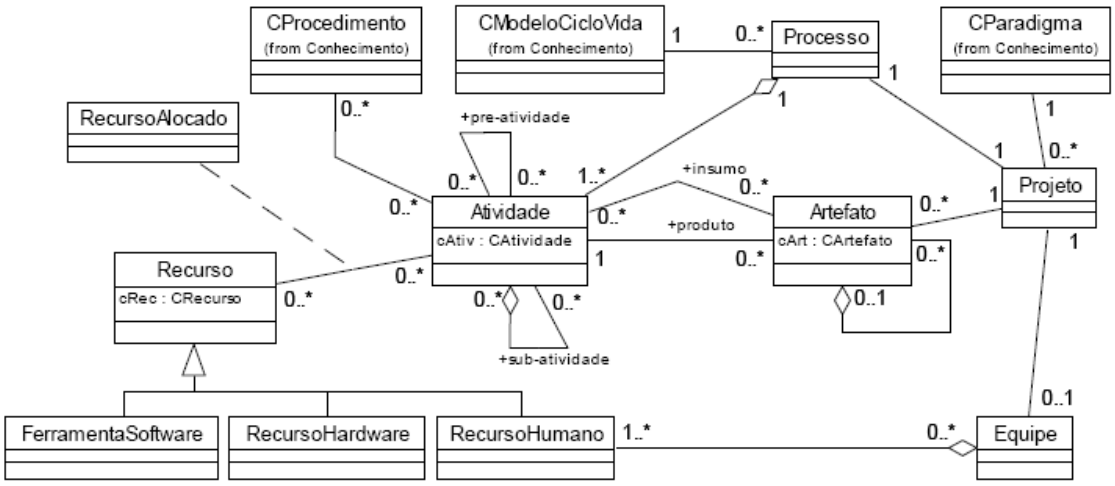

Figura 5 - Diagrama de Classes do Pacote Controle de ODE

ODE proporciona uma série de funcionalidades para tratar projetos como criar um novo projeto, abrir um projeto existente, caracterizar, excluir um projeto e definir o processo para o projeto. Caracterizar é definir as características relevantes do projeto, como o paradigma de desenvolvimento e equipe, podendo o ambiente fornecer ajuda na definição de uma série de informações sobre o projeto, tal como a sugestão de um modelo de ciclo de vida que melhor se adeque ao projeto em questão.

Para definir um processo de software é preciso escolher seu modelo de ciclo de vida e de acordo com as características do projeto, são sugeridos os modelos que melhor se ajustam ao projeto, filtrados a partir do repositório de conhecimento do ambiente, podendo o gerente de projeto aceitar a sugestão ou não.

O Pacote Conhecimento de ODE contém as macro-atividades que compõem a estrutura de cada modelo de ciclo de vida. Escolhido o modelo, essas macro-atividades são instanciadas para o processo, assim como a ordem de precedência das mesmas. A estrutura inicial pode ser redefinida, adicionando novas macro-atividades ou removendo aquelas não desejadas. A Figura 6 mostra a definição de processos em ODE.

Depois de definido o processo, é possível alocar recursos humanos para suas atividades e definir seu cronograma de execução. A alocação de recursos e a definição do cronograma são importantes para a configuração do ambiente, pois definem quais ferramentas cada usuário poderá utilizar em um determinado momento.

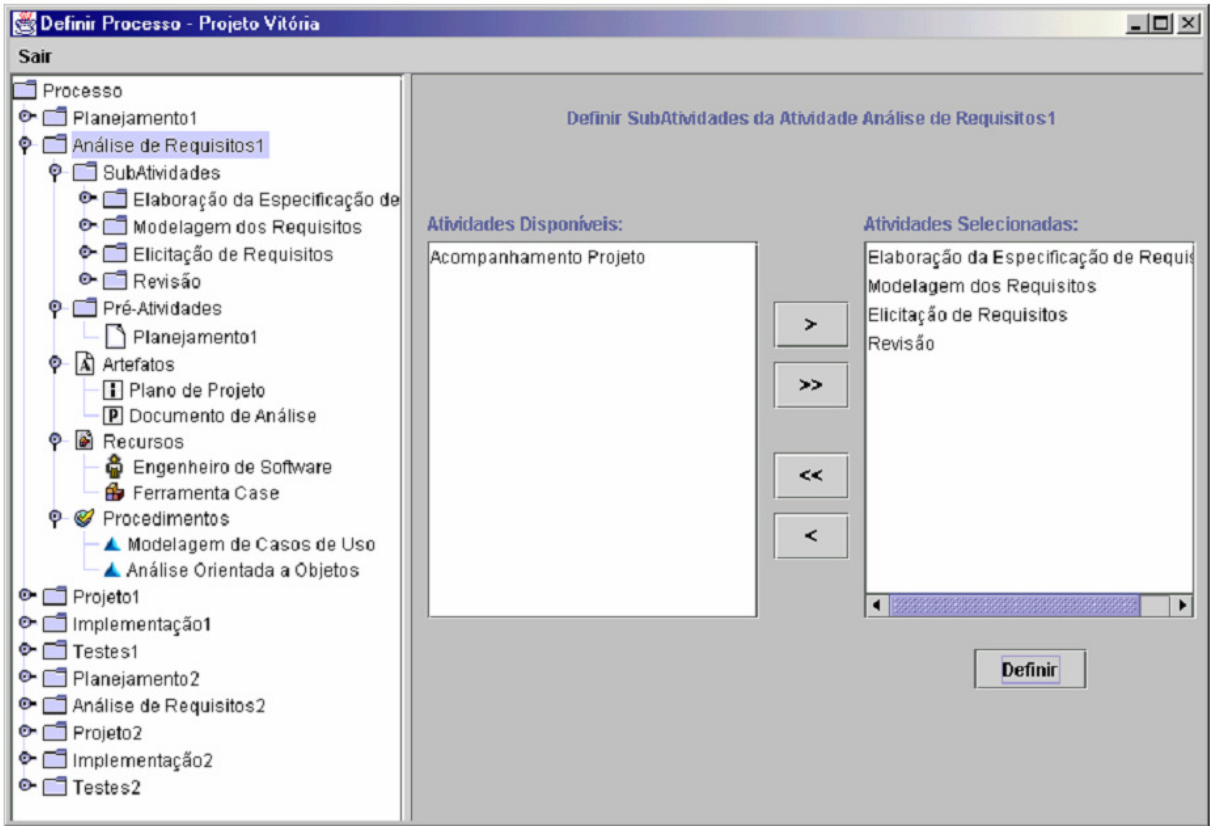

Figura 6 - Definição de processo em ODE

Uma vez que o projeto é iniciado, o ambiente permite que se faça o seu acompanhamento. Dessa maneira, é possível visualizar as atividades que compõem o processo de software, o estado em que se encontram, a ordem de precedência entre as mesmas e se são compostas ou não por outras atividades. O usuário pode, ainda, visualizar a quais atividades está alocado. Durante o andamento do projeto, os estados das atividades podem ser alterados.

ODE está implementado em Java, possuindo, assim, a característica de ser multiplataforma. A persistência é feita em um banco de dados relacional, usando uma camada de persistência para fazer o mapeamento entre o mundo de objetos e o mundo relacional.

2.5.4 OME3 Meta-Framework

Organization Modeling Environment (OME) (YU, 2000) é um ambiente de desenvolvimento e análise de modelos. Estes modelos são desenvolvidos dentro das restrições de um framework de modelagem, que é a especificação dos tipos de objetos que podem existir num modelo e as relações entre eles.

O ambiente de modelagem OME não pretende ser dedicado a nenhum framework de modelagem específico. Desenvolvedores e usuários podem criar novos frameworks descrevendo-os em Telos. Um framework pode ser apoiado pela ferramenta OME desde que estenda o meta-framework OME, orientado a objetos, disponibilizado junto com o ambiente. Desenvolver um framework em OME é o processo de descrever os tipos de objetos que podem existir num modelo desenvolvido no framework, que em essência, são uma coleção de objetos (instâncias de tipos descritas no framework) e seus atributos.

Existem duas classes principais de objetos que são implementadas: elementos e ligações. Elementos são os objetos básicos num modelo. Eles podem ser imaginados como os nós de um grafo. Ligações são usadas para representar conexões entre objetos no modelo. Ligações são similares a arestas num grafo, podem conectar qualquer par de objetos no modelo, outras ligações, são direcionais possuindo origem e destino.

A ferramenta OME3 é composta principalmente de duas partes: núcleo e *plugins.* O núcleo da OME é uma arquitetura em camadas que engloba três módulos: base de conhecimento, camada de modelos e camada de visão. Plugins são classes implementando funcionalidades de frameworks específicos, que podem ser juntados ao núcleo da OME e prover serviços para o usuário em tempo de execução. Em cada execução da ferramenta, diferentes tipos de plugins são carregados de acordo com os arcabouços nos quais o modelo atual carregado é baseado, por meio de um grupo de arquivos descrevendo a configuração destes frameworks.

Na Figura 7 vemos à esquerda os três módulos que compõem o núcleo da ferramenta OME3. No centro da arquitetura está a base de conhecimento, que guarda os objetos de um modelo, seus relacionamento e seus atributos (pertinente ao modelo) e também detalhes dos modelos dos frameworks. A base de conhecimento realiza checagens semânticas básicas, garantindo que o modelo sendo desenvolvido está coerente com o framework presente na base de conhecimento. Na camada acima se encontra o módulo de modelos ou framework, abstraindo a base de conhecimento do resto da ferramenta, por meio de objetos que servem como interface para a base de conhecimento. Acima da camada de modelos está o modulo de visão, responsável por apresentar os modelos ao usuário, permitindo-o modificá-los através de uma interface gráfica. Esta camada também

permite o usuário acessar as funcionalidades providas pelos plugins. Este conjunto de módulos permite a ferramenta realizar as tarefas de criação, alteração e análise de qualquer modelo. À direita temos o componente Plugin, composto de um conjunto de módulos que estendem a funcionalidade da ferramenta. Um plugin é uma coleção de métodos, que podem analisar ou alterar a visão de um modelo.

Os frameworks disponíveis para um modelo particular são determinados pelos arquivos de descrição de frameworks. Estes arquivos especificam os tipos de objetos que podem existir num modelo e as relações existentes entre si. Quando se inicia a modelagem de um novo modelo, o usuário deve escolher qual framework deve ser utilizado.

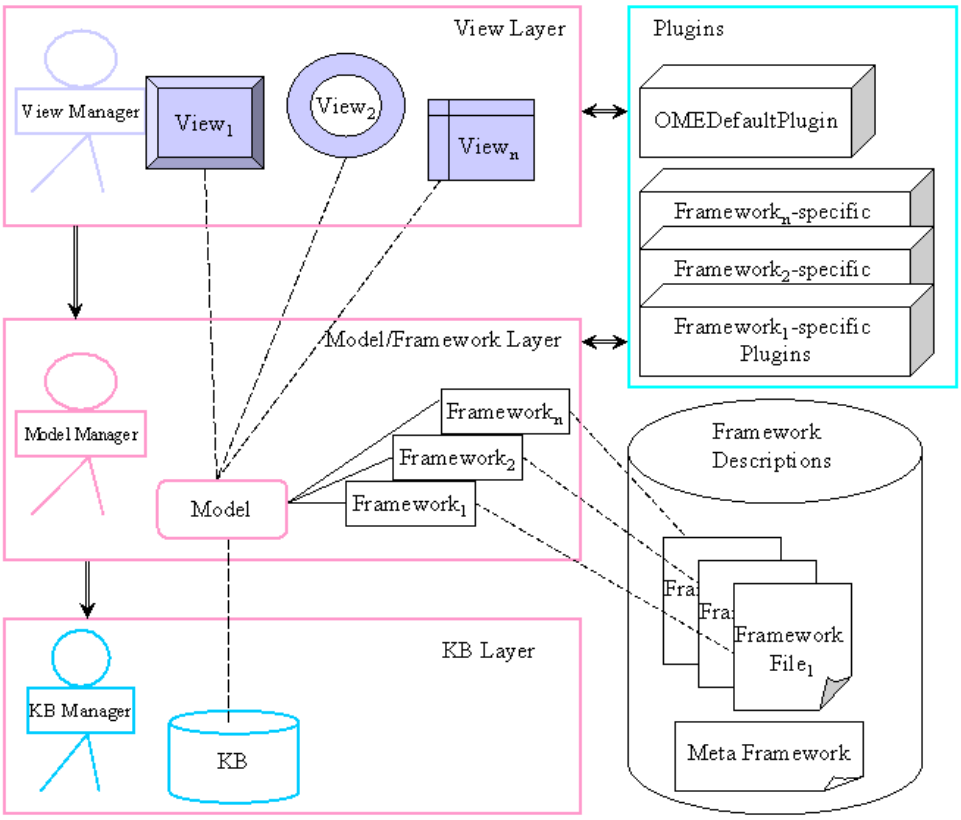

Figura 7 - Visão geral da arquitetura OME3

OME3 foi projetado com uma natureza dinâmica e extensível, para que usuários avançados possam facilmente adaptá-lo para prover suporte a novos frameworks de modelagem, provendo os usuários com rotinas para o desenvolvimento de *plugins* para desenvolver e analisar modelos destes frameworks.

2.5.5 ONTORMAS-T

A ferramenta ONTORMAS-T, é uma ferramenta desenvolvida no ambiente de desenvolvimento de sistemas baseados no conhecimento Protégé (GENNARI, 2008). Ela contém a ONTORMAS (LEITE et al., 2008a) é uma ontologia que representa uma conceitualização das metodologias MADEM (GIRARDI e MARINHO, 2007) e MAAEM (LINDOSO, 2006) (LEITE et al., 2008b). Ela é uma extensão da ONTOMADEM (*Ontology for Multi-Agent Application Domain Engineering Methodology*) (GIRARDI e LEITE, 2008) que conceitualizava apenas a metodologia MADEM (DRUMOND et. al., 2008).

Na ONTORMAS foram expressas todas as diretrizes para o desenvolvimento de sistemas multiagente especificadas por essas metodologias. É formada por um conjunto de classes organizadas hierarquicamente, tendo como superclasses principais: “Variable Concepts”, “Modeling Concepts”, “Modeling Tasks” e “Modeling Products”. A superclasse “Variable Concepts” e suas correspondentes subclasses são utilizadas para especificar a variabilidade de uma família de aplicações multiagente (Figura 8). Isso é realizado através da definição de “pontos de variação” e de “variantes”. Um ponto de variação é a representação de um conceito sujeito a variação. Uma variante representa uma variação do conceito propriamente dito. Uma variante poderá representar um grupo de variações alternativas ou opcionais. A superclasse “Modeling Concepts” especifica os conceitos fundamentais de modelagem presentes nas metodologias MADEM e MAAEM. Esses conceitos são básicos para o desenvolvimento de sistemas multiagente e estão especificados em níveis de abstração indo do nível mais abstrato (Fase de Análise dos Requisitos da Aplicação) ao mais concreto (Fase de Implementação) (LEITE, 2009). Os conceitos de modelagem podem ter entre si a relação de sinonímia, generalização ou especialização. Na ONTORMAS, conceitos específicos de uma plataforma de implementação de agentes, como o JADE, também podem ser representados para possibilitar a geração automática de código.

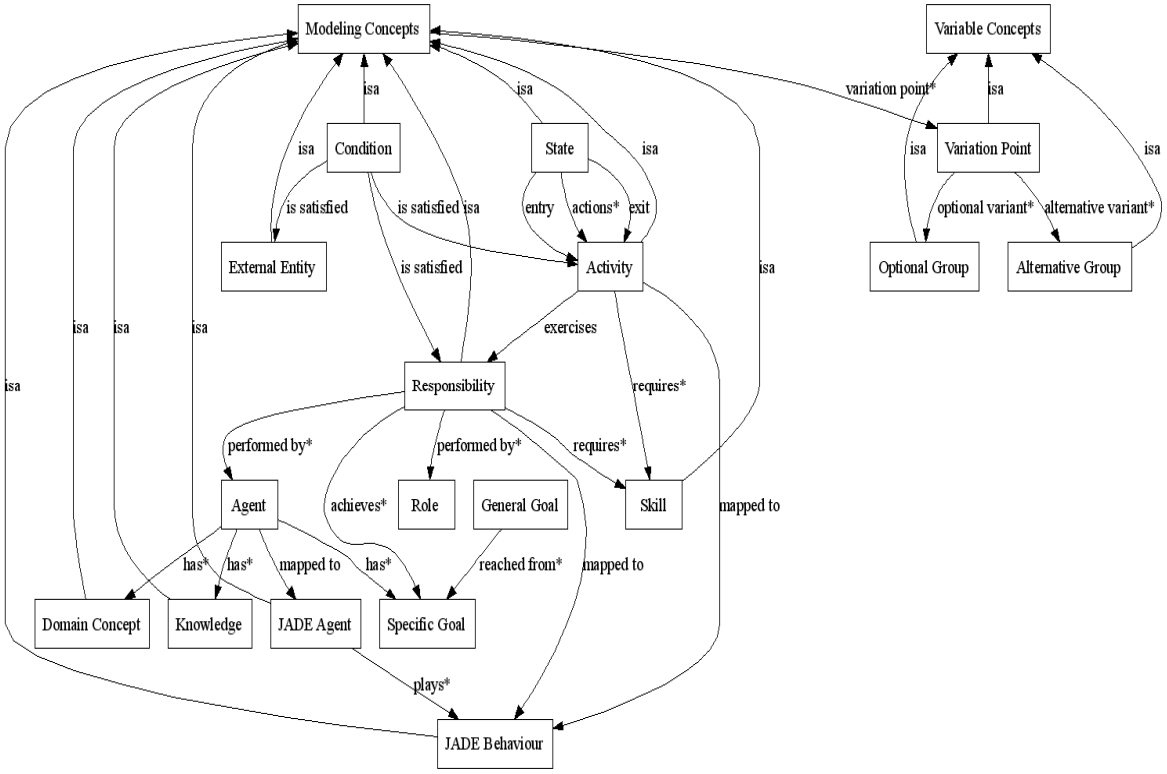

Figura 8 - Ontologia com os principais conceitos de modelagem especificados pelas metodologias MADEM e MAAEM

Na superclasse “Modeling Tasks” e em suas correspondentes subclasses são definidas as tarefas de modelagem especificadas pelas metodologias MADEM e MAAEM.

A Figura 9 ilustra a representação das tarefas realizadas na fase de Análise de Domínio e na de Análise dos Requisitos da Aplicação. Essas tarefas são divididas em “Domain Engineering Tasks”, cujas subtarefas são relacionadas à metodologia MADEM e “Application Engineering Tasks”, relacionadas à metodologia MAAEM. Na superclasse “Modeling Products” estão definidas as classes que representam os possíveis produtos de modelagem produzidos através das metodologias MADEM e MAAEM. Alguns destes produtos como “Domain Model” são compostos por subprodutos, isso é especificado através do relacionamento “subproducts”.

Na Figura 10, classes e instâncias do Modelo de Objetivo de ambas as metodologias são ilustradas, entre outros conceitos.

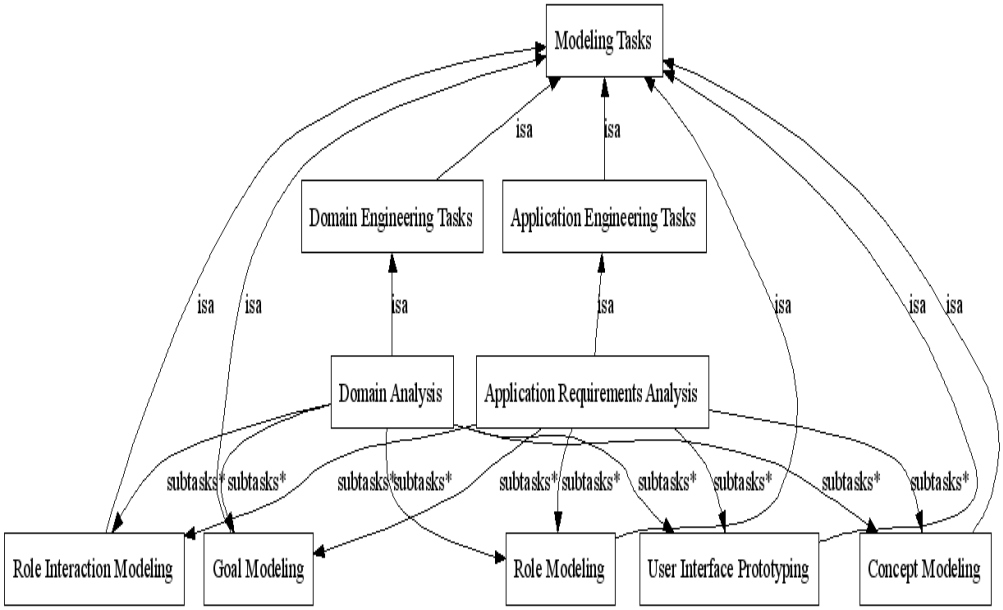

Figura 9 - Rede semântica das tarefas e subtarefas referentes à fase de Análise das metodologias MADEM e MAAEM

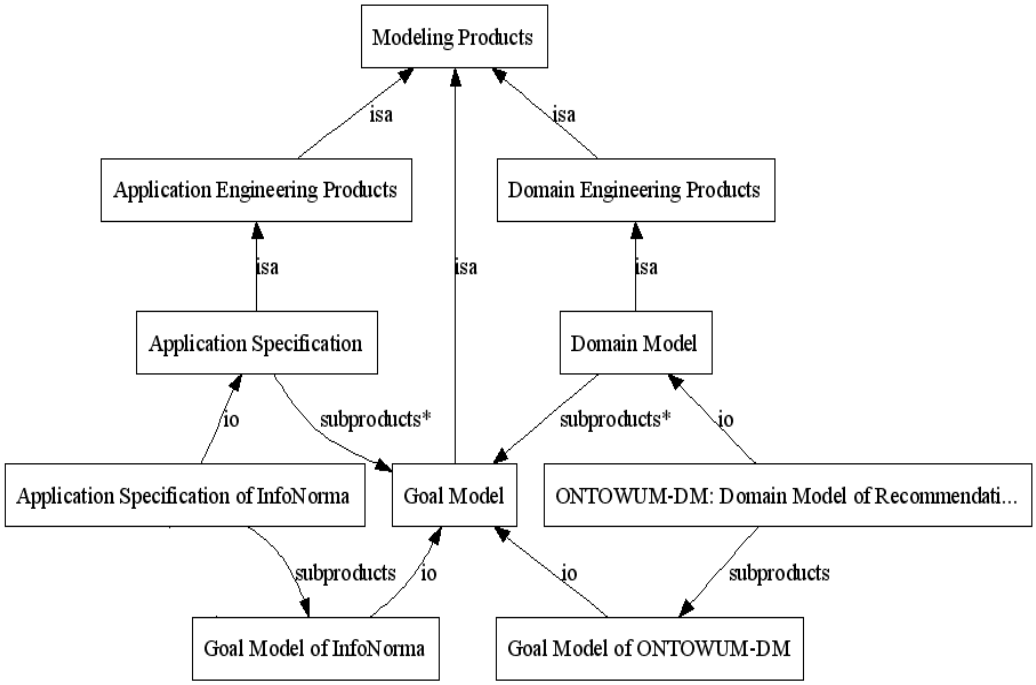

Figura 10 - Rede semântica exemplificando o relacionamento entre classes e instâncias de produtos presentes nas metodologias MADEM e MAAEM

A ONTORMAS-T é utilizada no processo MADAE-Pro (LEITE, 2009), para o desenvolvimento de sistemas multiagente, para realizar três tarefas básicas. A primeira é para realizar a criação dos modelos definidos no processo. A segunda é como repositório desses artefatos e, por último, como ferramenta de seleção, adaptação e integração dos componentes.

A aplicação das metodologias MADEM e MAAEM e os produtos das mesmas são obtidos através da instanciação dos correspondentes conceitos na hierarquia de classes da ONTORMAS. A linguagem de modelagem utilizada em todos os produtos presentes na ONTORMAS é a MADAE-ML, apresentada em detalhes no próximo capítulo.

A Figura 11 ilustra a criação do modelo de domínio ONTOWUM-DM e seus respectivos subprodutos. ONTOWUM-DM descreve os requisitos comuns e variáveis de uma família de aplicações multiagente para fornecer recomendações personalizadas através da Mineração de Uso. Na ONTORMAS, para criar um Modelo de Domínio é necessário a instanciação das classes que especificam suas tarefas de modelagem (“Concept Modeling”, “Goal Modeling”, “Role Modeling”, “Role Interaction Modeling” e “User Interface Prototyping”) e das que especificam os seus subprodutos (“Concept Model”, “Goal Model”, “Role Model”, “Role Interaction Models” e “Prototype of the User Interface”). Uma vez que o Modelo de Domínio e seus respectivos subprodutos estejam instanciados, tornam-se parte da base de conhecimento da ONTORMAS, sendo então utilizados em consultas e inferências através de raciocínio lógico, facilitando o seu reuso.

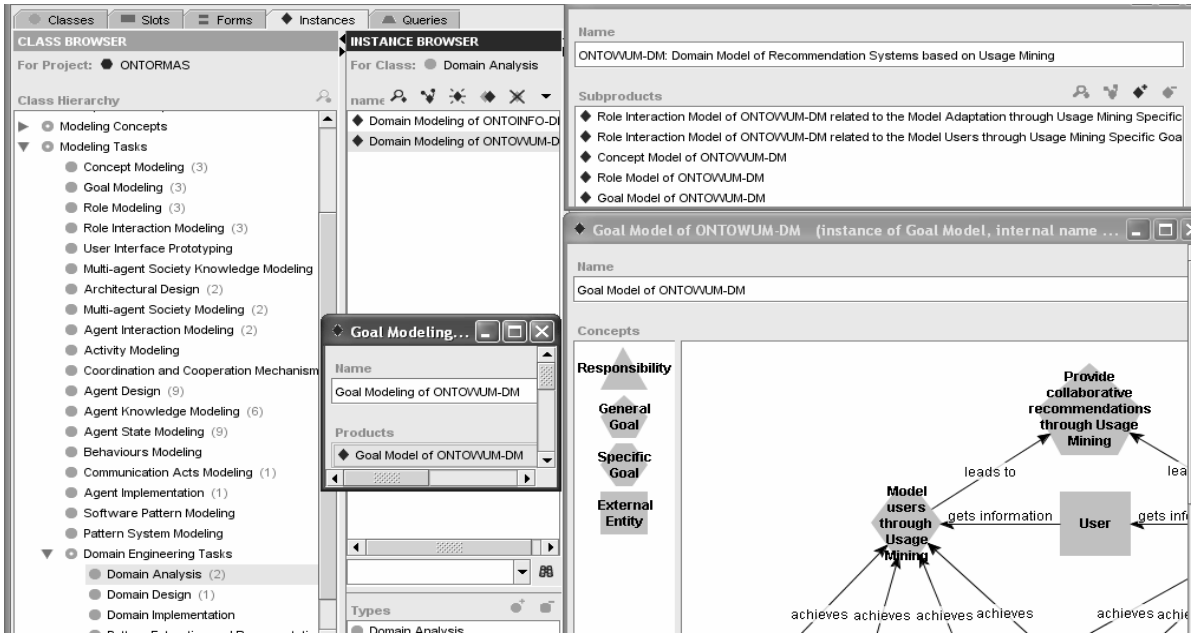

Figura 11 - Processo de Instanciação da ONTORMAS para criação de um Modelo de Domínio

2.5.6 Comparação entre as ferramentas

A ferramenta nada mais é do que um meio para atingir-se um objetivo, sendo por isso mesmo apenas uma variável do processo decisório. Cada projeto possui características que tornam essa decisão particular, sendo impossível definir-se um modelo ou solução genérica também ótima, havendo possibilidades e opções diversas, em função de cada caso.

Esta seção apresenta uma comparação entre as ferramentas descritas, portanto é necessário haver um critério de classificação das mesmas, de forma que possamos comparar suas características.

A Tabela 2 situa a utilização de cada ferramenta na Engenharia de Software. A Tabela 3 lista os insumos e produtos de cada ferramenta. Pode-se perceber que modelos são itens presentes em quase todas as ferramentas, exceto no framework OSEE.

Pela Tabela 4 visualizam-se maiores diferenças entre as ferramentas. A arquitetura está relacionada em como o a ferramenta foi implementada: OSEE, ARTICULATOR e ODE estão implementadas no paradigma de agentes e de orientação a objetos, o framework OME3 é um conjunto de classes orientadas a objetos e a ONTORMAS-T é construída como um sistema de frames. Suas bases de conhecimento foram implementadas de diversas formas, desde OWL e banco dados convencionais, passando por classes em linguages orientadas a objetos até Telos e Prolog. ODE, OME3 e ONTORMAS-T podem ser extensíveis através de novas ferramentas. Das cinco ferramentas, ARTICULATOR e ODE utilizaram linguagem declarativa no seu desenvolvimento.

Todas as ferramentas possuem interface gráfica, como visto na Tabela 5. Um problema encontrado no estudo da maioria das ferramentas é a quantidade de documentação encontrada. Dentre as ferramentas, apenas OME3 não permite que o usuário realize consultas semânticas na base de conhecimento. Este é um item importante, pois permite que o usuário explore as relações semânticas contidas nos modelos. O framework OSEE é o único desenvolvido para utilização na web. Todas as demais são utilizadas de forma monolítica.

Tabela 2 - Utilização da ferramenta na Engenharia de Software

Ferramenta	Utilização
OSEE	Processo de Engenharia de Software
ARTICULATOR	Gerência e Simulação de Processos
ODE	Definição de Processos de Software
OME3	Modelagem Orientada a Objetivos E/OU Agentes e Análise de Modelos
ONTORMAS-T	Engenharia de Domínio e de Aplicações Multiagente

Tabela 3 - Insumos e produtos das ferramentas

Ferramenta	Insumos e Produtos
OSEE	Ontologias de Produtos, Ontologias de Processos de Domínio, Base de Conhecimento em Engenharia de Software, Código executável
ARTICULATOR	Aquisição de Conhecimento, Meta-Modelos de Processos de Software
ODE	Axiomas, Ontologias, Modelos de Processos de Software, Tutoriais, Frameworks de Objetos
OME3	Base de Conhecimento de Modelos, Arcabouços de Modelagem
ONTORMAS-T	Modelos para Engenharia de Software Multiagente, Agentes JADE

Tabela 4 - Arquitetura da ferramenta e técnicas da inteligência artificial

Ferramenta	Arquitetura	Agentes	Base de Conhecimento	Extensível	Linguagem Declarativa
OSEE	Camadas OO Agentes	Sim	OWL SGBD	Não	Não
ARTICULATOR	Camadas OO Agentes	Sim	OO	Não	Sim
ODE	Camadas OO Agentes	Sim	OO Prolog	Sim	Sim
OME3	Camadas OO	Não	Telos	Sim	Não
ONTORMAS-T	Frames	Não	OWL	Sim	Não

Tabela 5 - Facilidades de utilização

Ferramenta	Interface Gráfica	Plataforma	Documentação	Busca Semântica
OSEE	Sim	Web	Baixa	Sim
ARTICULATOR	Sim	Monolítica	Média	Sim
ODE	Sim	Cliente-Servidor	Boa	Sim
OME3	Sim	Monolítica	Média	Não
ONTORMAS-T	Sim	Monolítica Cliente-Servidor	Boa	Sim

2.6 Considerações Finais

Este capítulo apresentou conceitos fundamentais do desenvolvimento baseado de Engenharia de Conhecimento e no reuso de software.

Primeiro, foi apresentada uma introdução à Engenharia de Conhecimento, contextualizando os sistemas baseados em conhecimento dentro da área de Inteligência Artificial.

Em seguida, foram apresentadas as abordagens de reuso baseadas na Engenharia de Domínio, Engenharia de Aplicações e ontologias.

Por fim, foram estudas algumas ferramentas para Engenharia de Software embasadas na teoria da Engenharia do Conhecimento, sendo realizado um estudo comparativo entre elas.

Os conceitos discutidos neste capítulo são a base teórica para a compreensão do processo MADAE-Pro, a ser apresentado no próximo capítulo e do desenvolvimento do ambiente MADAE-IDE apresentado no Capítulo 5.

3 MADAE-Pro – UM PROCESSO BASEADO NO CONHECIMENTO PARA A ENGENHARIA DE DOMÍNIO E APLICAÇÕES MULTIAGENTE

O MADAE-Pro (LEITE, 2009), (LEITE e GIRARDI, 2009) é um processo para o desenvolvimento e reúso de famílias de sistemas multiagente. Ele integra dois subprocessos complementares. Um é baseado nos conceitos da Engenharia de Domínio, isto é, visa construir artefatos reutilizáveis que representem uma família de aplicações e o outro, baseado na Engenharia de Aplicações, guia o desenvolvimento de uma aplicação específica reutilizando os produtos do primeiro subprocesso.

Outra característica do MADAE-Pro é ser um processo baseado no conhecimento. Isto é, a definição das suas fases, tarefas e produtos são classes da ontologia ONTORMAS e os relacionamentos entre esses conceitos são especificados através dos “slots” dessa ontologia. Durante o desenvolvimento das famílias de sistemas multiagente as classes da ONTORMAS são instanciadas formando uma base de conhecimento, o que faz da ONTORMAS também um repositório para os modelos desenvolvidos com o MADAE-Pro.

O MADAE-Pro envolve o uso de metodologias que especificam “como” realizar as tarefas de desenvolvimento, de um ciclo de vida que define a “ordem” de realização dessas atividades de desenvolvimento, de papéis do processo que representa “quem” é responsável por realizar cada tarefa e de tecnologias (ferramentas e linguagens) que apóiam os papéis do processo na realização de suas atividades.

Na rede semântica ilustrada na Figura 12 estão os principais elementos envolvidos no MADAE-Pro. O processo adota as metodologias MADEM e MAAEM. A MADEM guia o desenvolvimento de artefatos de software reusáveis integrantes de uma família de aplicações multiagente enquanto a MAAEM, o desenvolvimento de uma aplicação de software multiagente particular que pode ocorrer sem reuso ou reutilizando artefatos de software desenvolvidos a partir da metodologia MADEM. Uma técnica está associada com cada fase de desenvolvimento para guiar as tarefas de desenvolvimento das metodologias. As técnicas GRAMO, DDEMAS e DIMAS integram a metodologia MADEM, e estão associadas, respectivamente, as

suas fases de Análise de Domínio, Projeto de Domínio e Implementação de Domínio. As técnicas SRAMO, ADEMAS e AIMAS integram a metodologia MAAEM e estão associadas, respectivamente, as suas fases de Engenharia dos Requisitos da Aplicação, Projeto da Aplicação e Implementação da Aplicação.

O ciclo de vida adotado no MADAE-Pro é o iterativo e incremental, devido a complexidade inerente ao desenvolvimento de sistemas multiagente. O processo é apoiado pela ferramenta ONTORMAS, que é utilizada para guiar as tarefas de desenvolvimento, realizar a modelagem visual, documentar e armazenar os artefatos produzidos durante a execução do processo. A linguagem para modelagem de sistemas multiagente adotada é a MADAE-ML (LEITE, 2009). Esta linguagem estabelece uma representação gráfica para os modelos e conceitos de modelagem das metodologias MADAEM e MAAEM. Por fim, o processo define os papéis do processo (ex. Programador, Analista de Sistemas), encarregados da realização de uma ou mais tarefas durante a execução do processo.

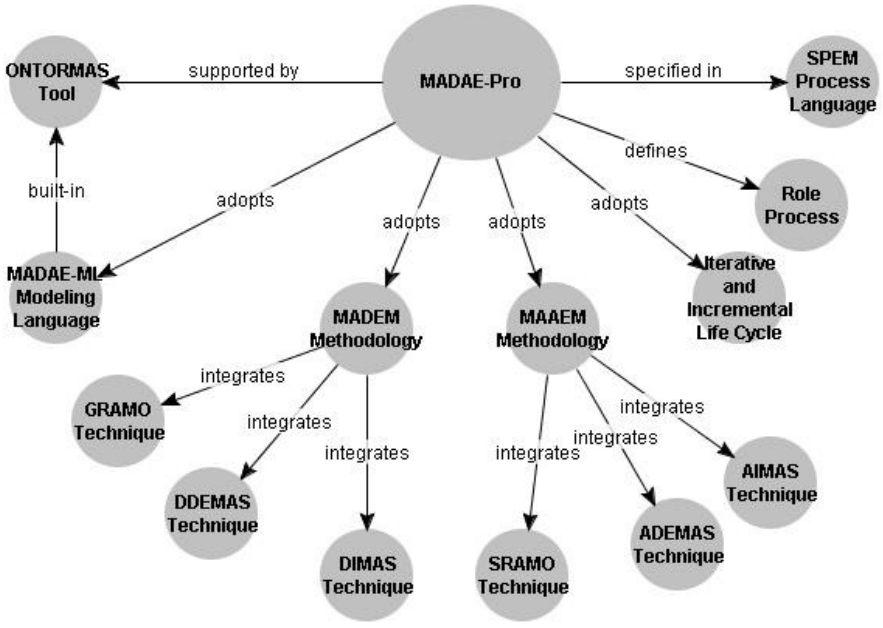

Figura 12 - Elementos envolvidos no MADAE-Pro

Quanto ao reúso o processo MADAE-Pro se enquadra na classificação de (PRIETO-DÍAZ, 1993), da seguinte forma:

- **Quanto ao conteúdo:** *componentes*, que são, no nível de especificação, os modelos desenvolvidos com instâncias da ONTORMAS. No nível de implementação, os componentes são agentes de software.

- **Quanto ao escopo:** *vertical*, pois o reúso é realizado em um mesmo domínio de aplicação.
- **Quanto ao modo:** *sistemático*, utilizando os processos da Engenharia de Domínio e de Aplicações.
- **Quanto à técnica:** composicional.
- **Quanto à intenção**: caixa-branca ("*white-box")*, o reúso tanto no nível de especificação quanto no nível de implementação permite a realização de adaptações.
- **Quanto ao Produto**: modelos de Domínio, Frameworks, Agentes.

3.1 O ciclo de vida

A Figura 13 ilustra o ciclo de vida do processo MADAE-Pro, composto por seis fases de desenvolvimento. As três primeiras, Análise de Domínio, Projeto de Domínio e Implementação de Domínio, referem-se à Engenharia de Domínio Multiagente, onde se constrói artefatos genéricos destinados ao reúso. E as três últimas, Análise da Aplicação, Projeto da Aplicação e Implementação da Aplicação, referem-se à Engenharia de Aplicação Multiagente, onde se constrói uma aplicação específica a partir do reúso dos artefatos desenvolvidos na Engenharia de Domínio Multiagente (LEITE, 2009).

O ciclo de vida é iterativo, incremental e dirigido por objetivos. O desenvolvimento é realizado a partir de sucessivos incrementos, com o intuito de reduzir a sua complexidade.

O ciclo inicia-se a partir da decisão de desenvolvimento de uma nova família de aplicações ou de uma aplicação específica (novo objetivo geral), conforme ilustrado no losango da figura.

O ciclo reinicia-se com a decisão de incrementar aquelas através da introdução de um novo objetivo específico (manutenção evolutiva) ou de alterar um objetivo específico preexistente (manutenção evolutiva e/ou corretiva). Iterações também podem ocorrer entre as fases, com o objetivo de aprimorar os produtos da modelagem.

Uma técnica está associada a cada fase de desenvolvimento para guiar a realização das tarefas de modelagem: as técnicas GRAMO (GIRARDI e FARIA,

2004), DDEMAS (GIRARDI e LINDOSO, 2005), DIMAS que integram a metodologia MADEM e as técnicas SRAMO (LINDOSO e GIRARDI, 2006), ADEMAS (LINDOSO e GIRARDI, 2005) e AIMAS que integram a metodologia MAAEM. Nas próximas seções, o subciclo de vida de cada fase definida no ciclo de vida do processo MADAE-Pro é apresentado.

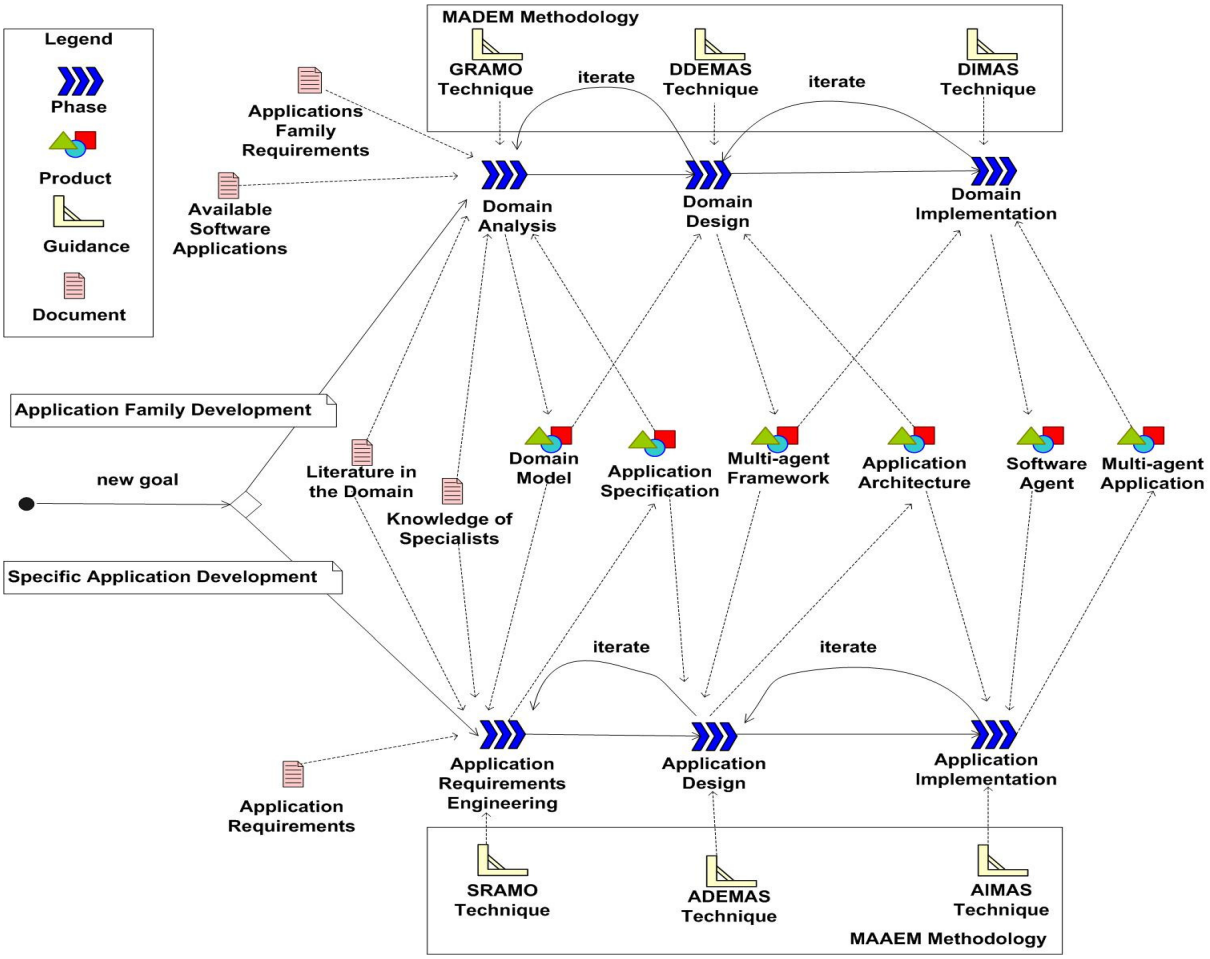

Figura 13 – O Ciclo de Vida do Processo MADAE-Pro

3.1.1 A Fase de Análise de Domínio

A fase de Análise de Domínio produz um Modelo de Domínio, que representa os requisitos comuns e específicos de uma família de aplicações multiagente. Essa fase é guiada pela técnica GRAMO.

A Figura 14 apresenta o subciclo de vida da fase de Análise de Domínio. O fluxo de realização das tarefas inicia-se com a modelagem de conceitos que utiliza

como insumos conhecimento de especialistas no domínio, aplicações já desenvolvidas e a literatura disponível no domínio para definir o escopo inicial do domínio. Ao final desta tarefa, tem-se como produto o Modelo de Conceitos, uma rede semântica com os principais conceitos do domínio. A seguir é feita a modelagem de objetivos que utiliza como insumos os requisitos da família de aplicações para determinar os objetivos geral e específicos da família, as entidades externas com as quais é necessário interagir e as responsabilidades necessárias para alcançar esses objetivos. Ao final desta tarefa é produzido o Modelo de Objetivos. O próximo passo é a realização da modelagem de papéis, onde cada responsabilidade definida no Modelo de Objetivos é associada a um papel que será encarregado delas. Nesse modelo, também é definido qual é o conhecimento que o papel necessitará para a realização de suas responsabilidades e as pré e pós condições que a realização dessas responsabilidades deverá atender. O produto dessa tarefa é um Modelo de Papéis. A seguir é realizada a modelagem de interações entre papéis. Essa tarefa representa a forma em os papéis se comunicam para realizarem suas responsabilidades a fim de atingirem os objetivos específicos. O produto dessa tarefa é um conjunto de Modelos de Interações entre Papéis, um para cada objetivo específico no nível inferior da hierarquia de objetivos no Modelo de Objetivos. Por último, é realizada a prototipagem da interface com o usuário, cujo objetivo é identificar as interações dos usuários com o sistema e construir um protótipo da interface com o usuário.

Para cada tarefa de modelagem descrita anteriormente, é realizada a subtarefa de modelagem de variabilidades com objetivo de identificar e representar as características comuns e variáveis da família de aplicações. Ao final desta fase, conforme o losango da Figura 14, passa-se para a fase de Projeto de Domínio ou inicia-se um novo ciclo para realizar um refinamento dos modelos a fim de adicionar novos requisitos ou alguma eventual correção.

Os papéis do processo “System Analyst” e “Domain Expert”, ilustrados na Figura 15, são os responsáveis pela realização das tarefas da fase de Análise de Domínio.

Essas tarefas são realizadas em conjunto pelos dois papéis do processo. O papel do processo “System Analyst” é o principal responsável pela realização e coordenação das tarefas, enquanto que “Domain Expert” o auxilia através do conhecimento que possui do domínio.

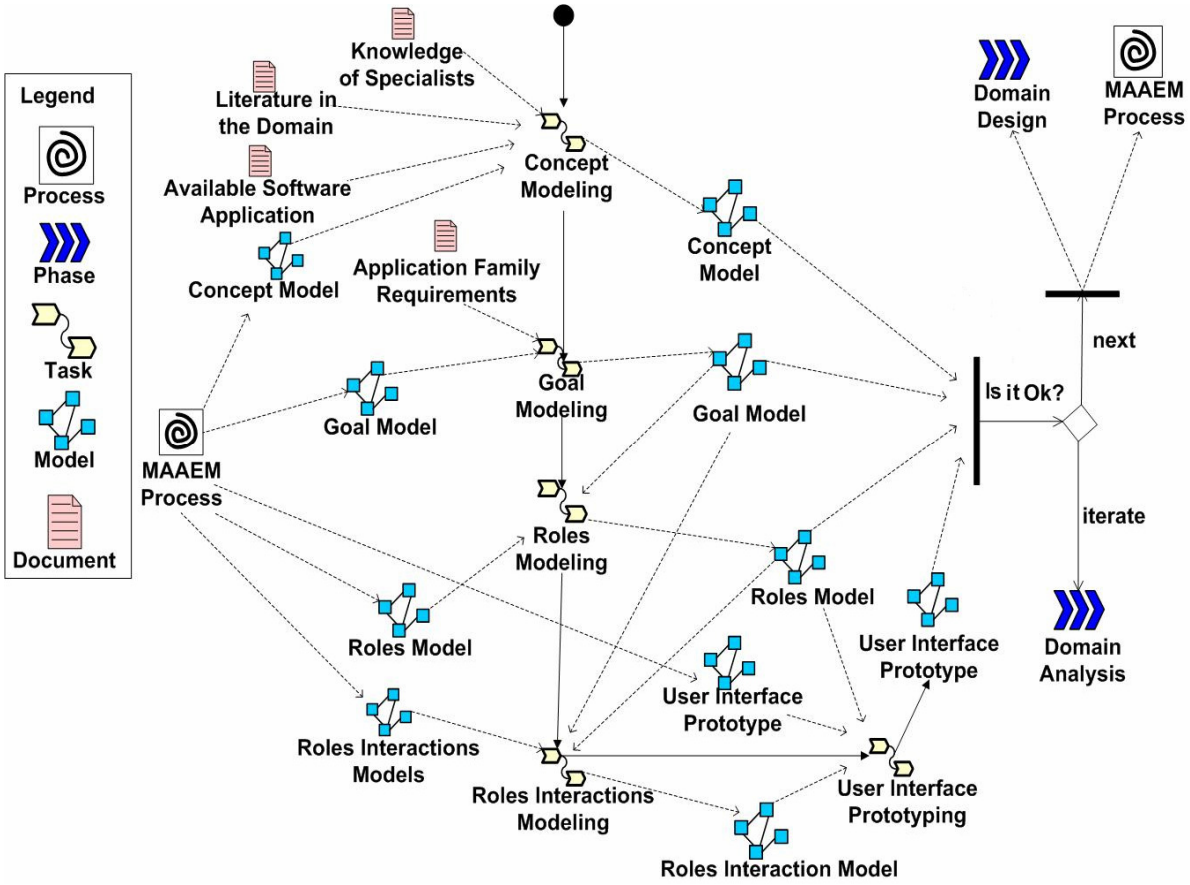

Figura 14. Subciclo de vida referente à fase de Análise de Domínio

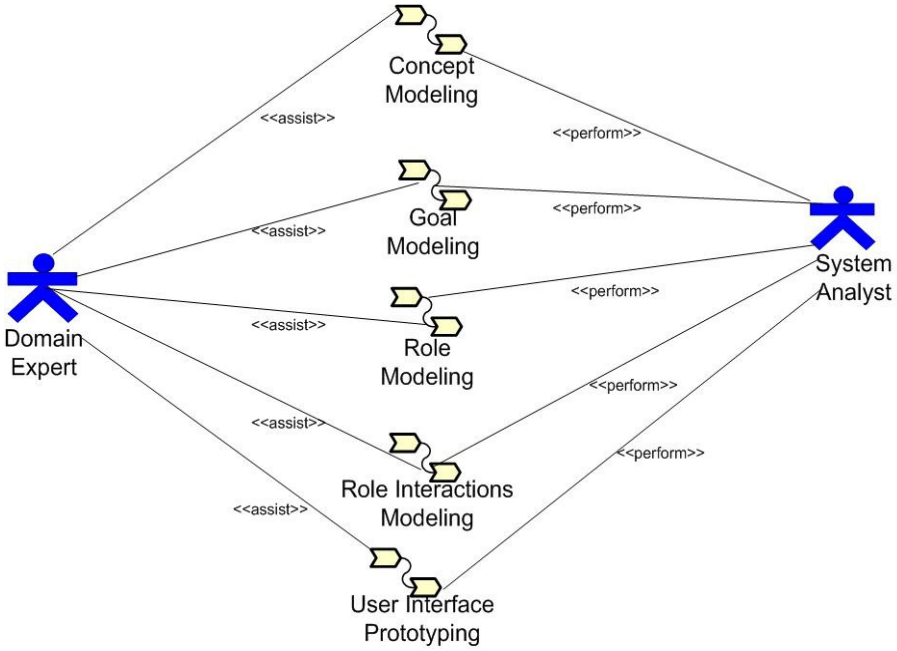

Figura 15 - Papéis do Processo da Fase de Análise de Domínio

3.1.2 A Fase de Projeto de Domínio

A fase de Projeto de Domínio visa definir uma solução de projeto aos requisitos de uma família de aplicações multiagente definidos na fase de Análise de Domínio. Nessa fase é realizado o Projeto Arquitetural, onde se constrói um modelo arquitetural da sociedade multiagente e o Projeto do Agente, que define a estrutura interna de cada agente da sociedade.

O subciclo de vida da fase de Projeto de Domínio, ilustrado na Figura 16, consiste na realização das tarefas de Projeto Arquitetural e Projeto do Agente, guiadas pela técnica DDEMAS. O subciclo inicia-se com a realização da subtarefa de modelagem do conhecimento da sociedade multiagente, onde é definido o conhecimento compartilhado pelos agentes da sociedade. Os insumos utilizados nessa tarefa são os *Modelos de Interações entre Papéis* da fase de Análise de Domínio, uma vez que nesse modelo é especificado o conhecimento trocado pelos papéis nas suas interações e possivelmente algum *Modelo do Conhecimento da Sociedade Multiagente* desenvolvidos pela metodologia MAAEM que seja do mesmo domínio de problema da família de aplicações que estiver sendo desenvolvida. Ao final dessa tarefa, obtém-se um *Modelo do Conhecimento da Sociedade Multiagente*, contendo o conhecimento compartilhado pelos agentes da sociedade. O próximo passo, é a modelagem da sociedade multiagente, onde são definidos os agentes da sociedade. Esse modelo é baseado no Modelo de Papéis da fase de Análise de Domínio, uma vez que os papéis são mapeados para agentes em função da semelhança de responsabilidades e de seus requisitos não-funcionais, e possivelmente de algum *Modelo da Sociedade Multiagente* desenvolvidos pela metodologia MAAEM. A seguir é realizada a tarefa de modelagem de interações entre agentes, onde são definidas as iterações que o agente deverá realizar para atingir o objetivo geral da família de aplicações. Esse modelo é baseado no Modelo da Sociedade Multiagente, a partir do qual são obtidos os agentes da sociedade. Ao final dessa tarefa obtem-se o Modelo de Interações entre Agentes, onde estarão definidas as interações dos agentes para atingir o objetivo da família de aplicações. A próxima tarefa é a modelagem dos mecanismos de cooperação e coordenação, onde é definido como os agentes irão cooperar e estar organizados a fim de atingir os objetivos da família de aplicações. Ao final dessa tarefa, obtém-se um Modelo de Cooperação e Coordenação da Sociedade Multiagente e finaliza-se a tarefa de

Projeto Arquitetural. A seguir é dado início a tarefa de Projeto do Agente, que é divida nas subtarefas de modelagem de conhecimento do agente e modelagem das ações do agente que produzem, respectivamente, um Modelo de Conhecimento do Agente e um Modelo de Ações do Agente para cada agente da sociedade. Na modelagem do conhecimento do agente, é definido o conhecimento particular de cada agente da sociedade e na modelagem das ações do agente são definidas as ações que permitiram aos agentes realizar as suas responsabilidades.

Os papéis do processo definidos para as tarefas da fase de Projeto de Domínio são “Domain Designer” e “System Analyst”, ilustrados na Figura 17. As tarefas desta fase são realizadas em conjunto pelos dois papéis do processo. O papel do processo “Domain Designer” é o principal responsável pela realização e coordenação das tarefas, enquanto que o “System Analyst” o auxilia através do conhecimento que possui sobre os requisitos da família.

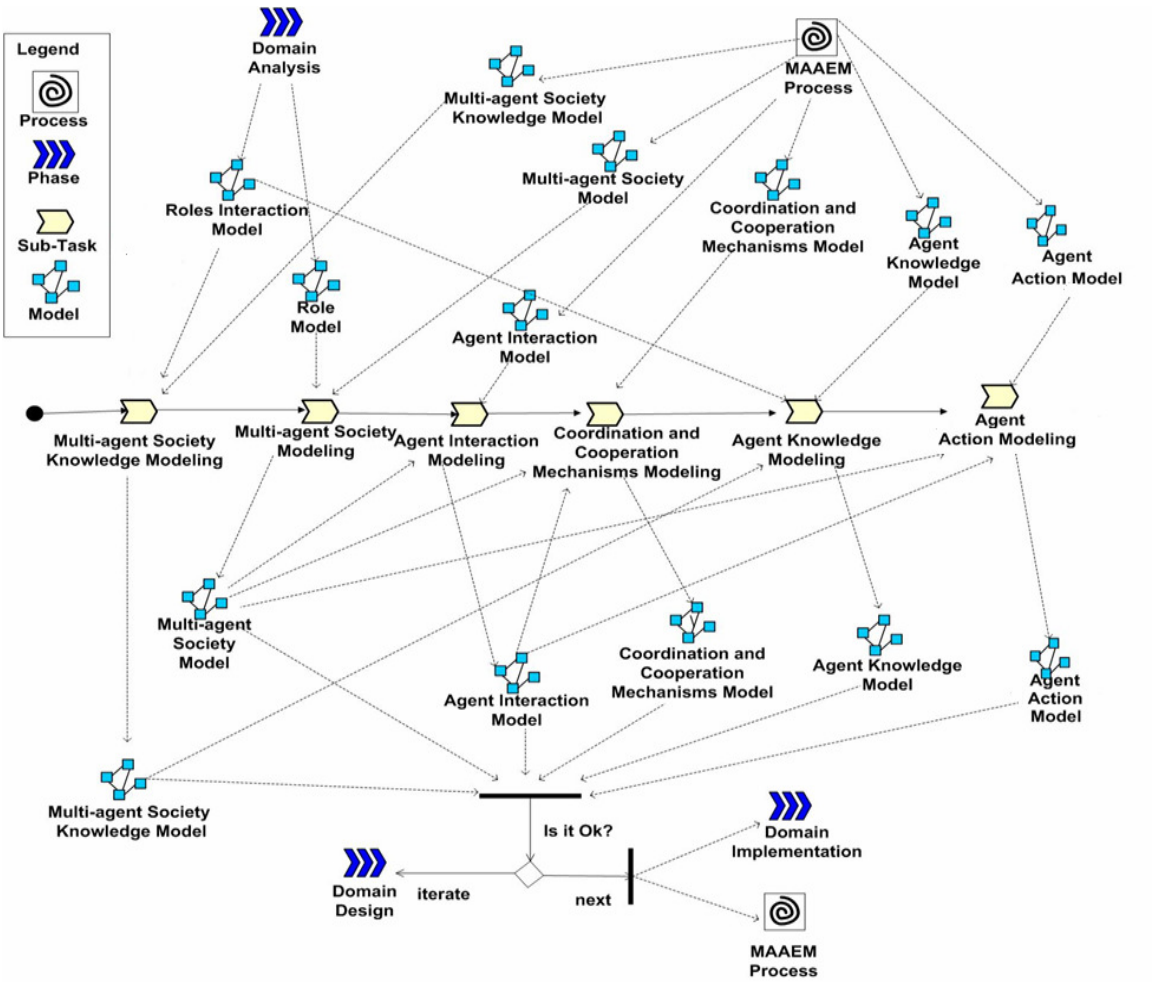

Figura 16. Subciclo de vida referente à fase de Projeto de Domínio

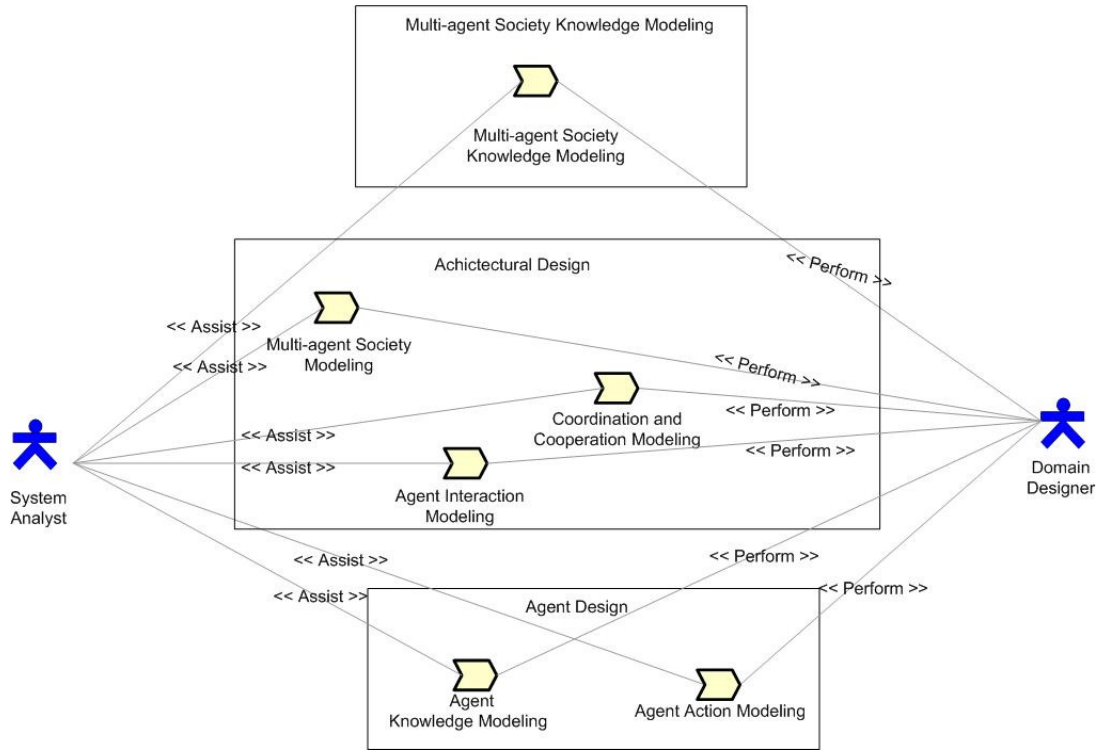

Figura 17 - Papéis do Processo da Fase de Projeto Domínio

3.1.3 A Fase de Implementação de Domínio

Na fase de Implementação de Domínio, os conceitos de projeto são mapeados para modelos específicos a uma plataforma de implementação de agentes. Para isso, tem-se adotado os frameworks JADE (BELLIFEMINE et al., 2003) e JESS (FRIEDMAN-HILL et al., 2003). Os modelos produzidos no subciclo de vida da fase de implementação de agentes são modelados de acordo com as diretrizes da técnica DIMAS.

O subciclo de vida da fase de Implementação de Domínio, ilustrado na Figura 18, inicia-se com a realização das tarefas de modelagem de comportamentos, onde as ações dos agentes e as responsabilidades são mapeadas do modelo de ações do agente para o modelo de comportamentos do agente seguindo os conceitos específicos de uma plataforma de implementação de agentes como JADE. A seguir é realizada modelagem de atos de comunicação, onde as interações identificadas no modelo de interações entre agentes são mapeadas para uma linguagem de comunicação específica como a FIPA-ACL (FIPA, 2008). Por fim, a tarefa de implementação dos agentes é realizada, onde a partir dos modelos de

Comportamentos do Agente e do Modelo de Atos de Comunicação os agentes são codificados. Parte dessa codificação pode ser gerado automaticamente a partir desses dois modelos.

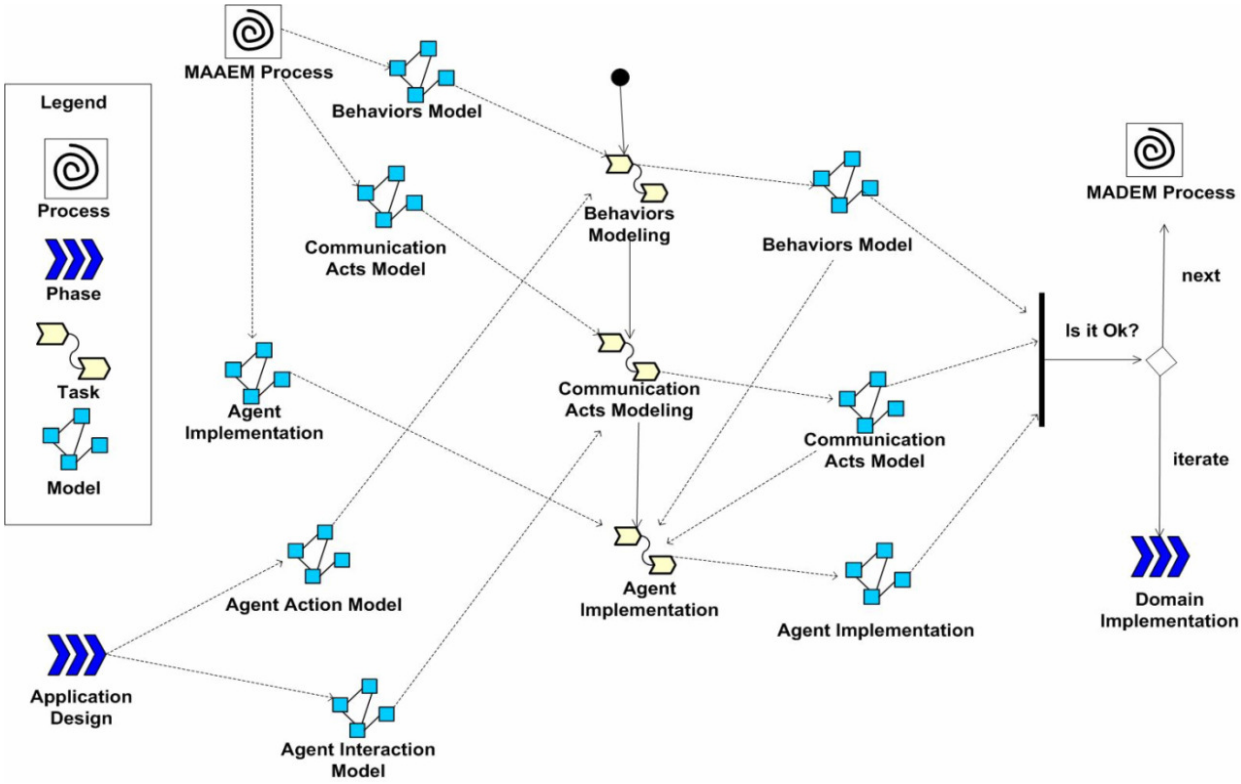

Figura 18. Subciclo de vida referente à fase de Implementação de Domínio

Os papel do processo “Programmer”, ilustrado na Figura 19, é o responsável pela realização das tarefas da fase de Implementação de Domínio e é supervisionado pelo papel “Domain Designer”.

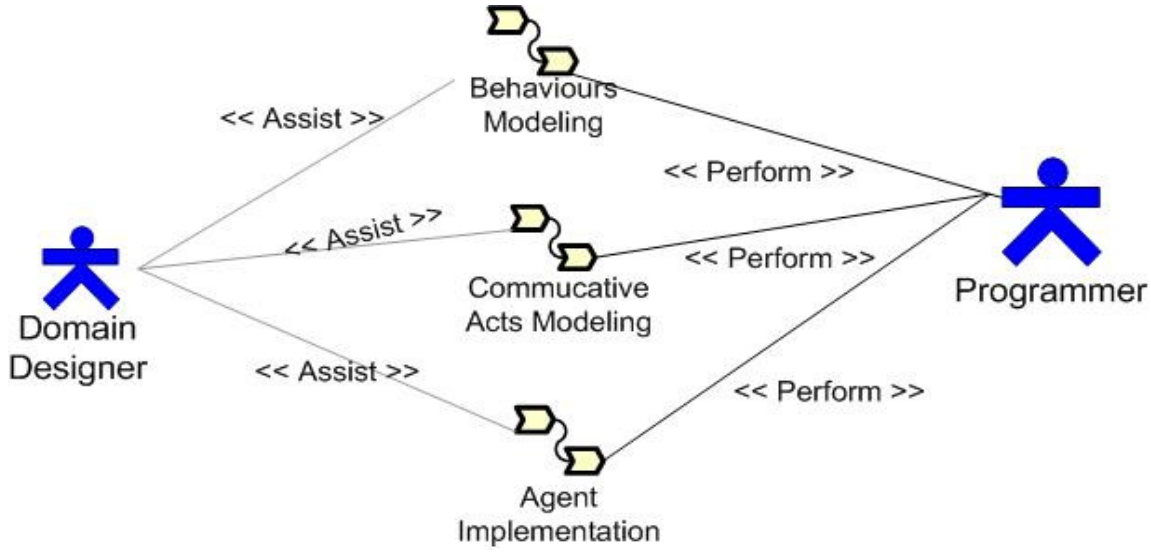

Figura 19 - Papéis do Processo da Fase de Implementação do Domínio

3.1.4 A Fase de Engenharia dos Requisitos da Aplicação

No subciclo de vida da fase de Engenharia dos Requisitos da Aplicação os requisitos de uma aplicação particular são identificados. Para isso, são reusados modelos desenvolvidos na Engenharia de Domínio, através da modelagem de conceitos, objetivos, papéis, interação entre papéis e prototipação da interface com o usuário seguindo as diretrizes da técnica SRAMO.

O subciclo da fase de Engenharia dos Requisitos da Aplicação, ilustrado na Figura 20, inicia-se com a Modelagem de Conceitos, na qual é realizada uma "tormenta de idéias" sobre os conceitos do domínio e estes são representados em uma rede semântica. A seguir é feita a Modelagem de Objetivos. Nessa tarefa, primeiro busca-se Modelos de Objetivos preexistentes na ONTORMAS, a fim de tornar essa tarefa mais produtiva. Se não houver modelos disponíveis para reúso, então se definem os objetivos geral e específicos da aplicação multiagente e as responsabilidades necessárias para alcançá-los através de um documento de requisitos da aplicação. Ao final desta tarefa o Modelo de Objetivos será produzido. O próximo passo é reusar Modelos de Papéis preexistentes que estejam associados aos objetivos específicos presentes no Modelo de Objetivos, através das responsabilidades que compartilham. Caso não haja Modelos de Papéis disponíveis para reúso, então se realiza a Modelagem de Papéis, a partir de responsabilidades identificadas no Modelo de Objetivos que são associadas a papéis (cada responsabilidade é associada a um papel), é definido ainda o conhecimento requerido para realização dessa responsabilidade, as entidades externas com as quais é necessário interagir e as suas pré e pós condições. Na tarefa seguinte, é realizada a modelagem de interações entre papéis. Nessa tarefa são reusados Modelos de Interações entre Papéis desenvolvidos no processo da Engenharia de Domínio que estejam associados as mesmas responsabilidades definidas no Modelo de Objetivos ou, no caso de não haver modelos disponíveis para reúso, esse modelo é construído identificando-se as necessidades de comunicação dos papéis definidos no Modelo de Papéis para alcançar os objetivos específicos da aplicação. Por último, é realizada a prototipação da interface com o usuário, onde é possível reusar protótipos de interface genéricos e especializar para a aplicação específica ou defini-la a partir do Modelo de Interações entre Papéis.

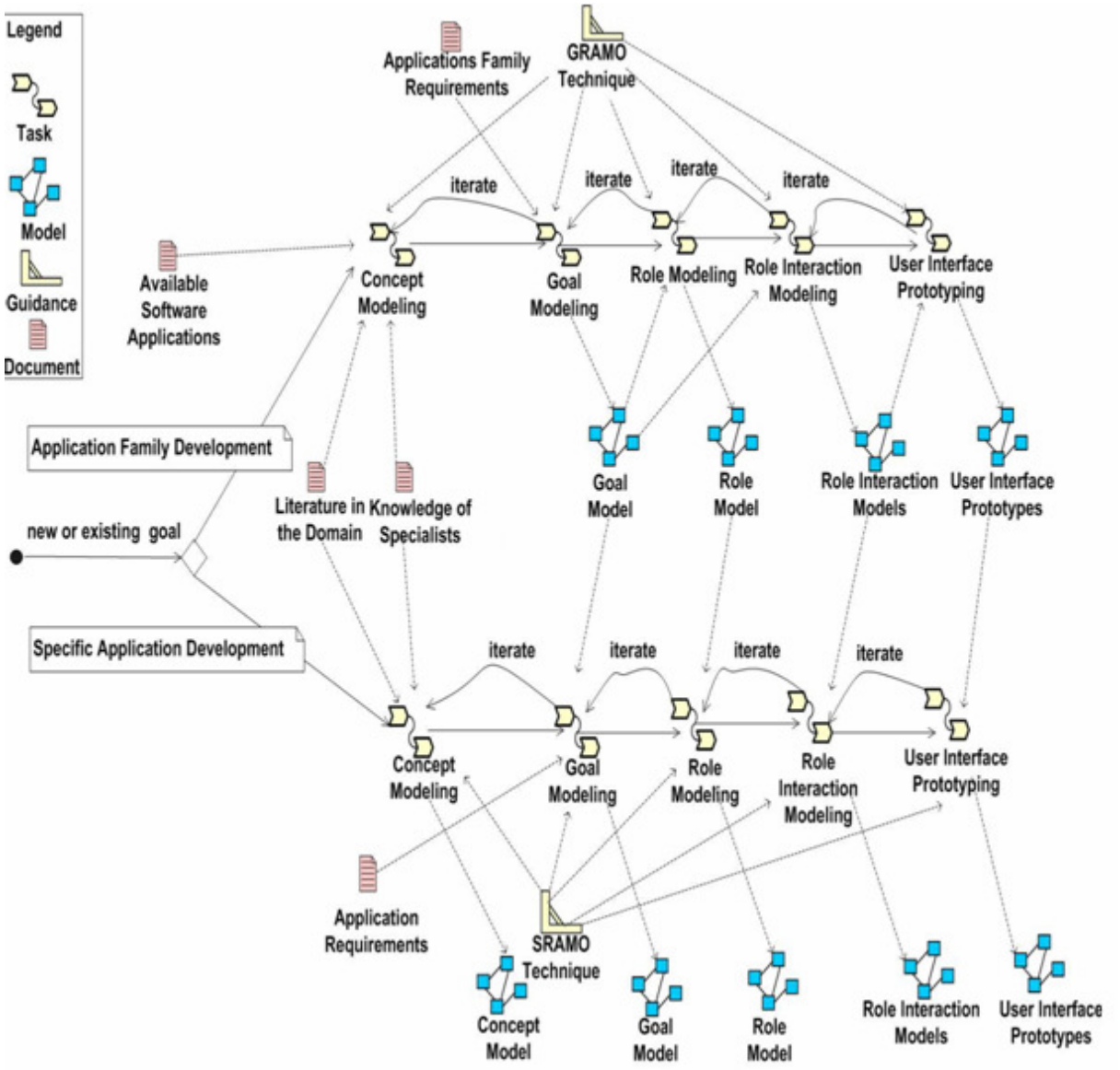

Figura 20. Subciclo de vida referente à fase de Engenharia dos Requisitos da Aplicação

3.1.5 A Fase de Projeto da Aplicação

Na fase de Projeto da Aplicação, o desenvolvedor poderá reutilizar soluções de projeto relacionadas a uma família de aplicações e adaptá-la aos requisitos específicos da aplicação em desenvolvimento. Esta fase consiste em realizar o Projeto Arquitetural da Sociedade Multiagente, incluindo seus mecanismos de coordenação e cooperação e o Projeto do Agente, onde é feito o projeto interno de cada agente da sociedade, modelando sua estrutura de conhecimento e seus comportamentos, seguindo as diretrizes da técnica ADEMAS.

O subciclo da fase de Projeto da Aplicação é ilustrado na Figura 8. Ele inicia-se com a realização da subtarefa de modelagem do conhecimento da sociedade multiagente, onde o conhecimento compartilhado pelos agentes da sociedade é representado em uma rede semântica, gerando o Modelo de Conhecimento da Sociedade Multiagente. A outra possibilidade é o reúso de Modelos do Conhecimento da Sociedade Multiagente desenvolvido no subprocesso da Engenharia de Domínio. O próximo passo é a modelagem da sociedade multiagente. Aqui são definidos os agentes da sociedade a partir dos papéis definidos no Modelo de Papéis da fase de Análise da Aplicação ou se já houver Modelos da Sociedade Multiagente desenvolvidos no subprocesso da Engenharia de Domínio da mesma família esse é reusado. A seguir, a modelagem de interações entre agentes é realizada, onde as interações dos agentes da sociedade são definidas. Aqui também podem ser reusados modelos preexistentes. A próxima tarefa é a modelagem dos mecanismos de cooperação e coordenação, onde é definido como os agentes irão cooperar e estar organizados na sociedade. Para isso, usam-se padrões já popularizados na comunidade, como o padrão de mercado, mestre-escravo, camadas, etc (GIRARDI, 2004). Modelos pré-existentes também podem ser reusados nesse passo. Após a construção do Modelo dos Mecanismos de Cooperação e Coordenação finaliza-se a tarefa de projeto arquitetural e dá-se inicio a tarefa de projeto do agente.

No projeto do agente a modelagem de conhecimento é realizada, definindo-se o conhecimento particular de cada agente da sociedade. A seguir, a modelagem das ações do agente é realizada, onde são definidas as ações que o agente terá para realizar suas responsabilidades. Aqui também podem ser reusados Modelos de Conhecimento e Modelos de Ações desenvolvidos no subprocesso da Engenharia de Domínio.

Ao final da fase de Projeto da Aplicação, conforme o losango da Figura 21, pode-se passar novamente para a fase de Projeto de Domínio ou inicia-se um novo ciclo para realizar um refinamento dos modelos a fim de adicionar novos requisitos ou alguma eventual correção.

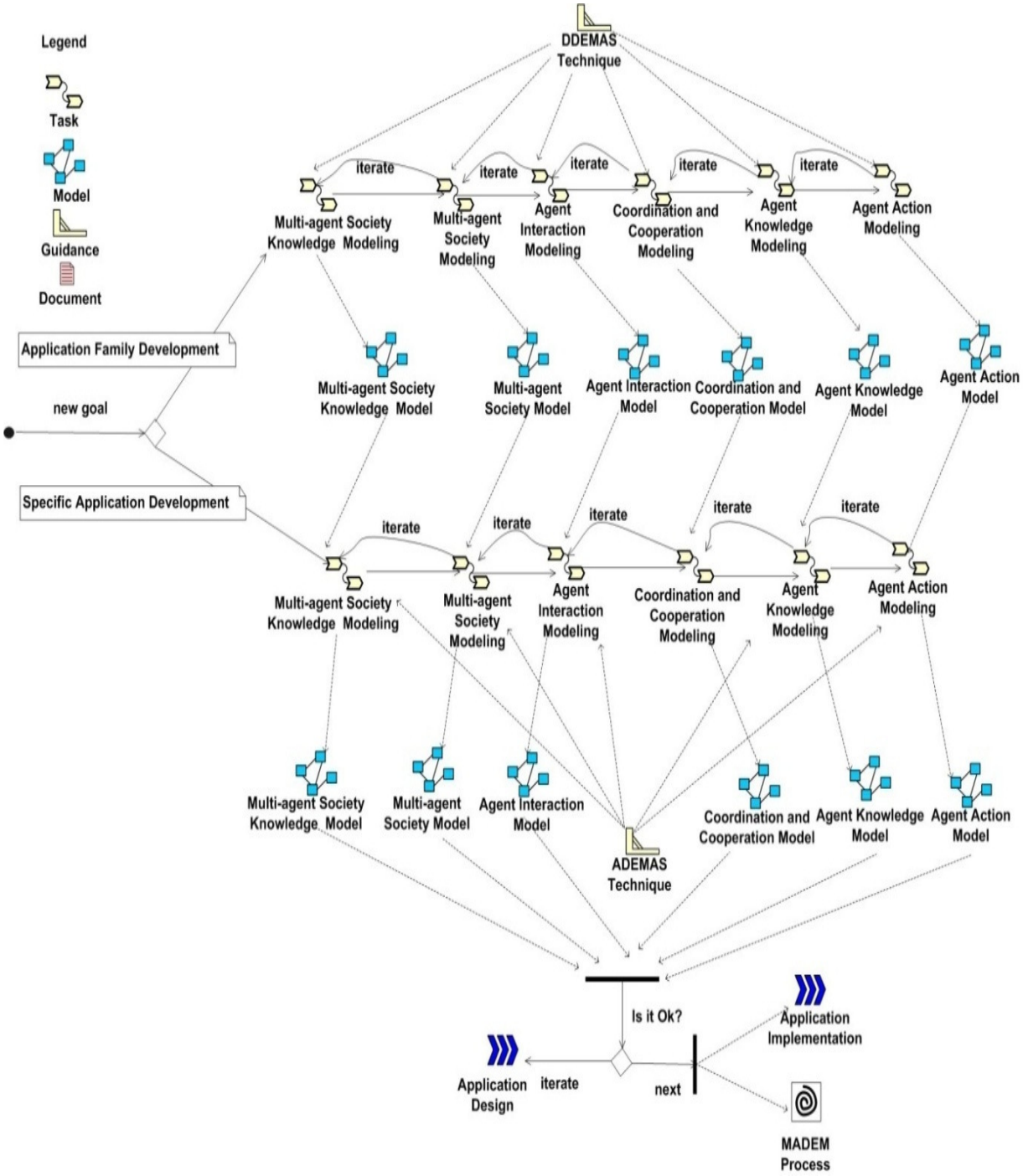

Figura 21. Subciclo de vida referente à fase de Projeto da Aplicação

3.1.6 A Fase de Implementação da Aplicação

No subciclo da fase de Implementação, ilustrado na Figura 22, são construídos o Modelo de Comportamentos e o Modelo de Atos de Comunicação. Com base nesses dois modelos é realizada a codificação da aplicação multiagente. No Modelo de Comportamentos são especificados os comportamentos que os

agentes terão e no Modelo de Comunicação as interações que eles precisam realizar para cumprir suas responsabilidades. Esses modelos são construídos de forma semelhante aos da fase de Implementação de Domínio e o papel do processo é o mesmo, a diferença é que aqui é uma aplicação específica é implementada a partir do reúso de agentes de software genéricos pertencentes a uma família de aplicações multiagente.

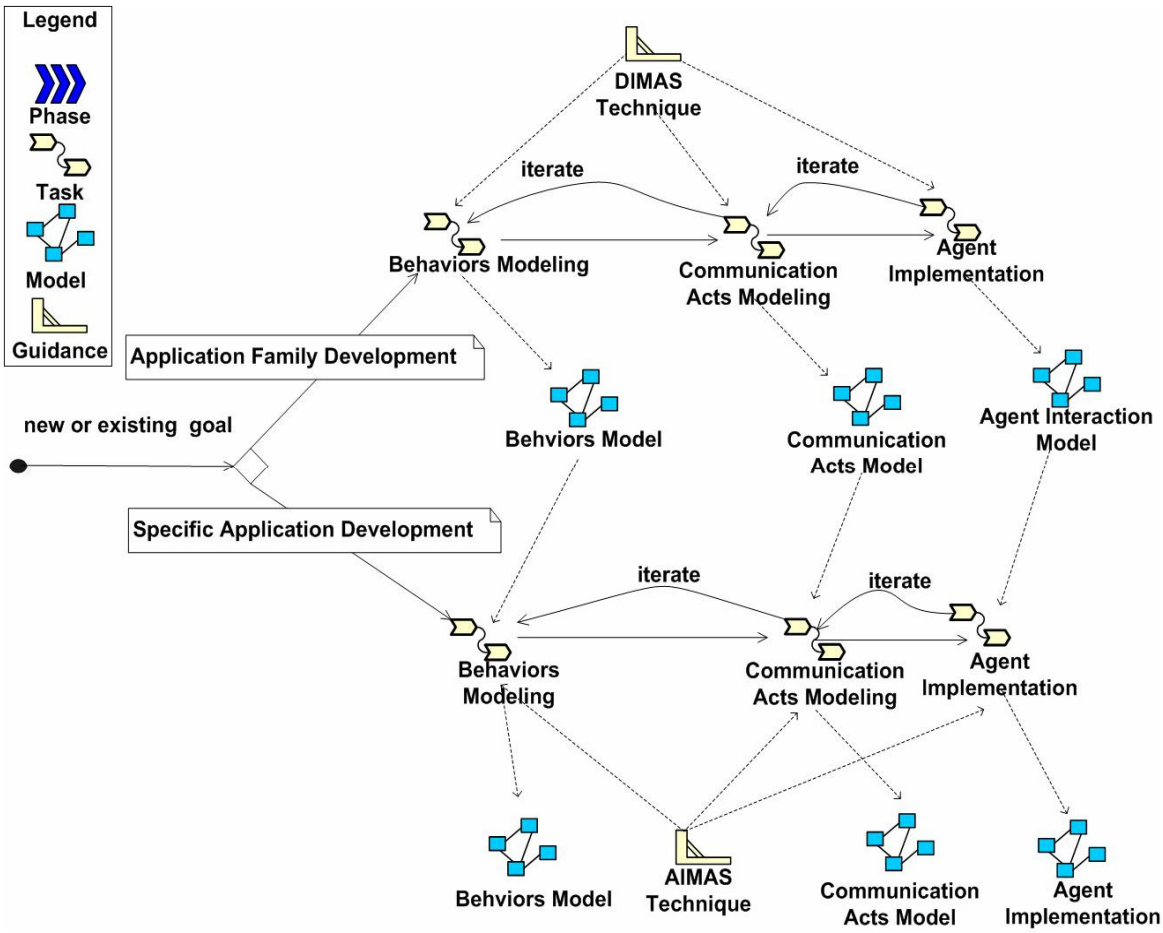

Figura 22. Subciclo de vida referente à fase de Implementação da Aplicação

3.2 A Linguagem de Modelagem MADAE-ML

MADAE-ML (“Multi-agent Domain and Application Engineering Modeling Language”) é uma linguagem visual baseada em ontologias para modelagem de sistemas multiagente (LEITE, 2009). Essa linguagem é apoiada pela ferramenta ONTORMAS e para sua utilização no processo MADAE-Pro. Na Figura 23, uma rede semântica é ilustrada com os principais conceitos da linguagem MADAE-ML

utilizados para realizar a modelagem de sistemas multiagente de acordo com as metodologias MADAEM e MAAEM.

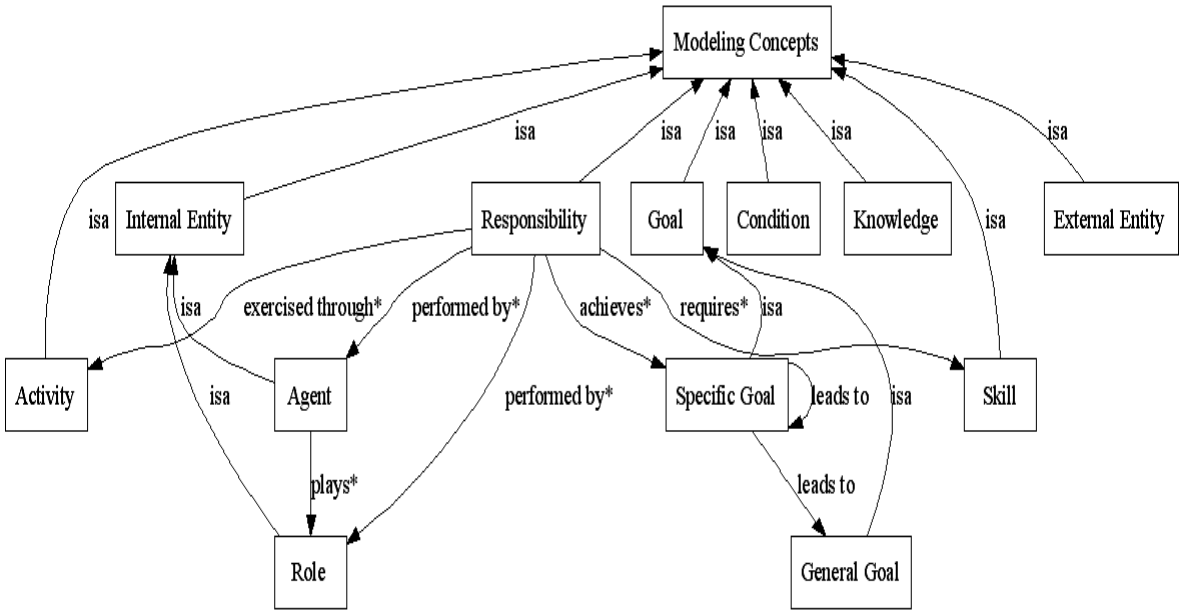

Figura 23 - Conceitos básicos de Modelagem das metodologias MADAEM e MAAEM

Na Tabela 6 os conceitos de modelagem da linguagem MADAE-ML são listados. Um determinado agrupamento desses conceitos constitui um modelo produzido no MADAE-Pro. Na coluna "*nome*" é definido o nome do conceito, na coluna "*descrição*" o conceito é definido, na coluna "*usado por*" são listados os modelos dos quais um determinado conceito pode fazer parte.

Tabela 6 - Conceitos da Linguagem MADAE-ML

Nome	Descrição	Usado por	Símbolo
Domain Concept	Representa qualquer conceito do domínio.	Concepts Model	
Responsibility	Representa uma responsabilidade a ser desempenhada por um Papel e/ou Agente.	Goal Model Role Model Multi-agent Society Model	
General Goal	Representa o Objetivo Geral do Sistema Multiagente.	Goal Model	
Specific Goal	Representa um Objetivo Específico do Sistema Multiagente.	Goal Model	

External Entity	Representa uma entidade externa ao Sistema com a qual o Papel ou Agente interage.	Goal Model Role Model Role Interaction Model Multi-agent Society Model Agent Interaction Model Coordination and Cooperation Model	
Role	Representa um Papel, uma entidade que terá uma ou mais responsabilidades necessárias para alcançar um objetivo específico.	Role Model Role Interaction Model	
Knowledge	Representa conhecimento de domínio.	Role Model Multi-agent Society Model Multi-agent Society Knowledge Model Agent Knowledge Model Agent Action Model	
Condition	Representa uma condição a ser satisfeita, previamente ou após a execução de uma responsabilidade.	Role Model Multi-agent Society Model Agent Action Model	
Action	Representa uma ação a ser executada por um agente para o cumprimento de sua Responsabilidade.	Agent Action Model	
Skill	São habilidades ou conhecimentos necessários para a execução de alguma responsabilidade ou atividade específica.	Role Model Multi-agent Society Model	
Agent	É uma entidade de software com autonomia, responsável pela obtenção dos objetivos do Sistema através de uma ou mais tarefas.	Multi-agent Society Model Agent Interaction Model Agent Action Model	
Organizational and Process Pattern	Representa padrões de processos e organizações.	Pattern System	

Analysis Pattern	Representa um Padrão de Análise	Pattern System	
Architectural Pattern	Representa um Padrão Arquitetural	Pattern System	
JADE Behaviour	Representa um comportamento de um agente JADE.	Model of Behaviours, Communicative Acts Model	
Method JADE Behaviour	Representa um método de um comportamento de um agente JADE.	Model of Behaviours, Communicative Acts Model	
FIPA-ACL Message	Representa uma mensagem com uma determinada performativa da linguagem de comunicação entre agentes FIPA-ACL.	Communicative Acts Model	

3.3 A metodologia MADEM

A MADEM é uma metodologia para o desenvolvimento de artefatos de software reusáveis integrantes de uma família de aplicações multiagente. Uma família de aplicações é definida como um conjunto de aplicações de software similares que compartilham algumas características em comum e outras específicas.

Para a especificação do domínio de problema a ser resolvido, a MADEM orienta a realização da modelagem de objetivos, papéis, interações de entidades de uma organização e prototipação da interface com usuário. As entidades possuem conhecimento e o usam para exibir comportamento autônomo. Uma organização é composta de entidades com objetivos geral e específicos que estabelecem o que a organização pretende alcançar. A execução dos objetivos específicos permite alcançar o objetivo geral da organização. Por exemplo, um sistema de informação pode ter o objetivo “satisfazer as necessidades de informação de uma organização” e os objetivos específicos de “satisfazer as necessidades de informação dinâmica ou em longo prazo”. Os objetivos específicos são alcançados através da execução de responsabilidades que as entidades têm para a execução de papéis com certo grau de autonomia.

Os papéis têm destrezas sobre uma ou um conjunto de técnicas que suportam a execução de responsabilidades. Pré-condições e pós-condições podem necessitar serem satisfeitas para ou após a execução de uma responsabilidade. Conhecimento pode ser requerido e produzido na realização de uma responsabilidade. Por exemplo, uma entidade pode atuar como o papel "recuperador" com a responsabilidade de executar atividades para satisfazer as necessidades de informações dinâmicas de uma organização. Outra entidade pode atuar como o papel de "filtrador" com a responsabilidade de executar atividades para satisfazer as necessidades de informações em longo prazo das organizações.

Destrezas podem ser, por exemplo, as regras da organização que as entidades conhecem para acessar e estruturar suas fontes de informação.

Algumas vezes, as entidades têm que se comunicar com outras entidades internas ou externas para cooperar na execução de uma responsabilidade. Por exemplo, uma entidade atuando como o papel de "filtrador" pode necessitar interagir com um usuário (entidade externa) para observar seu comportamento e requisitar a inferência do seu perfil de interesses de informação.

Um resumo das tarefas realizadas, dos insumos requeridos e dos produtos obtidos em cada fase de desenvolvimento de uma aplicação multiagente seguindo as diretrizes da metodologia MADEM é descrito na Tabela 8.

A fase de Análise de Domínio aborda a construção de um Modelo de Domínio, no qual são especificados os requisitos atuais e futuros de uma família de aplicações em um determinado domínio, considerando o conhecimento do domínio e as experiências de desenvolvimento extraídas a partir de especialistas e de aplicações já desenvolvidas nesse domínio. Na fase de Análise de Domínio, a técnica GRAMO recomenda a realização das seguintes tarefas: Modelagem de Conceitos, onde é realizada uma "tormenta de idéias" sobre os conceitos do domínio e seus relacionamentos; Modelagem de Objetivos, na qual são identificados e representados os objetivos de uma família de aplicações, as entidades externas com as quais coopera e as responsabilidades necessárias para alcançá-los; Modelagem de Papéis, onde são associadas as responsabilidades identificadas na tarefa de Modelagem de Objetivos aos papéis que serão encarregados delas; Modelagem de Interações entre Papéis, onde é identificado como as entidades externas e internas cooperam para alcançar um objetivo específico e a Prototipação da Interface com o

Usuário, cujo objetivo é identificar as interações dos usuários com o sistema e construir um protótipo da interface. O produto desta fase é um Modelo de Domínio.

Tabela 7 - Fases, Insumos, Tarefas e Produtos da metodologia MADEM

<table>
<tr><th>Fases</th><th>Insumos</th><th colspan="2">Tarefas</th><th colspan="2">Produtos</th><th></th></tr>
<tr><td rowspan="8">Análise de Domínio</td><td>Conhecimento de Especialistas, Literatura no Domínio, Aplicações de Software Existentes.</td><td colspan="2">Modelagem de Conceitos</td><td colspan="2">Modelo de Conceitos</td><td rowspan="8">Modelo de Domínio</td></tr>
<tr><td>Requisitos da família de aplicações</td><td colspan="2">Modelagem de Objetivos</td><td colspan="2">Modelo de Objetivos</td></tr>
<tr><td>Modelo de Objetivos</td><td colspan="2">Modelagem de Papéis</td><td colspan="2">Modelo de Papéis</td></tr>
<tr><td>Modelo de Objetivos</td><td colspan="2" rowspan="2">Modelagem de Interações entre Papéis</td><td colspan="2" rowspan="2">Modelo de Interações entre Papéis</td></tr>
<tr><td>Modelo de Papéis</td></tr>
<tr><td>Modelo de Interações entre Papéis</td><td colspan="2" rowspan="2">Prototipagem da Interface com o usuário</td><td colspan="2" rowspan="2">Protótipo da Interface com o usuário</td></tr>
<tr><td>Modelo de Papéis</td></tr>
<tr><td rowspan="13">Projeto de Domínio</td><td>Modelo de Conceitos</td><td rowspan="7">Modelagem Arquitetural</td><td>Modelagem do Conhecimento da Sociedade Multiagente</td><td>Modelo do Conhecimento da Sociedade Multiagente</td><td rowspan="7">Modelo Arquitetural</td><td rowspan="13">Modelo do Framework Multiagente</td></tr>
<tr><td>Modelo de Interações entre Papéis</td><td>Modelagem da Sociedade de Agentes</td><td>Modelo da Sociedade Multiagente</td></tr>
<tr><td>Modelo de Papéis</td><td>Modelagem das Interações entre Agentes</td><td>Modelo de Interações entre Agentes</td></tr>
<tr><td>Modelo da Sociedade Multiagente</td><td rowspan="4">Modelagem dos Mecanismos de Cooperação e Coordenação</td><td rowspan="4">Modelo dos Mecanismos de Cooperação e Coordenação</td></tr>
<tr><td>Modelo de Objetivos (Requisitos Não-Funcionais)</td></tr>
<tr><td>Modelo da Sociedade Multiagente</td></tr>
<tr><td>Modelo das Interações entre Agentes</td></tr>
<tr><td>Modelo de Interações entre Papéis</td><td rowspan="4">Projeto do Agente</td><td>Modelagem do Conhecimento do Agente</td><td>Modelos do Conhecimento do Agente</td><td rowspan="4">Modelos do Agente</td></tr>
<tr><td>Modelo do Conhecimento da Sociedade Multiagente</td></tr>
<tr><td>Modelo da Sociedade Multiagente</td><td rowspan="2">Modelagem das Ações dos Agentes</td><td rowspan="2">Modelos de Ações do Agente</td></tr>
<tr><td>Modelo de Interações entre Agentes</td></tr>
<tr><td rowspan="4">Implementação de Domínio</td><td>Modelos de Ações</td><td colspan="2">Modelagem de Comportamentos</td><td colspan="2">Modelo de Comportamentos</td><td rowspan="2">Modelo de Implementação da Sociedade Multiagente</td></tr>
<tr><td>Modelo de Interações entre Agentes</td><td colspan="2">Modelagem de Atos de Comunicação</td><td colspan="2">Modelo de Atos de Comunicação</td></tr>
<tr><td>Modelo de Comportamentos</td><td colspan="2" rowspan="2">Implementação dos Agentes</td><td colspan="3" rowspan="2">Agentes de Software Executáveis</td></tr>
<tr><td>Modelo de Atos de Comunicação</td></tr>
<tr><td>Extração e Representação de Padrões</td><td>Experiências de Desenvolvimento e outras fontes disponíveis</td><td colspan="2">Extração e Representação de Padrões</td><td colspan="3">Padrões de Sistema e de Software</td></tr>
</table>

Na Engenharia de Domínio, a modelagem de variabilidades especifica as características comuns e variáveis da família de aplicações. As características comuns são aquelas compartilhadas por todas as aplicações da família, enquanto que as características variáveis são as que se diferenciam em algumas aplicações da família. A Modelagem de Variabilidades no processo MADAE-Pro, é baseada na especificação de pontos de variação e variantes. Um ponto de variação representa

um conceito sujeito à variação e uma variante representa grupos de variações alternativas ou opcionais desse conceito. Na Figura 24, é ilustrado o metamodelo de variabilidades, onde temos os conceitos já definidos anteriormente de “Variation Point”, “Alternative Group”, “Optional Group”, “Variant” e “Modeling Concepts”.

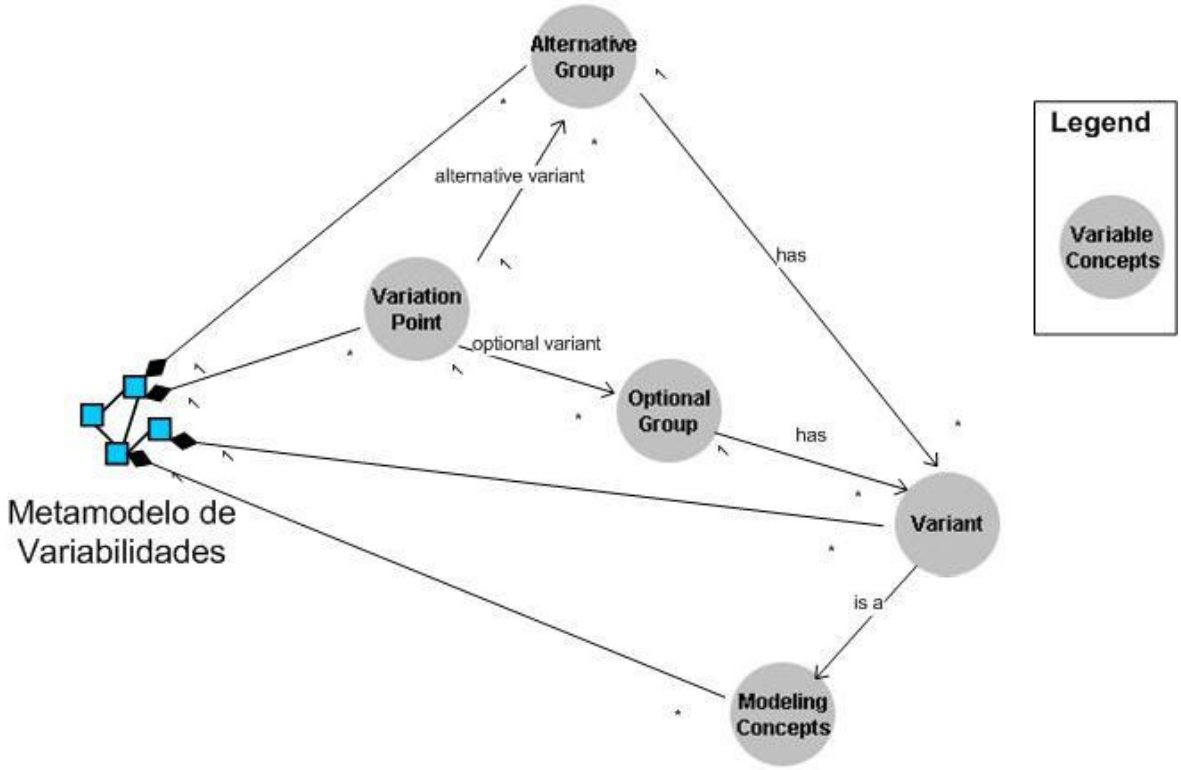

Figura 24 - Metamodelo de Variabilidades

3.4 A metodologia MAAEM

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 a construção de uma aplicação. Na fase de análise dos requisitos, pretende-se identificar e especificar os requisitos de uma determinada aplicação, partindo de modelos de domínio, elaborados na fase correspondente da Engenharia de Domínio Multiagente.

Desse modo, a tarefa central do desenvolvedor é reusar um conjunto de requisitos da família em um domínio. Tais requisitos específicos da aplicação,

quando provenientes da família, são levantados a partir da seleção dentre os requisitos comuns ou variáveis no domínio. Um resumo das tarefas realizadas, dos insumos requeridos e dos produtos obtidos em cada fase de desenvolvimento de uma aplicação multiagente seguindo as diretrizes da metodologia MAAEM está descrito na Tabela 8.

Tabela 8 - Fases, Insumos, Tarefas e Produtos da metodologia MAAEM

Fases	Insumos	Tarefas		Produtos		
Engenharia dos Requisitos da Aplicação	Modelo de Conceitos (MADEM)	Modelagem de Conceitos		Modelo de Conceitos		Especificação dos Requisitos da Aplicação
	Conhecimento de Especialistas, Literatura no Domínio, Aplicações de Software Existentes					
	Modelo de Objetivos (MADEM)	Modelagem de Objetivos		Modelo de Objetivos		
	Requisitos da família de aplicações					
	Modelo de Papéis (MADEM)	Modelagem de Papéis		Modelo de Papéis		
	Modelo de Objetivos					
	Modelo de Interações entre Papéis (MADEM)	Modelagem de Interações entre Papéis		Modelo de Interações entre Papéis		
	Modelo de Objetivos					
	Modelo de Papéis					
	Protótipo da Interface com o usuário (MADEM)	Prototipagem da Interface com o usuário		Protótipo da Interface com o usuário		
	Modelo de Papéis					
	Modelo de Interações entre Papéis					
Projeto da Aplicação	Modelo do Conhecimento da Sociedade Multiagente (MADEM)	Modelagem Arquitetural	Modelagem do Conhecimento da Sociedade Multiagente	Modelo do Conhecimento da Sociedade Multiagente	Modelo Arquitetural	Arquitetura da Aplicação
	Modelo de Conceitos					
	Modelo de Interações entre Papéis					
	Modelo da Sociedade Multiagente (MADEM)		Modelagem da Sociedade de Agentes	Modelo da Sociedade Multiagente		
	Modelo de Papéis					
	Modelo de Interações entre Agentes (MADEM)		Modelagem das Interações entre Agentes	Modelo de Interações entre Agentes		
	Modelo da Sociedade Multiagente					
	Modelo dos Mecanismos de Cooperação e Coordenação (MADEM)		Modelagem dos Mecanismos de Cooperação e Coordenação	Modelo dos Mecanismos de Cooperação e Coordenação		
	Modelo de Objetivos (Requisitos Não-Funcionais)					
	Modelo da Sociedade Multiagente					
	Modelo das Interações entre Agentes					
	Modelos do Conhecimento do Agente (MADEM)	Projeto do Agente	Modelagem do Conhecimento do Agente	Modelos do Conhecimento do Agente	Modelos do Agente	
	Modelo de Interações entre Papéis					
	Modelo do Conhecimento da Sociedade Multiagente					
	Modelos das Ações do Agente (MADEM)		Modelagem das Ações dos Agentes	Modelos das Ações do Agente		
	Modelo da Sociedade Multiagente					
	Modelo de Interações entre Agentes					
Implementação da Aplicação	Modelo de Comportamentos (MADEM)	Modelagem de Comportamentos		Modelo de Comportamentos		Modelo de Implementação da Aplicação
	Modelos de Ações					
	Modelo de Atos de Comunicação (MADEM)	Modelagem de Atos de Comunicação		Modelo de Atos de Comunicação		
	Modelo de Interações entre Agentes					
	Agentes de Software Executáveis (MADEM)	Implementação dos Agentes		Agentes de Software Executáveis		
	Modelo de Comportamentos					
	Modelo de Atos de Comunicação					

4 MADAE-IDE – UM AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE BASEADO NO CONHECIMENTO PARA O REUSO COMPOSICIONAL NO DESENVOLVIMENTO DE SISTEMAS MULTIAGENTE

4.1 Introdução

Várias metodologias (BRESCIANI et al., 2004), (CLYNCH e COLLIER, 2007), (LEITE et al., 2008) e alguns processos (COSSENTINO e POTTS, 2002), (CERNUZZI et al., 2005), (CERNUZZI e ZAMBONELLI, 2008) e (LEITE, 2009) para o desenvolvimento de software orientado a agentes têm sido propostos. MADAE-Pro ("Multi-agent Domain and Application Engineering Process") (LEITE, 2009) é um processo para o desenvolvimento e reuso de famílias de sistemas de software multiagente.

Além de prover suporte para o reuso no desenvolvimento de software multiagente através da integração dos conceitos da Engenharia de Domínio e Engenharia de Aplicação, MADAE-Pro é um processo baseado em conhecimento nos quais modelos de agentes e frameworks são representados como instâncias ontológicas. Assim, conceitos são semanticamente relacionados permitindo buscas e inferências eficazes facilitando o entendimento e reuso dos modelos durante o desenvolvimento de aplicações específicas num domínio. Os modelos baseados em ontologia do MADAE-Pro pode ser documentados, adaptados e integrados.

Apesar destas facilidades, as tarefas de modelagem do processo têm sido executadas manualmente apenas apoiadas pela linguagem gráfica e editor de modelos da ontologia, o que se mostrou trabalhoso e suscetível a erros tornando a base de conhecimento ONTORMAS, contida na ferramenta ONTORMAS-T geralmente inconsistente.

Para resolver este problema, utilizando-se da vantagem fornecida pelas relações semânticas entre os conceitos e modelos na base de conhecimento ONTORMAS, o ambiente MADAE-IDE (Multi-agent Domain and Application Engineering – Integrated Development Environment) foi desenvolvido (CAVALCANTE e GIRARDI, 2010).

Este capítulo apresenta o ambiente de desenvolvimento de sistemas multiagente MADAE-IDE, fundamentado no processo MADAE-Pro, automatizando-o e permitindo a modelagem de sistemas multiagente totalmente guiados e apoiados por uma única ferramenta. O ambiente MADAE-IDE, desenvolvido em Java, é distribuído como um plug-in do Protégé (GENNARI, 2002).

Inicialmente apresenta-se o projeto do ambiente MADAE-IDE, detalhando-se a interface com o usuário e o núcleo do ambiente. A seguir examina-se a implementação do ambiente. Ao fim do capítulo são apresentadas as considerações finais.

4.2 Projeto do ambiente MADAE-IDE

O ambiente MADAE-IDE foi projetado com base no processo MADAE-Pro e de discussões com as pessoas envolvidas no desenvolvimento deste. A integração de um processo de desenvolvimento de software a um ambiente de desenvolvimento difere dos ambientes tradicionais que incluem recursos e ferramentas para serem utilizados de forma geral, sem um processo específico. Um ambiente capaz de prover o desenvolvimento de um software multiagente, baseado no conhecimento do processo MADAE-Pro a partir da análise, projeto, implementação e reuso de artefatos é de fundamental importância para que o desenvolvimento de sistemas multiagente seja beneficiado com a automação das atividades citadas.

O ambiente MADAE-IDE assiste os desenvolvedores na aplicação do processo MADAE-Pro, automatizando total ou parcialmente suas tarefas de modelagem através de um sistema de regras que explora a representação semântica dos produtos de modelagem, instanciados na base de conhecimento ONTORMAS.

MADAE-IDE é estruturado como um sistema baseado em conhecimento (GIARRATANO e RILEY, 1989) e sua arquitetura é composta por uma interface com o usuário e um núcleo do ambiente, como mostra a Figura 25. A interface com o usuário consiste de um conjunto de assistentes de modelagem para guiar o desenvolvedor na execução de todas as fases de desenvolvimento do processo MADAE-Pro. O núcleo do ambiente é composto de uma base de conhecimento e um mecanismo de inferência. A base de conhecimento consiste da ontologia

ONTORMAS, um sistema de regras de inferência e uma memória de trabalho contendo um conjunto de fatos (cláusulas incondicionais) representando o conhecimento introduzido pelo desenvolvedor durante a interação com o ambiente. O sistema de regras de inferência consiste de regras para criação e transformação de produtos de modelagem de uma família de aplicações multiagente, seguindo os diferentes níveis de abstração das tarefas de modelagem do processo MADAE-Pro, representando-os como instâncias da ontologia ONTORMAS.

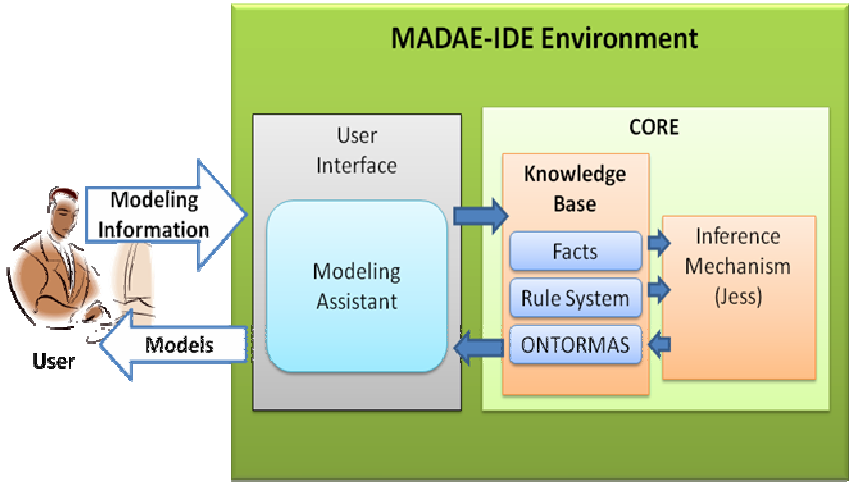

Figura 25 – Arquitetura do ambiente MADAE-IDE

4.2.1 Interface com o usuário

A interface com o usuário do ambiente MADAE-IDE possui duas finalidades: fornecer suporte amigável ao usuário, guiando-o pelas tarefas que necessitam ser realizadas durante a aplicação do processo MADAE-Pro e adquirir as informações para a modelagem do sistema. Cada atividade do processo MADAE-Pro possui um assistente que direcionará todos os passos do desenvolvedor na sua execução, diminuindo o risco deste em cometer erros.

O ambiente MADAE-IDE apoia o ciclo de vida iterativo e incremental do processo MADAE-Pro através da visualização e possível alteração de um modelo já produzido e que necessite ser refinado. Se necessário, é possível executar apenas parte do processo, no entanto, para se instanciar determinada fase do processo MADAE-Pro é necessária a execução das fases anteriores, pois o MADAE-Pro define uma seqüência de atividades que deve ser atendida.

4.2.2 Núcleo do ambiente

O sistema de regras de inferência é composto de regras de criação e regras de transformação. As informações de modelagem do sistema em desenvolvimento fornecidas pelo usuário são representadas em fatos da memória de trabalho no núcleo do ambiente. Estes fatos disparam regras em Jess contidas no núcleo do ambiente, que têm conhecimento do ciclo de vida, das fases, das tarefas, dos insumos e produtos do processo MADAE-Pro, podendo criar ou alterar suas instâncias de tarefas e produtos na ontologia ONTORMAS. Para que seja possível a utilização do motor de inferência Jess com a ontologia ONTORMAS utiliza-se o plugin JessTab (ERIKSSON, 2003). Cada pacote de regras atua em uma fase do processo MADAE-Pro. A Figura 26 detalha o núcleo do ambiente MADAE-IDE.

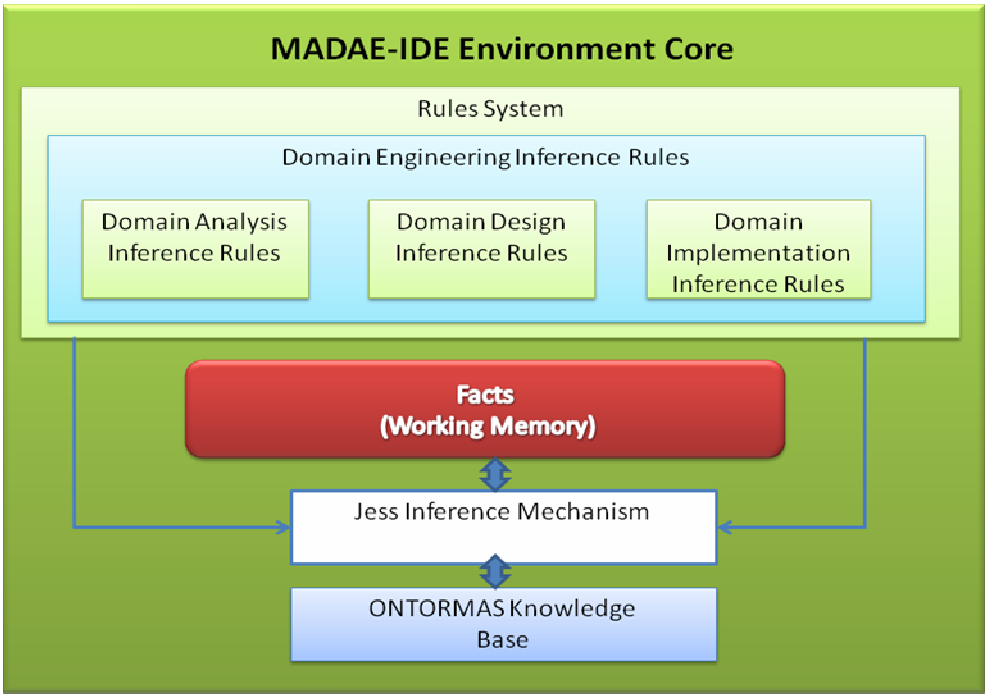

Figura 26 – Núcleo do ambiente MADAE-IDE

As regras de criação instanciam as classes que representam na ONTORMAS as fases, as tarefas, os produtos e os conceitos de modelagem definidos no processo MADAE-Pro com base nas informações de modelagem fornecidas pelo desenvolvedor na interação com o ambiente. Além disto, estas regras mantêm a consistência entre as fases, tarefas e produtos de modelagem

associando a cada fase de modelagem as suas tarefas e produtos que são instanciados.

4.2.2.1 Sistema de Regras de criação de modelos

O processo de criação (P_c) de um modelo alvo é definido como $P_c = (T, U, R_c)$, sendo o modelo alvo $T = MC_t \cup SR_t$, $MC_t \subseteq MC$, $SR_t \subseteq SR$. $MC_i \in MC$ é um conceito de modelagem e $SR_i \in SR$ é a um relacionamento semântico na base de conhecimento ONTORMAS. Por exemplo, na fase de análise de domínio, um modelo de objetivos deve ser criado, considerando os conceitos de modelagem e as relações semânticas na Figura 27: T = "Goal Model" = {"General Goal", "Specific Goal", "Responsibility", "External Entity", "Gets Information", "Leads To", "Achieved By"}. U é a informação de modelagem fornecida pelo desenvolvedor através do uso do MADAE-IDE utilizando o assistente de modelagem (ver exemplo na Figura 34) associado com a respectiva tarefa de modelagem. R_c é o conjunto de regras de criação para a geração do modelo alvo.

Como um exemplo, as regras em Jess da Listagem 1 são um subconjunto das regras de criação para geração das instâncias que representam a tarefa de modelagem de objetivos e o produto modelo de objetivos na ONTORMAS.

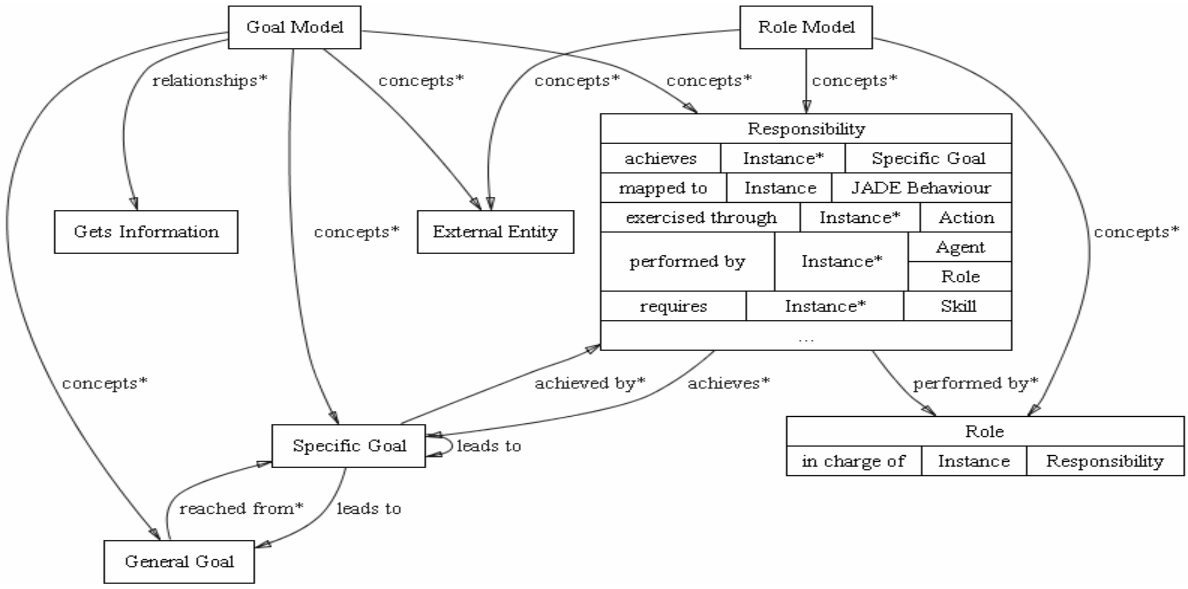

Figura 27 - Relações semânticas entre um Modelo de Objetivos e um Modelo de Papéis

```
(defrule GoalModelingTaskCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil))
  ?GoalModelingTask <-   (ModelingTask (task GoalModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Goal Modeling" (name (str-cat "Goal Modeling Of "
?*project*))))
  (modify ?GoalModelingTask (ref ?ref))
  (slot-insert$ ?DomainAnalysisTask subtasks 1 ?ref)
  (assert (ModelingProduct (product GoalModel) (name (str-cat "Goal Model Of " ?*project*))))
)

(defrule GoalModelCreate
  (ModelingProduct (product DomainModel) (ref ?DomainModel&~nil))
  (ModelingTask (task GoalModeling) (ref ?GoalModeling&~nil))
  ?GoalModel <- (ModelingProduct (product GoalModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Goal Model" (name ?name)))
  (modify ?GoalModel (ref ?ref))
  (slot-set ?GoalModeling products ?ref)
  (slot-insert$ ?DomainModel subproducts 1 ?ref)
)
```

Listagem 1 - Regras em Jess de criação do modelo de objetivos

As regras acima criam na ONTORMAS as instâncias das classes correspondentes à tarefa de modelagem de objetivos e seu produto. Na primera regra, se a fase de modelagem corrente for análise de domínio e a próxima tarefa a ser executada for modelagem de objetivos, então a instância da classe correspondente na ONTORMAS será instanciada e será associada à instância da classe que corresponde à fase de análise de domínio. Quando a tarefa de modelagem de objetivos for criada, é necessário instanciar a classe correspondente ao seu produto e criar a relação semântica entre elas, o que é feito pela segunda regra que, além de associar o produto modelo de objetivos à tarefa modelagem de objetivos, também é necessário criar a relação semântica entre este produto como um subproduto da fase de análise de domínio.

4.2.2.2 Sistema de Regras de transformação de modelos

As regras de transformação geram automaticamente um modelo alvo a partir de um conjunto de modelos fonte, com base no ciclo de vida do MADAE-Pro. O processo de transformação (P_t) de um conjunto de modelos fonte em um modelo

alvo é definido como $P_t = (S_1, S_2, ..., S_n, T, U, R_t)$, sendo S_i um modelo fonte, T o modelo alvo produto da tarefa de modelagem t, e tendo S_i e T a mesma definição do conjunto T no processo de criação (P_c) descrito acima. U é a informação de modelagem fornecida pelo desenvolvedor através das interações com o ambiente MADAE-IDE usando o assistente de modelagem (ver exemplo da Figura 39) associado com a respectiva tarefa de modelagem.

R_t é o conjunto de regras de transformação para geração automática de um modelo alvo a partir de um conjunto de modelos fonte e a informação U fornecida pelo desenvolvedor. Por exemplo, na fase de análise de domínio, um modelo de papéis deve ser automaticamente gerado através de um conjunto de regras de transformação que exploram as relações semânticas entre um modelo de objetivos e um modelo de papéis, como ilustrado na Figura 27.

A regra em Jess da Listagem 2, subconjunto das regras de transformação, gera o produto da fase de modelagem de papéis na ONTORMAS, com base o modelo de objetivos que serve de insumo e do qual reusam-se conceitos nele instanciados.

```
(defrule GoalModelToRoleModel
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  (ModelingTask (task RoleModeling) (ref ?RoleModeling&~nil))
  (ModelingProduct (product RoleModel) (name ?name) (ref ?RoleModel&~nil))
  =>
  (foreach ?concept (slot-get ?GoalModel concepts)
    (if (or (eq Responsibility (class ?concept))
            (eq External\ Entity (class ?concept))) then
      (slot-insert$ ?RoleModel concepts 1 ?concept)
      (bind ?*aa_reused* (+ 1 ?*aa_reused*))
    )))
```

Listagem 2 - Regra em Jess para transformação do modelo de papéis com base no modelo de objetivos

A regra acima realiza a transformação do modelo de objetivos para o modelo de papéis. Após a classe correspondente à modelagem de papéis tiver sido instanciada, recupera-se a instância da classe correspondente ao modelo de objetivos e dele extraem-se todas as instâncias de classes que representam conceitos de modelagem. Caso a classe da instância do conceito de modelagem for uma responsabilidade ou entidade externa, esta instância é reusada no modelo de papéis.

A Figura 28 mostra as relações de transformação existentes entre os modelos do processo MADAE-Pro. Analisando o gráfico, depreende-se que o Modelo de Papéis pode ser gerado tendo como insumo o Modelo de Objetivos. Já o Modelo da Sociedade Multiagente é insumo para a criação do Modelo de Interação entre Agentes, Modelo de Conhecimento do Agente e Modelo de Ações do Agente.

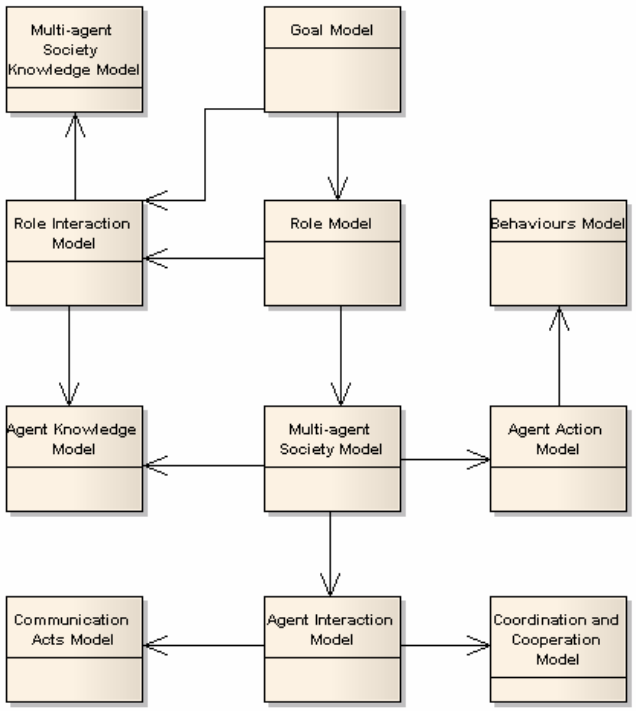

Figura 28 - Relações de transformação entre modelos

A seguir é detalhado para cada fase do processo MADAE-Pro, a maneira como os conceitos de modelagem são adquiridos através da interface com o usuário, como são criados e inseridos os fatos na memória de trabalho do núcleo do ambiente e, finalmente, como são exibidos os modelos para o usuário. Em paralelo é examinado como o núcleo do ambiente infere os fatos instanciados e realiza alterações na ONTORMAS. Também mostra-se a geração automática de modelos onde for aplicável.

4.2.3 Implementação do ambiente MADAE-IDE

As tarefas de modelagem do processo MADAE-Pro estão conceitualizadas na ontologia ONTORMAS como subclasses da classe abstrata

"Modeling Tasks". Como mostra a rede semântica da Figura 29, a ONTORMAS possui as classes abstratas "Domain Engineering Tasks", representando o subprocesso Engenharia de Domínio, e "Application Engineering Tasks" que representa o subprocesso Engenharia de Aplicações, subclasses de "Modeling Tasks", através da relação semântica "is-a". O produto da Engenharia de Domínio é conceitualizado na classe "Domain Engineering Products" e o da Engenharia de Aplicações na classe "Application Engineering Products", representado na relação semântica "products", sendo ambas classes abstratas herdadas da classe abstrata "Modeling Products".

Durante a modelagem utilizando-se o processo MADAE-Pro, toda instância da classe correspondente a cada tarefa de modelagem, como por exemplo, "Modelagem de Papéis", possuirá o slot "products" contendo referência à instância da classe que corresponde ao seu produto de modelagem, no caso "Modelo de Papéis". A instância da classe correspondente a determina tarefa de modelagem será referenciada pelo slot "subtasks" da instância da classe correspondente à fase de modelagem a qual pertence que neste exemplo é "Análise de Domínio", que por sua vez possui uma referência à instância da classe que representa seu produto de modelagem através do slot "products", que neste exemplo é "Modelo de Domínio"; este possuirá uma referência à instância da classe que representa o produto de modelagem de cada tarefa específica da fase, pelo slot "subproducts", que conterá, neste caso, uma referência ao modelo de papéis.

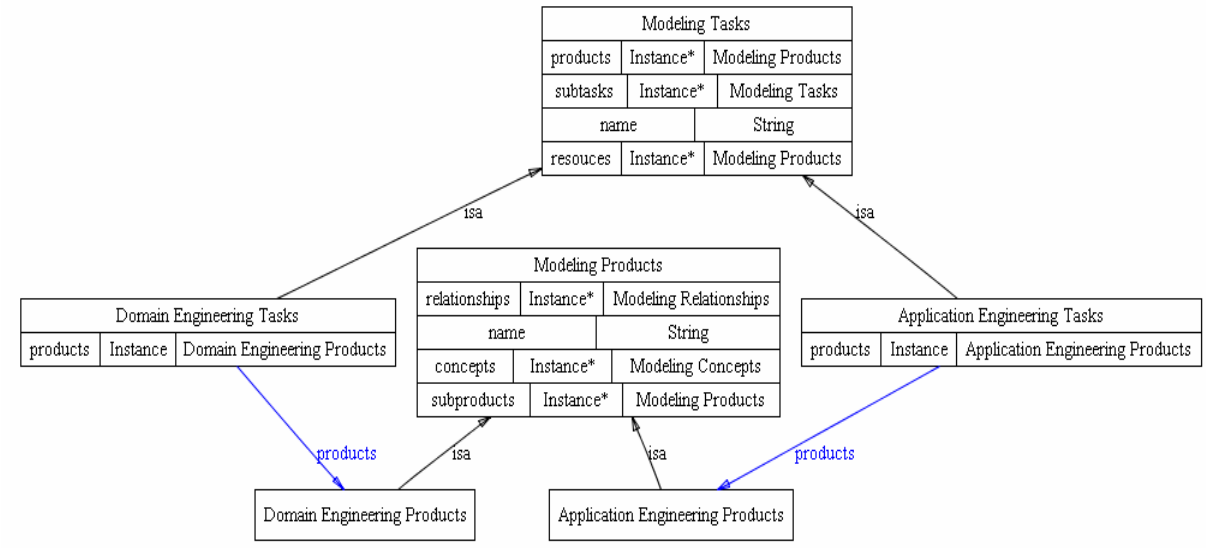

Figura 29 - Classes conceituais de tarefas e produtos de modelagem do processo MADAE-Pro na ONTORMAS

A modelagem de uma família de aplicações multiagente no ambiente MADAE-IDE inicia com a definição do nome do projeto (Figura 30) que serve de nomenclatura para os nomes das instâncias das tarefas e produtos do processo MADAE-Pro na ONTORMAS. Este passo cria e insere um fato na memória de trabalho do ambiente, que dispara as regras de criação e transformação de modelos, associadas com o sub-processo Engenharia de Domínio ou "Domain Engineering Tasks".

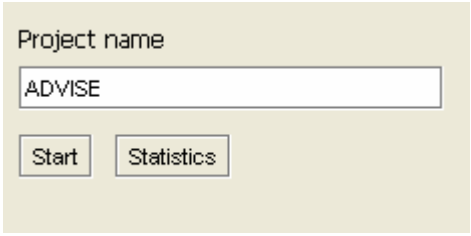

Figura 30 – Assistente de modelagem de definição do projeto no ambiente MADAE-IDE

4.2.4 Engenharia de Domínio no ambiente MADAE-IDE

Esta seção detalha como o ambiente MADAE-IDE automatiza as fases da Engenharia de Domínio do processo MADAE-Pro.

4.2.4.1 Fase de Análise de Domínio

A fase de Análise de Domínio aborda a construção de um Modelo de Domínio, no qual são especificados os requisitos atuais e futuros de uma família de aplicações em um determinado domínio, considerando o conhecimento do domínio e as experiências de desenvolvimento extraídas a partir de especialistas e de aplicações já desenvolvidas nesse domínio.

A rede semântica da Figura 31 ilustra os conceitos e relações semânticas das classes correspondentes à fase de análise de domínio e seu produto. As regras 1.1 do apêndice A são utilizadas para automatizar esta fase.

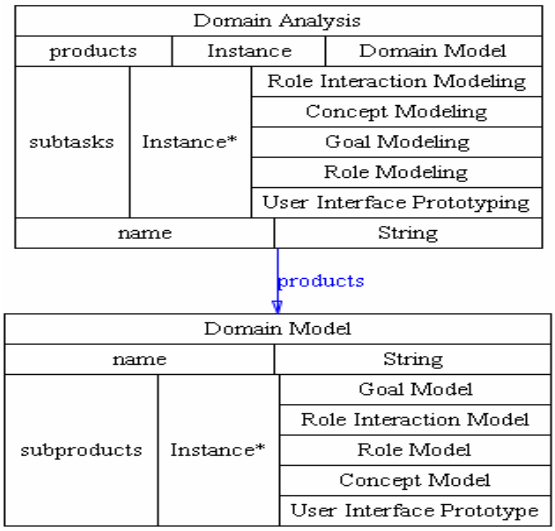

Figura 31 - Representação da subfase Análise de Domínio do processo MADAE-Pro na ONTORMAS

4.2.4.1.1 Modelagem de Conceitos

A Modelagem de Conceitos tem por objetivo reunir o máximo de informações relevantes ao domínio. Nele são representados os conceitos do domínio e suas relações em uma rede semântica, onde os conceitos são representados por nós e suas relações pelos vínculos entre os nós. Este modelo é construído com base em possíveis Modelos de Conceitos preexistentes, no conhecimento de especialistas no domínio, em aplicações já desenvolvidas e na literatura disponível no domínio. As regras 1.2 do apêndice A são utilizadas para automatizar esta tarefa. Na Figura 32 é ilustrada graficamente a tarefa de modelagem de conceitos numa rede semântica.

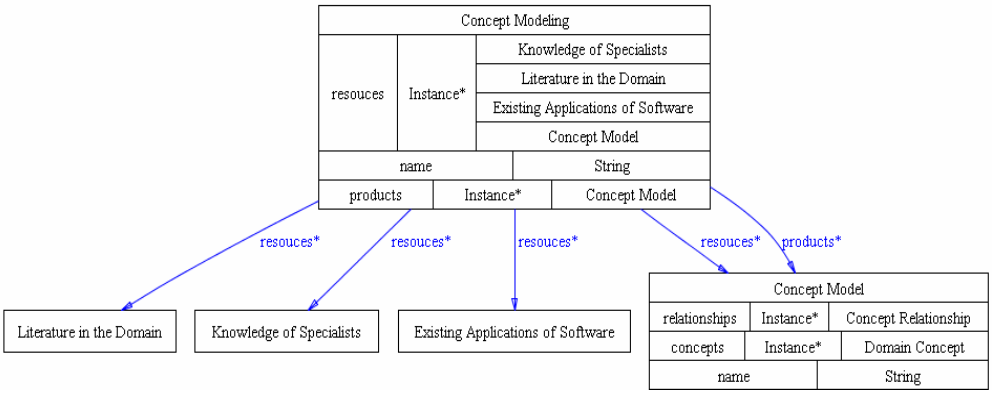

Figura 32 - Representação da subtarefa Modelagem de Conceitos do processo MADAE-Pro na ONTORMAS

Visto que aqui é construída simplesmente uma rede semântica entre os conceitos de domínio, o ambiente não fornece um assistente de modelagem para esta tarefa específica. O modelo de conceitos é exibido vazio ao usuário e assim ele pode criá-lo livremente.

4.2.4.1.2 Modelagem de Objetivos

Na tarefa de modelagem de objetivos são identificados os objetivos comuns e variáveis da família de aplicações multiagente, as entidades externas, e as responsabilidades necessárias para alcançá-los. O Modelo de Objetivos é construído a partir de, e representa, os requisitos de uma família de aplicações multiagente. Nesta tarefa, a modelagem de variabilidades é realizada identificando-se pontos de variação em objetivos específicos relacionados com grupos de responsabilidades variantes. As regras 1.3 do apêndice A são utilizadas para automatizar esta fase. A Figura 33 ilustra a rede semântica da tarefa de modelagem de objetivos.

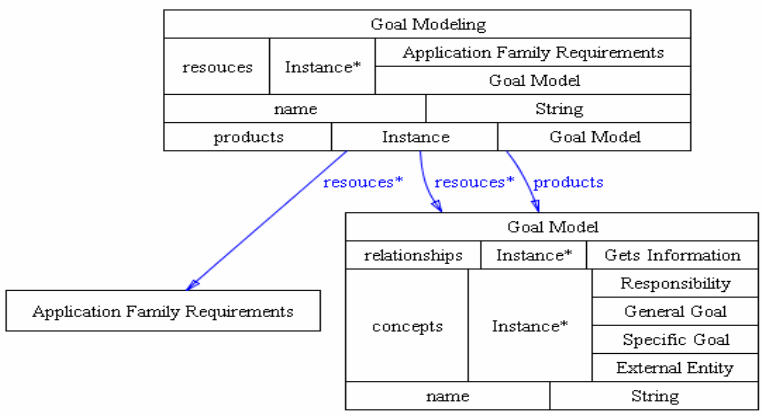

Figura 33 - Representação da subtarefa Modelagem de Objetivos do processo MADAE-Pro na ONTORMAS

O conceito “General Goal”, representa o objetivo geral da família de aplicações multiagente. “Specific Goal” representa um objetivo específico associado ao objetivo geral. O conceito “Specific Goal” está associado a ele mesmo através do relacionamento semântico “leads to” para indicar os casos onde existe uma hierarquia de objetivos específicos. O conceito “General Goal” está relacionado a dois ou mais conceitos “Specific Goal” através do relacionamento semântico “leads

to", isso significa que dois ou mais objetivos específicos conduzem ao objetivo geral. Estes conceitos são obtidos através do assistente de modelagem apresentado na Figura 34.

O conceito "Responsibility" representa uma responsabilidade necessária para alcançar um ou mais objetivos específicos. O conceito "Responsibility" está associado a "Specific Goal" pelo relacionamento semântico "achieves", isso denota que um objetivo específico será alcançado através da execução de uma ou mais responsabilidades. Estes conceitos são obtidos através do assistente de modelagem apresentado na Figura 35.

O conceito "External Entity" representa uma entidade externa com a qual é necessário interagir para alcançar objetivos específicos e podem ser informados pelo assistente de modelagem apresentado na Figura 36.

O conceito "External Entity" está associado a "Specific Goal", indicando que para alcance de um objetivo específico pode haver necessidade de interação com entidades externas. Estas associações podem ser informadas através do assistente de modelagem apresentado na Figura 37.

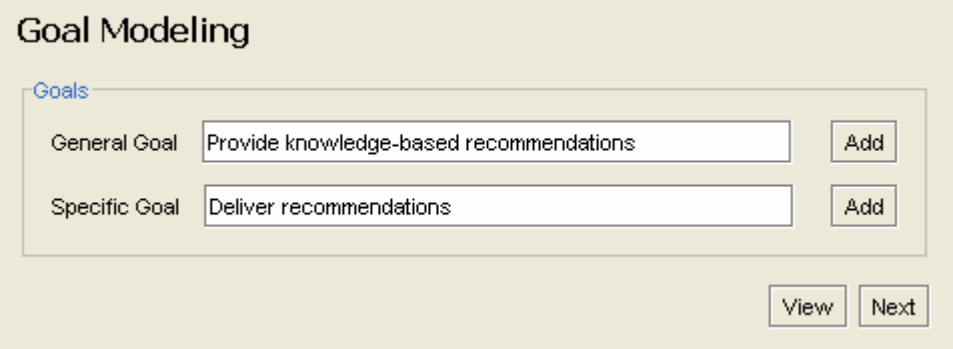

Figura 34 - Interface do ambiente MADAE-IDE para obtenção dos objetivos na modelagem de objetivos do processo MADAE-Pro

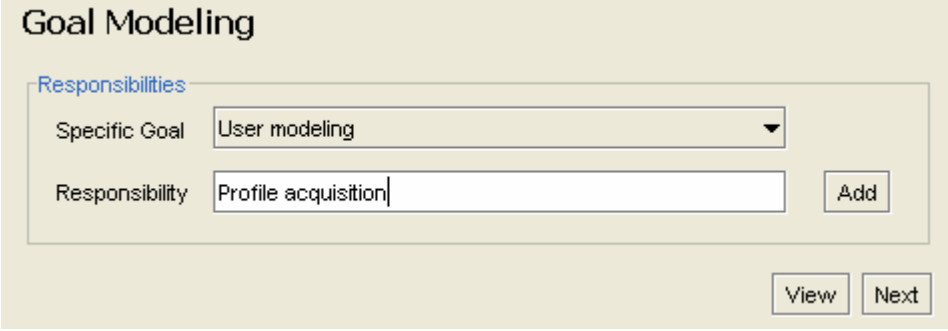

Figura 35 - Interface do ambiente MADAE-IDE para obtenção das responsabilidade na modelagem de objetivos do processo MADAE-Pro

Figura 36- Interface do ambiente MADAE-IDE para obtenção das entidades externas na modelagem de objetivos do processo MADAE-Pro

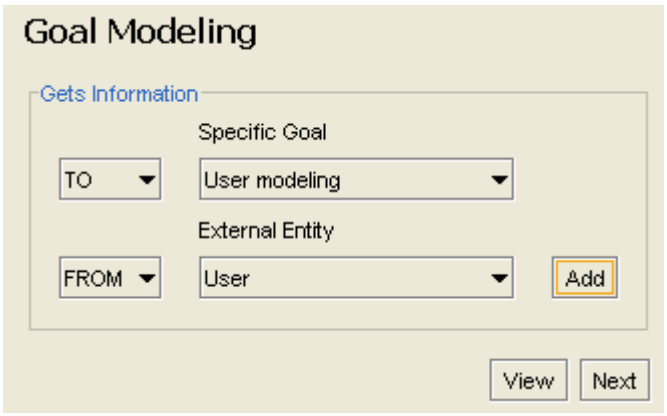

Figura 37 - Interface do ambiente MADAE-IDE para obtenção dos relacionamentos entre as entidades externas e os objetivos na modelagem de objetivos do processo MADAE-Pro

4.2.4.1.3 Modelagem de Papéis

A tarefa de modelagem de papéis associa as responsabilidades, comuns ou variantes, identificadas na tarefa de modelagem de objetivos aos papéis que serão encarregados delas. As pré e pós-condições que devem ser satisfeitas antes e depois da execução de uma responsabilidade são identificadas. O conhecimento requerido a partir de outras entidades (papéis ou entidades externas) para a realização das responsabilidades e o conhecimento produzido a partir da sua execução é identificado. Esta tarefa produz um conjunto de modelos de papéis, um para cada objetivo específico ou, tendo um ou mais pontos de variação, um modelo de papéis para cada variante, especificando papéis, responsabilidades, pré e pós-condições, conhecimento e os relacionamentos semânticos entre estes conceitos.

Esse modelo é construído base no modelo de objetivos (Figura 28). Todas as responsabilidades do modelo de objetivos são automaticamente adicionadas ao modelo de papéis e através das informações de modelagem adicionais fornecidas pelo usuário o modelo será construído iterativamente. As

regras 1.4 do apêndice A são utilizadas para automatizar a modelagem de papéis. A Figura 38 ilustra a rede semântica da modelagem de papéis.

O conceito “Role” possui o relacionamento semântico “in charge of” com o conceito “Responsibility”, significando que um papel é encarregado pela realização de uma responsabilidade. O conceito “Responsibility” possui os relacionamentos semânticos “used by” e “produces” com o conceito “Knowledge”, isso significa que para a/depois da realização de uma responsabilidade conhecimento pode ser requerido/produzido. Estes conceitos são obtidos através do assistente de modelagem apresentado na Figura 39.

O conceito “External Entity” relaciona-se com o conceito “Knowledge” através das relações semânticas “knows”, informando que a entidade externa possui determinado conhecimento, e “turns known”, informando que determinado conhecimento é disponibilizado à entidade externa. Estas relações são obtidas através do assistente de modelagem apresentado na Figura 40.

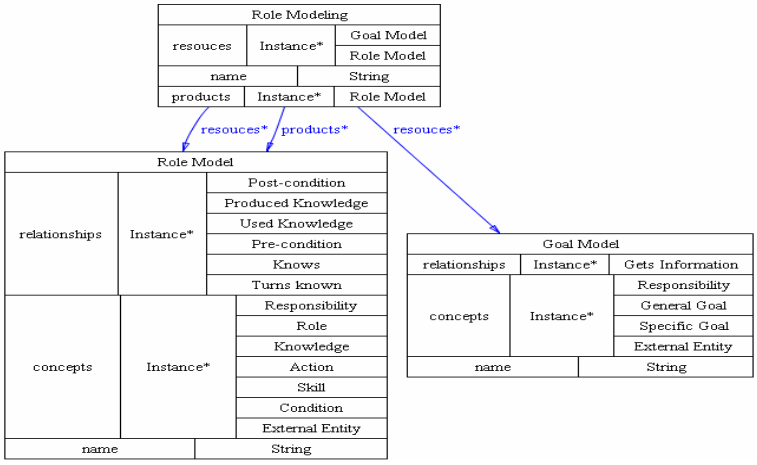

Figura 38 - Representação da subtarefa Modelagem de Papéis do processo MADAE-Pro na ONTORMAS

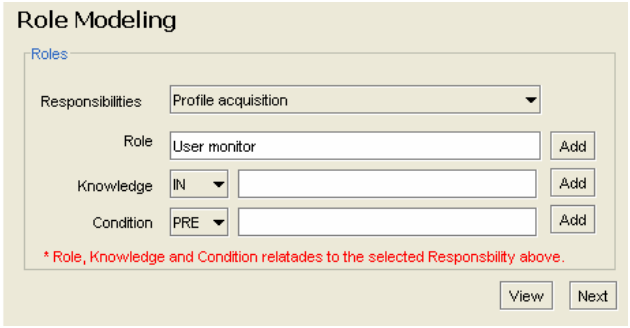

Figura 39 - Interface do ambiente MADAE-IDE para obtenção dos papéis, conhecimentos e condições para cada responsabilidade na modelagem de papéis do processo MADAE-Pro

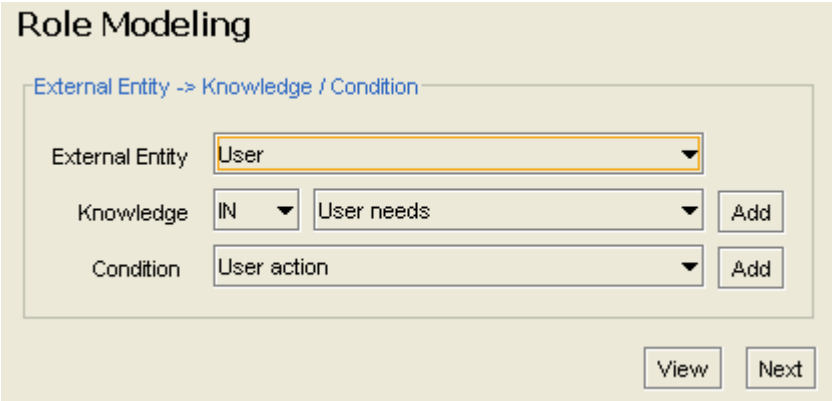

Figura 40 - Interface do ambiente MADAE-IDE para informação dos relacionamentos entre as entidades externas e conhecimentos e condições na modelagem de papéis do processo MADAE-Pro

4.2.4.1.4 Modelagem de Interação entre Papéis

A tarefa de modelagem de interações entre papéis representa a cooperação das entidades externas (usuários, outros sistemas, etc) e internas (papéis) para alcançar um objetivo específico. Esse modelo é construído com base no modelo de objetivos e no modelo de papéis (Figura 28). O modelo de objetivos é utilizado para que seja criado um modelo de interações entre papéis para cada objetivo específico nele presente e o modelo de papéis para que se possa inserir no novo modelo de interação entre papéis, os papéis responsáveis pelo alcance de cada objetivo específico. As regras 1.5 do apêndice A são utilizadas para automatizar a modelagem de interação entre papéis. Na Figura 42 a rede semântica da modelagem de interação entre papéis é ilustrada.

Estão disponíveis os conceitos “Role”, “External Entity” e “Role Interaction”. O conceito “Role Interaction” representa as interações entre dois conceitos “Role” ou entre um conceito “Role” e um conceito “External Entity”, através do relacionamento semântico “participates”.

Este modelo é gerado automaticamente pelas regras 1.5 do apêndice A. Para visualizá-lo, basta o usuário escolher para qual objetivo específico deseja-se visualizar o modelo de interação entre papéis, pelo assistente de modelagem visto na Figura 41.

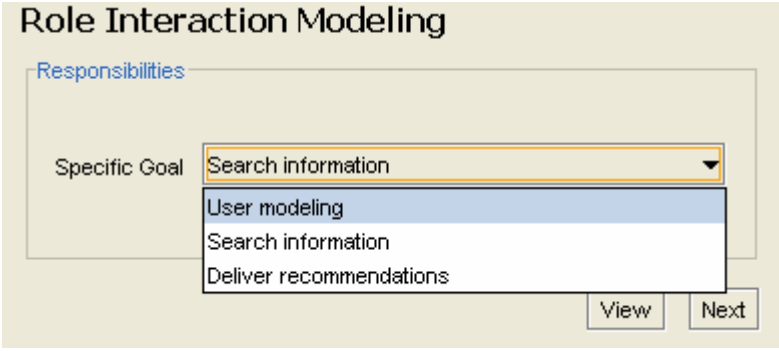

Figura 41 - Interface do ambiente MADAE-IDE para seleção do modelo de interação entre papéis do processo MADAE-Pro para visualização

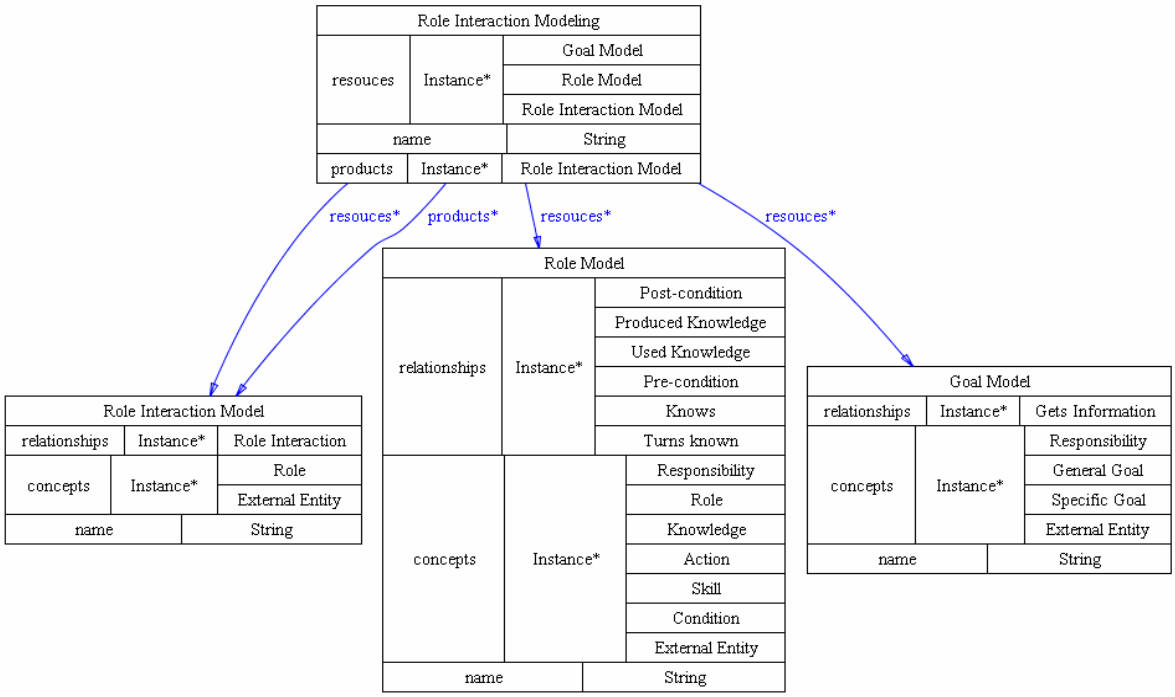

Figura 42 - Representação da subtarefa Modelagem de Interação entre Papéis do processo MADAE-Pro na ONTORMAS

4.2.4.1.5 Prototipação da Interface do Usuário

O Protótipo da Interface com o Usuário representa as interações dos usuários com o sistema. Esse protótipo é construído a partir do reuso de protótipos de interface com os usuários preexistentes e com base no Modelo de Papéis e no Modelo de Interações entre Papéis.

O ambiente MADAE-IDE não conta com um assistente para esta tarefa, pois seu produto é um documento contendo as imagens com a prototipação das telas que fazem interface com o usuário.

4.2.4.2 Fase de Projeto de Domínio

A fase de Projeto de Domínio aborda o projeto arquitetural e detalhado de frameworks multiagente provendo uma solução aos requisitos de uma família de aplicações multiagente especificadas em um Modelo de Domínio. Na ONTORMAS a classe “Domain Design” conceitualiza esta tarefa e a classe “Domain Model” representa seu produto, como mostra a rede semântica da Figura 43. As regras 2.1 do apêndice A são utilizadas para automatizar a fase de Projeto de Domínio.

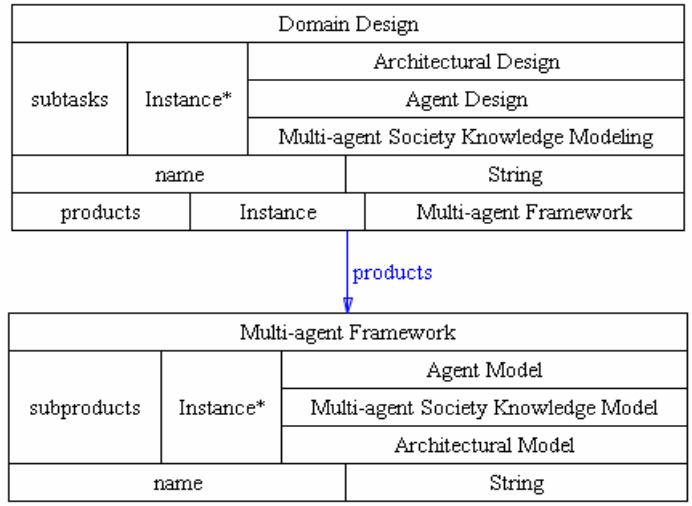

Figura 43 - Representação da subfase Projeto de Domínio do processo MADAE-Pro na ONTORMAS

4.2.4.2.1 Fase de Modelagem Arquitetural

A instância da classe correspondente à fase projeto arquitetural, (“Architectural Design”), é associada à instância da classe correspondente à de projeto de domínio. Uma instância da classe correspondente ao produto desta tarefa (“Architectural Model”) é associada à instância da classe correspondente à subfase de modelagem arquitetural pelo relacionamento semântico “products” e também à instância da classe correspondente ao produto da fase de projeto de domínio pelo relacionamento semântico “subproducts”. A instância “Architectural Design” conterá a representação de suas subtarefas pelo relacionamento semântico “subtasks”, e pelo relacionamento semântico “products”, o seu produto de modelagem “Architectural Model”, que contém as representações dos produtos gerados em cada

subtarefa, como mostra a rede semântica da Figura 44. As regras 2.1 do apêndice A são utilizadas para automatizar estas tarefas.

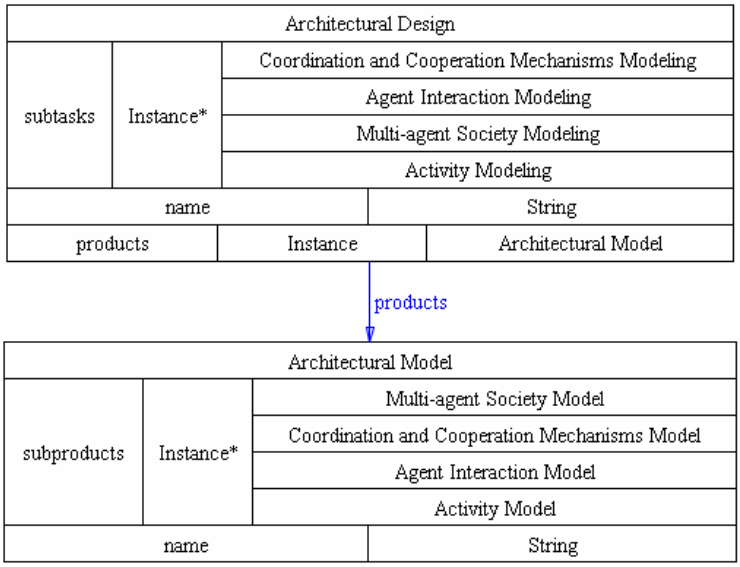

Figura 44 - Representação da subfase Modelagem Arquitetural do processo MADAE-Pro na ONTORMAS

4.2.4.2.1.1 Modelo do Conhecimento da Sociedade Multiagente

O Modelo do Conhecimento da Sociedade Multiagente representa os conceitos que os agentes da sociedade precisam entender para se comunicar. Este modelo é construído com base no Modelo de Interações entre Papéis (Figura 28). O conhecimento trocado pelos papéis durante as suas interações é integrado e refinado neste modelo.

Na Figura 45, a rede semântica da modelagem do conhecimento da sociedade multiagente é ilustrada. Estão disponíveis os conceitos “Knowledge”, que representa o conhecimento do domínio e “Knowledge Relationship”, que representa os possíveis relacionamentos entre conceitos de conhecimento. “Part of”, “kind of” e “is a” são exemplos de “Knowledge Relationship”.

De cada modelo de interação entre papeis são extraídos os conhecimentos, pela regra de transformação 2.1.1 do apêndice A. Se um conhecimento é usado no slot “parameter” em um relacionamento “Role Interaction” do modelo de interação entre papéis, o conhecimento é adicionado ao modelo de conhecimento da sociedade multiagente.

Para visualizá-lo, basta o usuário clicar no botão “View”, do assistente de modelagem visto na Figura 46. Após a exibição do modelo, o usuário poderá criar os relacionamentos entre os conhecimentos diretamente no modelo exibido.

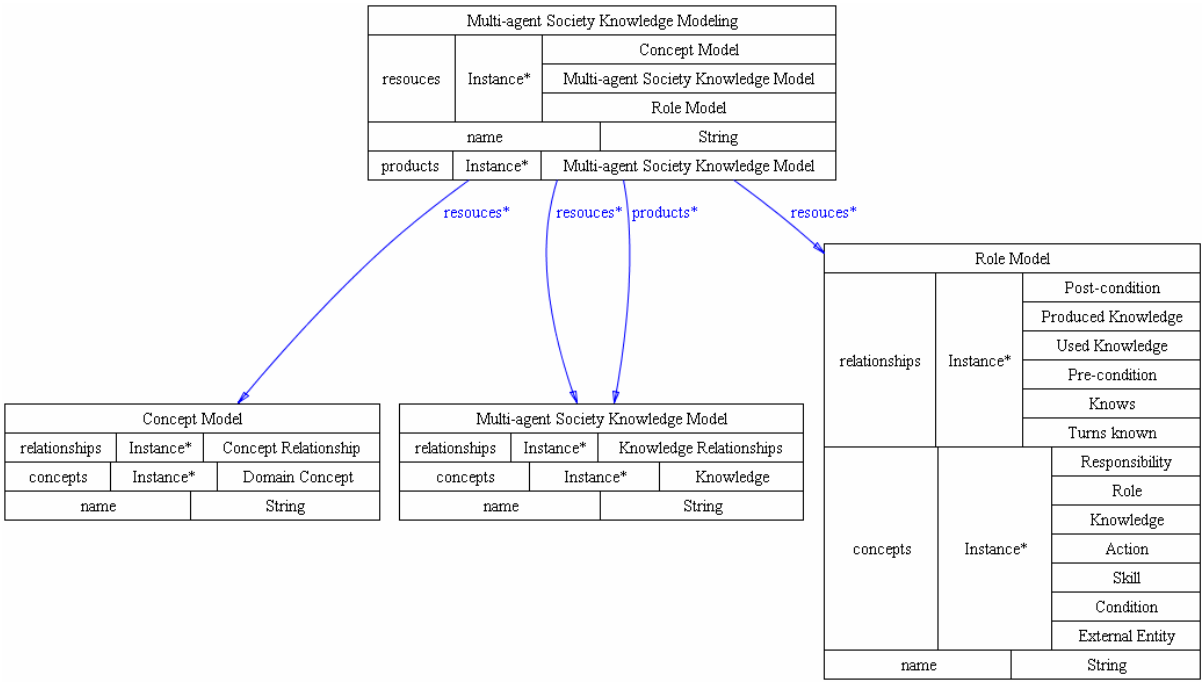

Figura 45 - Representação da subtarefa Modelagem do Conhecimento da Sociedade Multiagente do processo MADAE-Pro na ONTORMAS

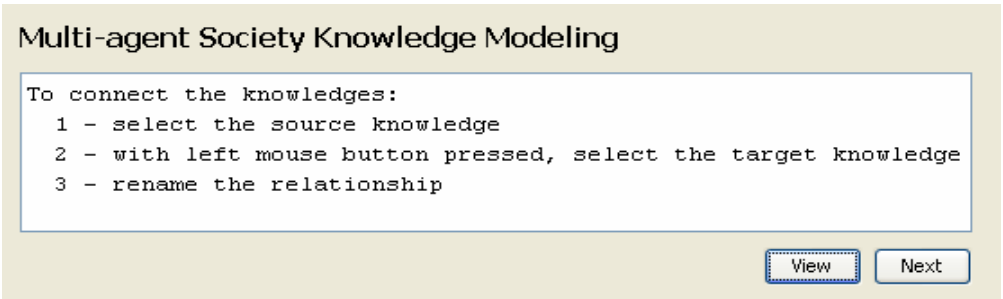

Figura 46 - Interface do ambiente MADAE-IDE para visualizar o modelo de conhecimento da sociedade multiagente

4.2.4.2.1.2 Modelo da Sociedade Multiagente

O Modelo da Sociedade Multiagente representa os agentes da sociedade, juntamente com o conhecimento, as destrezas e as pré e pós-condições necessárias para realização das suas responsabilidades. Esse modelo é construído com base no Modelo de Papéis da fase de Análise de Domínio (Figura 28). Pré e pós-condições,

responsabilidades, conhecimento e entidades externas são obtidas a partir do Modelo de Papéis. Um agente pode representar um ou mais papéis de acordo com a afinidade entre suas responsabilidades, número de interações entre eles ou critério de coesão funcional.

Na Figura 47 a rede semântica da tarefa de modelagem da sociedade multiagente é ilustrada. Os conceitos são “Agent”, representando os agentes da sociedade; “Responsibility”, representando as responsabilidades dos agentes; “Knowledge”, representando o conhecimento que o agente usa e produz durante a realização de suas responsabilidades; “Condition”, representando as pré e pós-condições que devem ser satisfeitas para/depois da realização de uma responsabilidade; “Skill”, que representa uma habilidade que o agente deve possuir para realizar uma determinada responsabilidade; “External Entity”, representando as entidades externas com as quais os agentes interagem para realizar suas responsabilidades. Os relacionamentos semântico são “in charge of”, que indica que um agente será encarregado de uma responsabilidade; “requires”; utilizado quando para a realização de uma responsabilidade é necessário atender uma pré ou pós-condição; “required by”, representando que uma destreza é necessária para realização de uma determinada responsabilidade; “has knowledge”, que indica o conhecimento fornecido por entidades externas; “exercised through”, representando a realização de uma responsabilidade através de um conjunto de atividades; “used by” e “produces”, representando o conhecimento requerido e produzido através da realização de uma atividade.

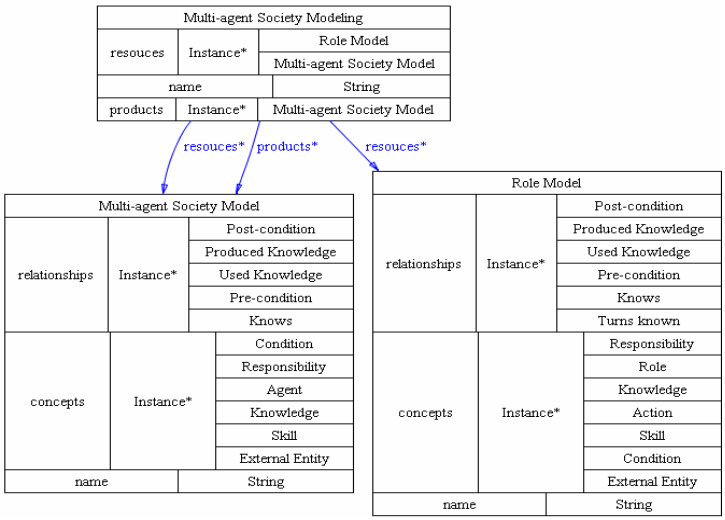

Figura 47 - Representação da subtarefa Modelagem da Sociedade Multiagente do processo MADAE-Pro na ONTORMAS

Nas regras 2.1.2 do apêndice A, os conceitos e relacionamentos do modelo de papéis passíveis de reuso são adicionados ao modelo da sociedade multiagente da seguinte forma:

- Toda responsabilidade, que faz parte do modelo de papel, é adicionada ao modelo da sociedade multiagente;
- Todo conhecimento, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente;
- Toda condição, que faz parte do modelo de papel, é adicionada ao modelo da sociedade multiagente;
- Toda entidade externa, que faz parte do modelo de papel, é adicionada ao modelo da sociedade multiagente;
- Todo relacionamento do tipo “produce”, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente;
- Todo relacionamento do tipo “used by”, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente;
- Todo relacionamento do tipo “is sufficient for”, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente;
- Todo relacionamento do tipo “is necessary for”, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente.
- Todo relacionamento do tipo “Knows” entre uma entidade externa e um conhecimento, que faz parte do modelo de papel, é adicionado ao modelo da sociedade multiagente.

Para finalizar a modelagem da sociedade multiagente deve-se definir o agente e as habilidades relacionadas a cada responsabilidade, através dos assistentes de modelagem vistos na Figura 48 e Figura 49.

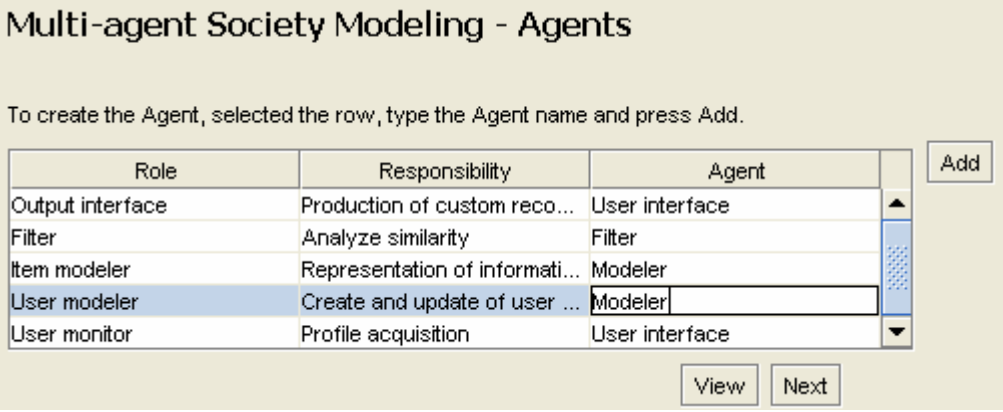

Figura 48 - Interface do ambiente MADAE-IDE para associar-se os agentes às responsabilidades

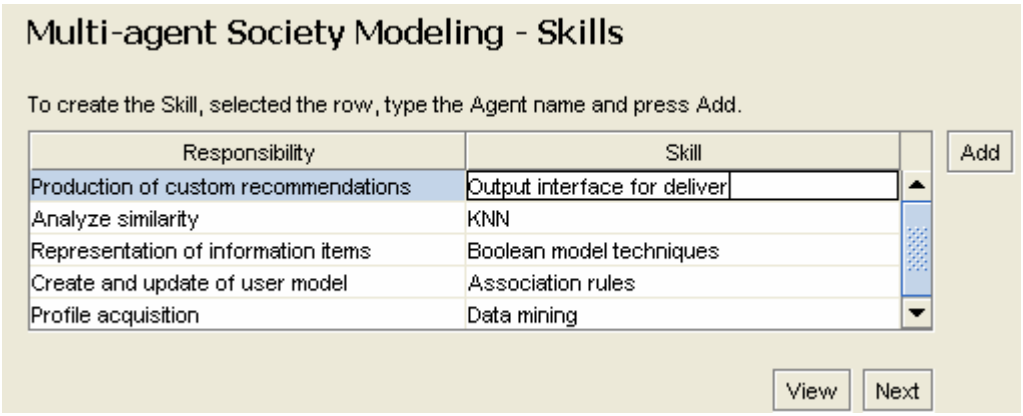

Figura 49 - Interface do ambiente MADAE-IDE para associar-se as habilidades às responsabilidades

4.2.4.2.1.3 Modelo de Interação entre Agentes

O Modelo de Interações entre Agentes representa as interações entre os agentes da sociedade. Este modelo é semelhante a um diagrama de seqüencia da UML. Fazendo uma analogia entre estes dois modelos, um agente ou entidade externa no Modelo de Interações entre Agentes seria um objeto no diagrama de seqüencia da UML. As mensagens são especificadas seguindo as diretrizes da linguagem de comunicação entre agentes FIPA-ACL (FIPA, 2008). Este modelo é construído com base no Modelo da Sociedade Multiagente (Figura 28), de onde são obtidos os agentes, as entidades externas com que eles interagem e o conhecimento trocado em suas comunicações.

Na Figura 50 a rede semântica da modelagem de interações entre agentes é ilustrada. Os novos conceitos apresentados aqui são “Agent Exchanged

Message" que representa a interação entre agentes e "Agent Notification from External Entity" que representa a interação entre um agente e uma entidade externa.

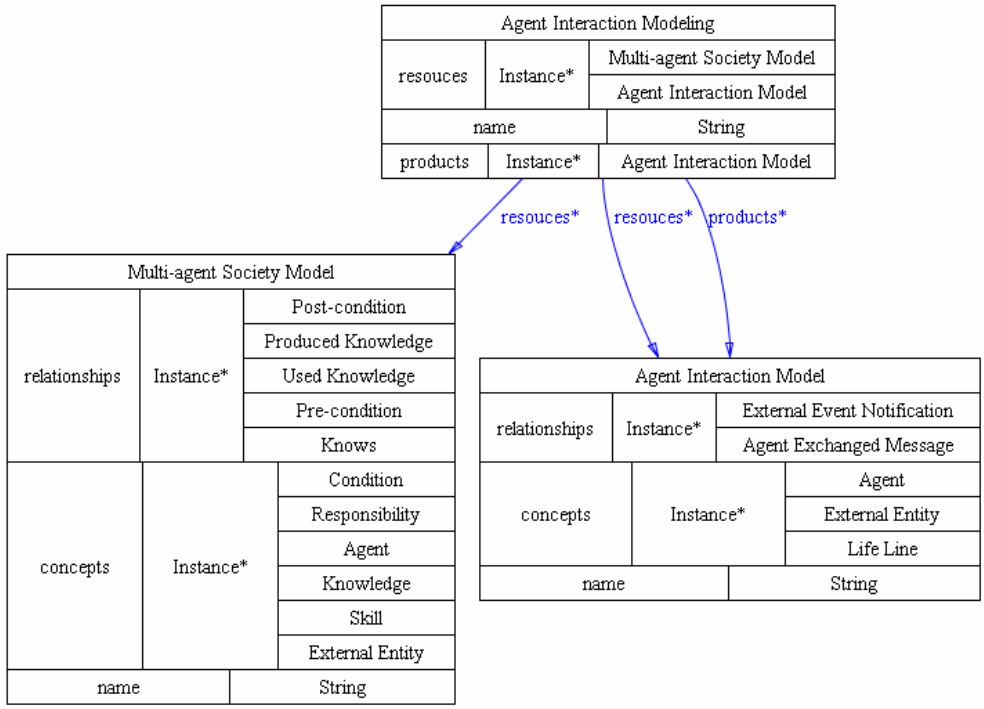

Figura 50 - Representação da subtarefa Modelagem de Interação entre Agentes do processo MADAE-Pro na ONTORMAS

Os agentes e as entidades externas são adquiridos do modelo da sociedade multiagente. Uma linha de tempo é criada e relacionada para os conceitos adicionados. Por fim são criados os relacionamentos entre os conceitos. Este modelo é gerado automaticamente pelas regras 2.1.3 do apêndice A, seguindo os seguintes critérios:

- Para todo agente do modelo da sociedade multiagente, é adicionado um agente ao modelo de interação entre agentes e adicionada uma linha de vida com o slot "associated with" relacionado a este agente.
- Para toda entidade externa do modelo da sociedade multiagente, é adicionada uma entidade externa ao modelo de interação entre agentes e adicionada uma linha de vida com o slot "associated with" relacionado a esta entidade externa.
- Para todo relacionamento do tipo "Knows" (k) que está contido no modelo da sociedade multiagente e tem slot ":FROM" igual ao

agente (a1); verifica-se o conhecimento (c) que está no slot ":TO" de (k), então se filtra um relacionamento do tipo "used by" (u) que tenha o slot ":FROM" igual ao conhecimento (c); verifica-se que a responsabilidade (r) seja igual ao slot ":TO" de (u), assim filtra-se o agente (a2) contido no slot "performed by" de (r); verifica-se as linhas de vida (l1) e (l2), onde (l1) têm o slot "associated with" igual a (a1) e (l2) têm o slot "associated with" igual a (a2). Então se adiciona um relacionamento do tipo "Agent Exchanged Message" com parâmetros ":FROM"= (l1),:"TO" = (l2)," content" = (c) ao modelo de interação entre agentes.

- Para todo relacionamento do tipo "Knows" (k) que está contido no modelo da sociedade multiagente e tem slot ":FROM" igual à entidade externa (ee), verifica-se o conhecimento (c) que está no slot ":TO" de (k), então se filtra um relacionamento do tipo "used by" (u) que tenha o slot ":FROM" igual ao conhecimento (c), e verifica-se a responsabilidade (r) que seja igual ao slot ":TO" de (u), filtra-se o agente (a) que tenha o slot "in_charge_of" igual a (r), assim verifica-se as linhas de vida (l1) e (l2), onde (l1) tem o slot "associated with" igual a (ee) e (l2) tem o slot "associated with" igual a (a), filtra-se o relacionamento "is sufficient for" (s) onde o slot ":TO" for igual a (r) e verifica-se a condição (con) que está no slot (:FROM) de (s). Então se adiciona um relacionamento do tipo "External Event Notification" com parâmetros ":FROM"= (l1),:"TO" = (l2)," content" = (con) ao modelo de interação entre agentes.
- Para todo relacionamento do tipo "Turns Knows" (tk) que está contido no modelo da sociedade multiagente e tem slot ":TO" igual à entidade externa (ee), verifica-se o conhecimento (c) que está no slot ":FROM" de (tk), então se filtra um relacionamento do tipo "Produced Knowledge" (u) que tenha o slot ":TO" igual ao conhecimento (c), e verifica-se a responsabilidade (r) que seja igual ao slot ":FROM" de (u), filtra-se o agente (a) contido no slot "performed by" de (r), verifica-se as linhas de vida (l1) e (l2), onde (l1) tem o slot "associated with" igual a (a) e (l2) tem o slot "associated with" igual a (ee), filtra-se o relacionamento "is

necessary for” (n) onde o slot “:FROM” for igual a (r) e verifica-se a condição (con) que está no slot (:TO) de (n). Então se adiciona um relacionamento do tipo "External Event Notification" com parâmetros “:FROM”= (I1),:”TO” = (I2),” content” = (con) ao modelo de interação entre agentes.

Para visualizá-lo, basta o usuário clicar no botão “View”, como mostra a Figura 51.

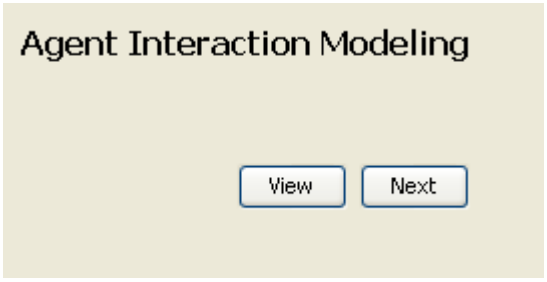

Figura 51 - Interface do ambiente MADAE-IDE visualizar-se o modelo de interação entre agentes

4.2.4.2.1.4 Modelo de Cooperação e Coordenação

Para o Modelo de Cooperação e Coordenação ser construído é reusado ou criado uma arquitetura de cooperação e coordenação entre agentes que satisfaçam os requisitos da família de aplicações presentes no Modelo de Objetivos. Esse modelo especifica como os agentes da sociedade vão cooperar, organizar-se e interagir para realizar uma determinada tarefa. O padrão de cooperação determina como os agentes vão interagir para realizar suas responsabilidades. Exemplos de padrões de cooperação são o quadro-negro e o federativo (GIRARDI, 2004). O padrão de coordenação especifica como os agentes estarão organizados. Eles podem estar em um mesmo nível hierárquico como é o caso do padrão de mercado, onde os agentes são classificados em consumidores e vendedores ou em níveis de hierarquia diferentes como, por exemplo, o mecanismo mestre-escravo, onde os agentes são classificados em gerentes (mestres) e em trabalhadores (escravos). Os agentes trabalhadores são coordenados por um gerente que distribui as tarefas entre estes e espera o resultado. Esses padrões estão incluídos na ONTORMAS, como instâncias da classe “Architectural Pattern”.

O Modelo de Cooperação e Coordenação é construído com base no Modelo de Interações entre Agentes (Figura 28). A partir do Modelo de Interações entre Agentes extraem-se as entidades externas, os agentes da sociedade e o conjunto de interações entre eles, auxiliando na escolha da melhor arquitetura de cooperação/coordenação. O Modelo de Objetivos é exibido ao usuário, para que os requisitos não-funcionais sejam visualizados e possam ser atendidos na escolha da arquitetura de cooperação/coordenação.

Na Figura 52, a rede semântica da modelagem de cooperação e coordenação é ilustrado, onde está presente o conceito “Architectural Pattern”, através do slot “applies”, representando o padrão arquitetural no qual os agentes estarão organizados e o conceito “External Entity”, representando as entidades externas com as quais os agentes interagem. As regras 2.1.4 do apêndice A são utilizadas para automatizar esta tarefa de modelagem.

Para se obter este modelo, o usuário deve selecionar o mecanismo de cooperação e coordenação entre agentes. Atualmente, o modelo “Layer” é o único que está implementado, em seguida clicar em “Next”, no assistente de modelagem visto na Figura 53.

Agora se devem montar as camadas do programa. Para cada “Layer” seleciona-se um agente e clica-se botão no botão “Add”, no assistente da Figura 54. Um agente pertence a uma camada, mas uma camada pode conter mais de um agente, logo se existirem “N” agentes poderão existir no máximo “N” camadas.

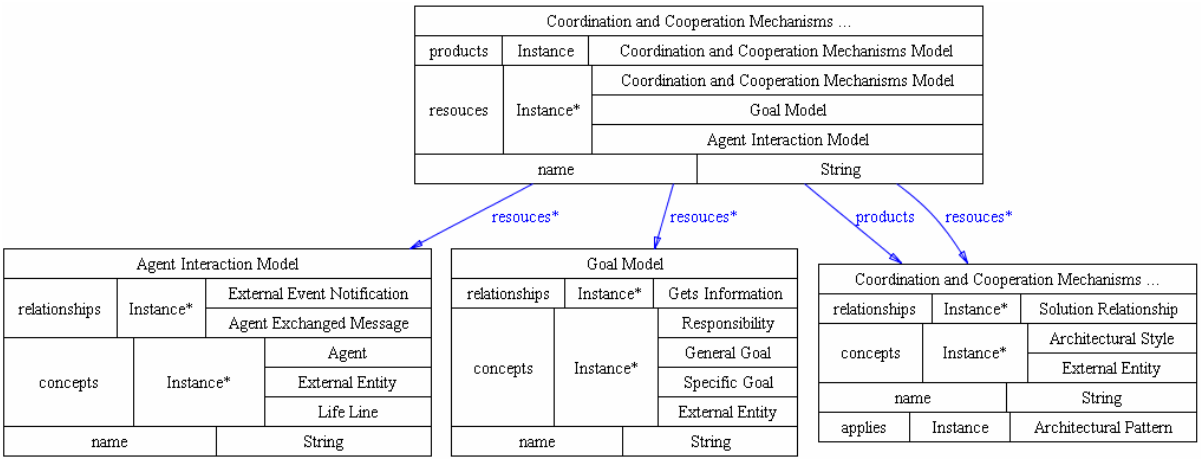

Figura 52 – Representação da subtarefa Modelagem dos Mecanismos de Coordenação e Cooperação do processo MADAE-Pro na ONTORMAS

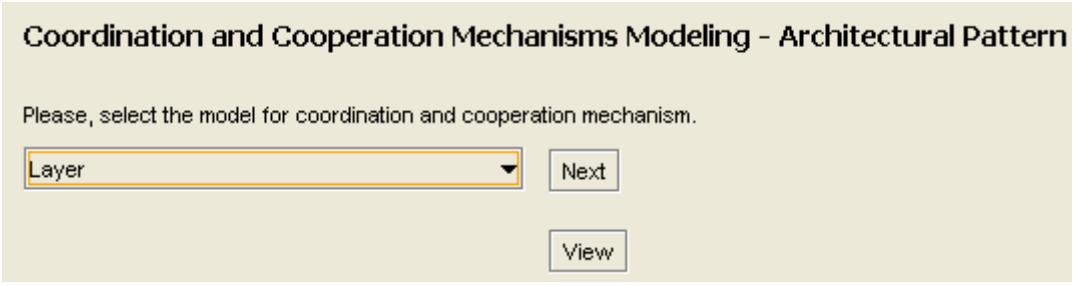

Figura 53 - Interface do ambiente MADAE-IDE para seleçao do mecanismo de cooperação e coordenação

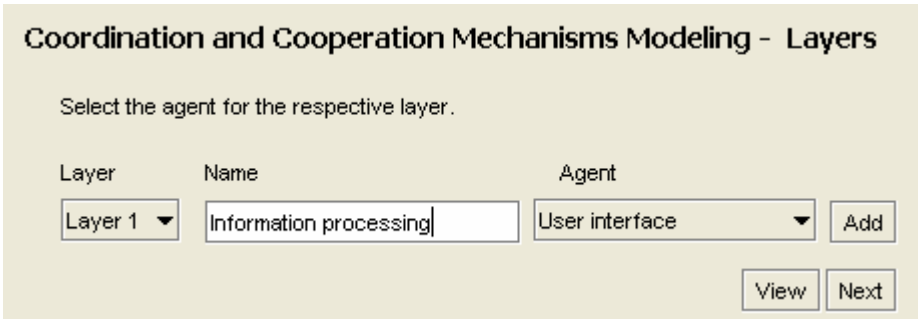

Figura 54 - Interface do ambiente MADAE-IDE para configuração das camadas do mecanismo de coordenação e cooperação

O passo seguinte é montar os relacionamentos semânticos “Solution Relationships” entre as camadas e uma camada e uma entidade externa. Este relacionamento passa como parâmetro um conhecimento. Primeiro seleciona-se o conceito “From” do relacionamento, a seguir o conceito que estará na parte “To” do relacionamento e por fim o conhecimento que será passado como parâmetro, pelo assistente de modelagem visto na Figura 55.

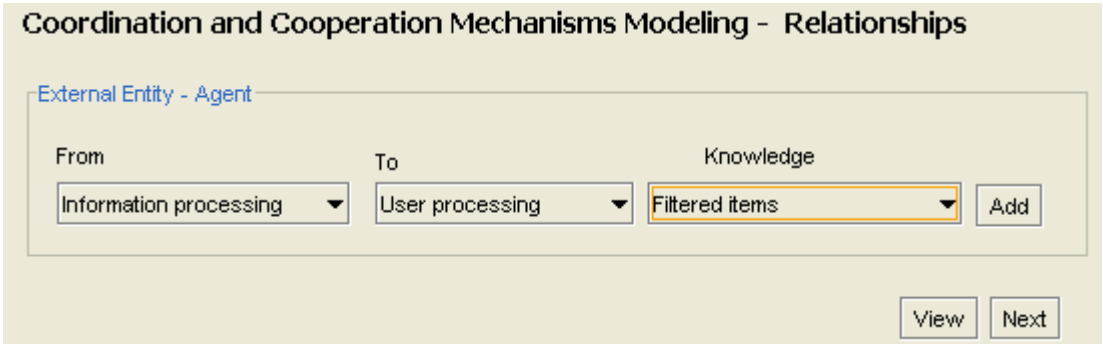

Figura 55 - Interface do ambiente MADAE-IDE para criação dos relacionamentos entre as camadas do mecanismo de coordenação e cooperação

4.2.4.2.2 Fase de Projeto do Agente

Para cada agente da família de aplicações, uma instância da classe "Agent Design" é associada à instância da classe "Domain Design". Uma instância da classe correspondente ao produto desta tarefa, representada na ONTORMAS pela classe "Agent Model" é associada à instância da classe correspondente à subfase de projeto do agente no slot "products" e também à instância da classe correspondente ao produto da fase de projeto de domínio pelo slot "subproducts". A instância da classe "Agent Design" conterá a representação de suas subtarefas através do relacionamento semântico "subtasks" e, através do relacionamento semântico "products", a instância da classe correspondente ao seu produto de modelagem "Agent Model", que contém as representações dos produtos gerados em cada subtarefa, como mostra a rede semântica da Figura 56 e implementadas nas regras 2.2 do apêndice A.

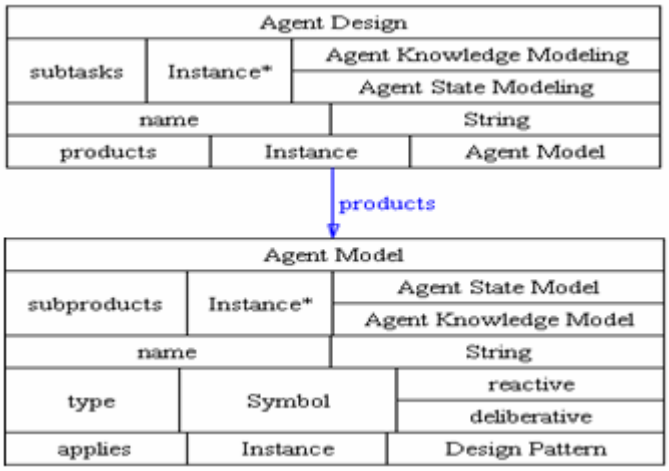

Figura 56 - Representação da subfase Projeto do Agente do processo MADAE-Pro na ONTORMAS

4.2.4.2.2.1 Modelo do Conhecimento do Agente

O Modelo de Conhecimento do Agente representa o conhecimento particular de cada agente da sociedade. Ele é construído com base no Modelo de Interações entre Papéis, o qual provê o conhecimento compartilhado entre os papéis durante as suas interações, e no Modelo do Conhecimento da Sociedade

Multiagente, do qual se obtém os conhecimentos específicos de cada agente (Figura 28). Na Figura 57, a rede semântica da modelagem do conhecimento do agente é ilustrada, onde se vêem os conceitos “Knowledge” e “Knowledge Relationship” que representam, respectivamente, o conhecimento particular de cada agente e o relacionamento entre esses conhecimentos.

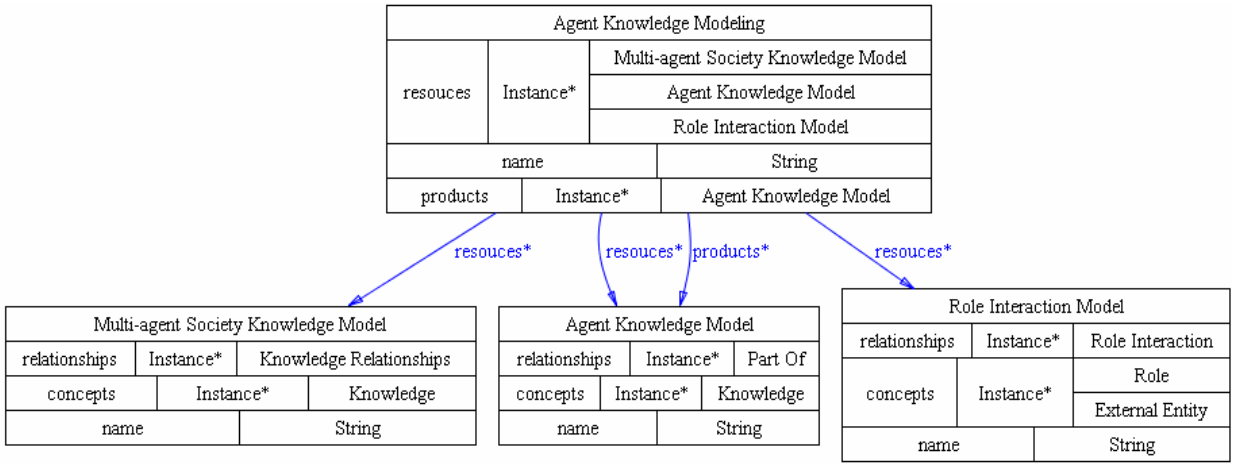

Figura 57 - Representação da subtarefa Modelagem do Conhecimento do Agente do processo MADAE-Pro na ONTORMAS

Para cada agente do modelo da sociedade multiagente é criado um modelo do conhecimento do agente, em seguida, os relacionamentos do modelo de conhecimento da sociedade multiagente entre dois conhecimentos que já compõem o modelo do conhecimento do agente são adicionados. As regras 2.2.1 do apêndice A estão baseadas nos seguintes critérios:

- Para cada agente contido no modelo da sociedade multiagente é criado um modelo do conhecimento do agente.
- Para cada relacionamento do tipo “Knows” (k), contido no modelo da sociedade multiagente, verifica-se se o slot “:FROM” de (k) é igual ao agente (a) e o slot “:TO” de (k) é igual ao conhecimento (c), então é adicionado (c) ao modelo de conhecimento do agente gerado por (a).

No assistente de modelagem visto na Figura 58 seleciona-se um agente e para cada agente será carregado o seu modelo de conhecimento.

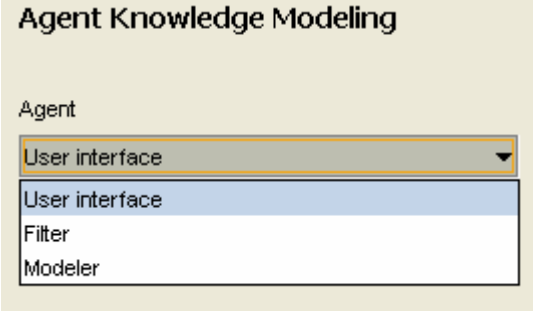

Figura 58 – Assistente de modelagem do ambiente MADAE-IDE para visualização do modelo de conhecimento do agente

4.2.4.2.2.2 Modelos de Ações do Agente

O Modelo de Ações do Agente representa as ações que o agente terá no seu ambiente de acordo com as responsabilidades de que tiver sido encarregado.

Nesse modelo são representadas as percepções que o agente tem do ambiente: um evento ou as possíveis mensagens que ele poderá receber de outros agentes ou de entidades externas que irão disparar as suas ações. Existem duas formas de mapeamento dessas percepções para ações. A primeira é quando uma percepção é mapeada diretamente para uma ação. Nesse caso, o agente é considerado do tipo reativo. A outra forma é quando ocorre um planejamento na busca da ação adequada de acordo com a percepção do agente para atingir um objetivo. Nesse caso, o agente é considerado do tipo deliberativo e usa um motor de inferência no seu processo de raciocínio. Para realizar uma ação o agente pode necessitar de uma destreza (uma destreza é uma habilidade que o agente deverá possuir para a execução de suas responsabilidades) e também atender a uma determinada pré-condição. Ao termino de uma ação, uma pós-condição também pode ser necessária. Aqui os pontos de variação dos agentes em relação a responsabilidades variantes são os mesmos definidos no Modelo da Sociedade Multiagente.

É construído um Modelo de Ações do Agente para cada agente do modelo da sociedade multiagente, logo este é o modelo usado como base para a criação do modelo de ações do agente (Figura 28). Os conceitos de entidade externa, destreza, conhecimento e condição (não mais relacionados com as responsabilidades, mas sim com as ações.) são obtidos do modelo da sociedade multiagente. Os conceitos de “mensagens” e “eventos” buscam-se no modelo de

interações entre agentes. Os conceitos conhecimento e condição extraem-se também do modelo da sociedade multiagente, que aqui não estarão mais relacionados com responsabilidades, mas sim com ações. Os demais relacionamentos serão informados pelo usuário. Na Figura 59 ilustrada a rede semântica da modelagem das ações do agente e as regras 2.2.2 do apêndice A são utilizadas para automatizar estas tarefas.

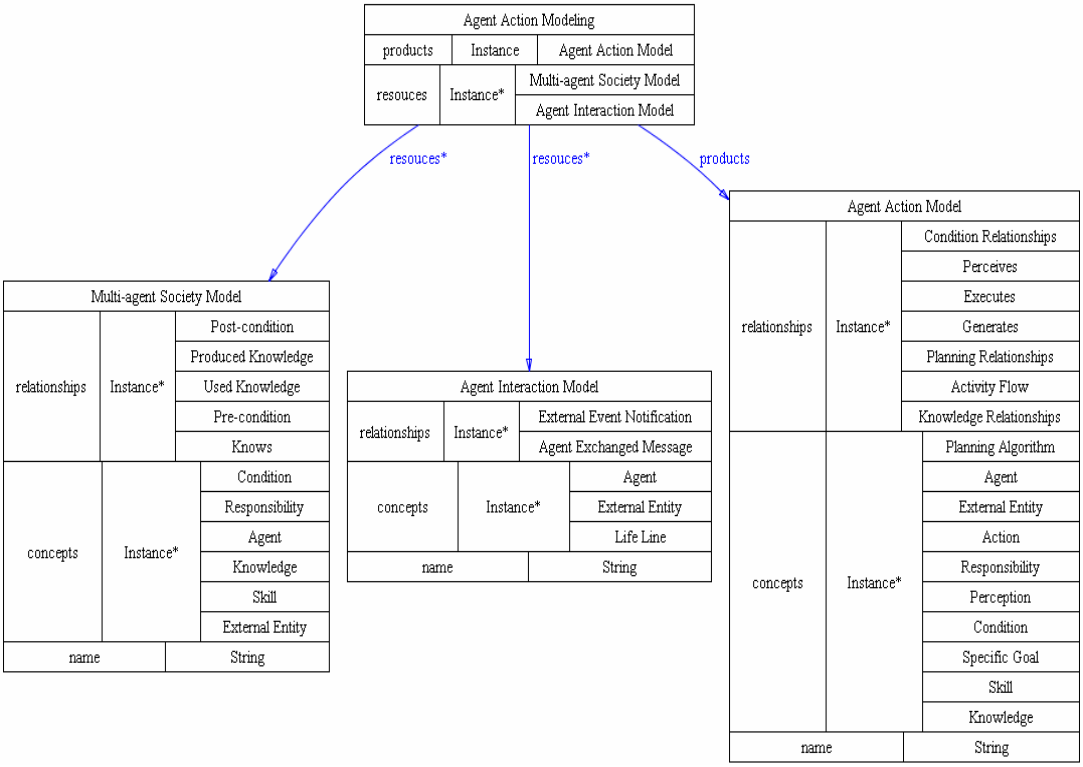

Figura 59 - Representação da subtarefa Modelagem das Ações do Agente do processo MADAE-Pro na ONTORMAS

Para cada agente pertencente ao modelo sociedade multiagente, deve-se informar o tipo do agente (reativo/deliberativo), através da tela da Figura 60. Se o agente for do tipo deliberativo deve-se criar o conceito de “Rule Engine” automaticamente.

A seguir solicita-se ao desenvolvedor que selecione e atribua a cada responsabilidade as ações em que ela deverá ser decomposta.

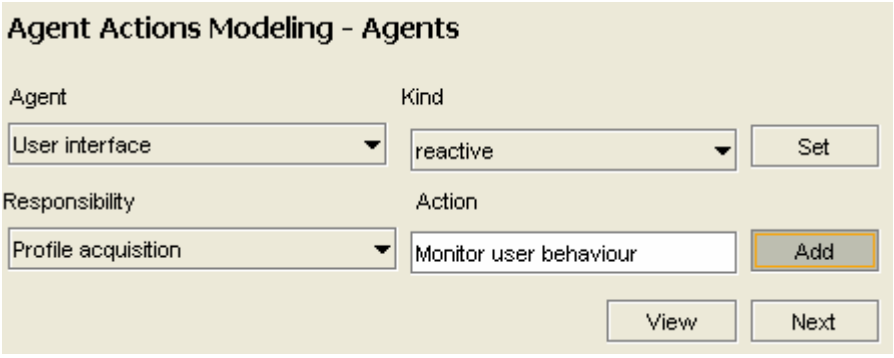

Figura 60 - Interface do ambiente MADAE-IDE para informar o tipo do agente no modelo de ações do agente

4.2.4.3 Fase de Implementação de Domínio

Na fase de Implementação do Domínio, a técnica DIMAS recomenda o mapeamento de modelos de projeto para agentes, comportamentos e atos de comunicação envolvidos no framework JADE, o qual tem sido adotado como plataforma de implementação. Um Modelo de Implementação da Sociedade Multiagente é construído como produto desta fase da MADEM, composto de um Modelo de Comportamentos e de um Modelo de Atos de Comunicação.

A instância da classe “Domain Implementation” conterá a representação de suas subtarefas através do relacionamento semântico “subtasks” e através do relacionamento semântico “products” o seu produto de modelagem (“Implementation Model of the Multi-agent Society”), que contém as representações dos produtos gerados em cada subtarefa, como mostra a rede semântica da Figura 61. Estas tarefas são automatizadas pelas regras 3.1 do apêndice A.

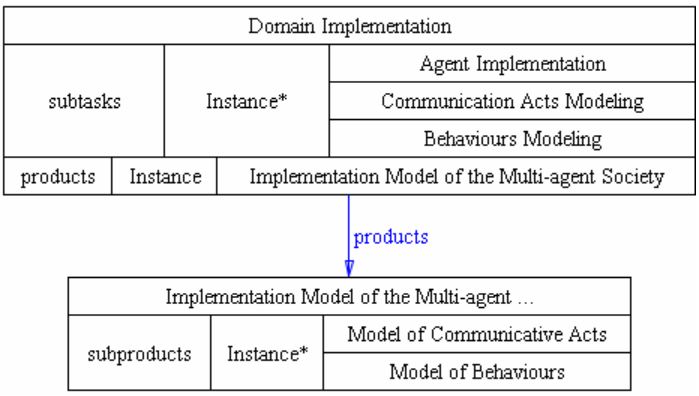

Figura 61 - Representação da subfase Implementação de Domínio do processo MADAE-Pro na ONTORMAS

4.2.4.3.1 Modelagem de Comportamentos e Atos de Comunicação

Na fase de Implementação do Domínio, os agentes são identificados a partir do modelo de ações do agente da fase anterior, sendo associado a cada uma de suas responsabilidades comportamentos em uma determinada linguagem/plataforma de desenvolvimento de agentes, através do assistente de modelagem visto na Figura 64, como o JADE. São ainda identificadas as interações entre agentes presentes no modelo de ações dos agentes, sendo estas mapeadas para uma determinada linguagem de comunicação entre agentes, como a FIPA-ACL.

Na Figura 62 a rede semântica da modelagem de comportamentos é ilustrada. Nesse metamodelo, o conceito “JADE Agent”, representa um agente JADE; “JADE Behaviour”, representa um comportamento JADE e “Method JADE Behaviour”, representa um método do comportamento JADE.

Na modelagem de Atos de Comunicação, as mensagens trocadas entre agentes presentes no Modelo de Interações entre Agentes são mapeadas para a linguagem FIPA-ACL (FIPA, 2008). As interações identificadas no Modelo de Interações entre Agentes são mapeadas para uma linguagem de comunicação entre agentes.

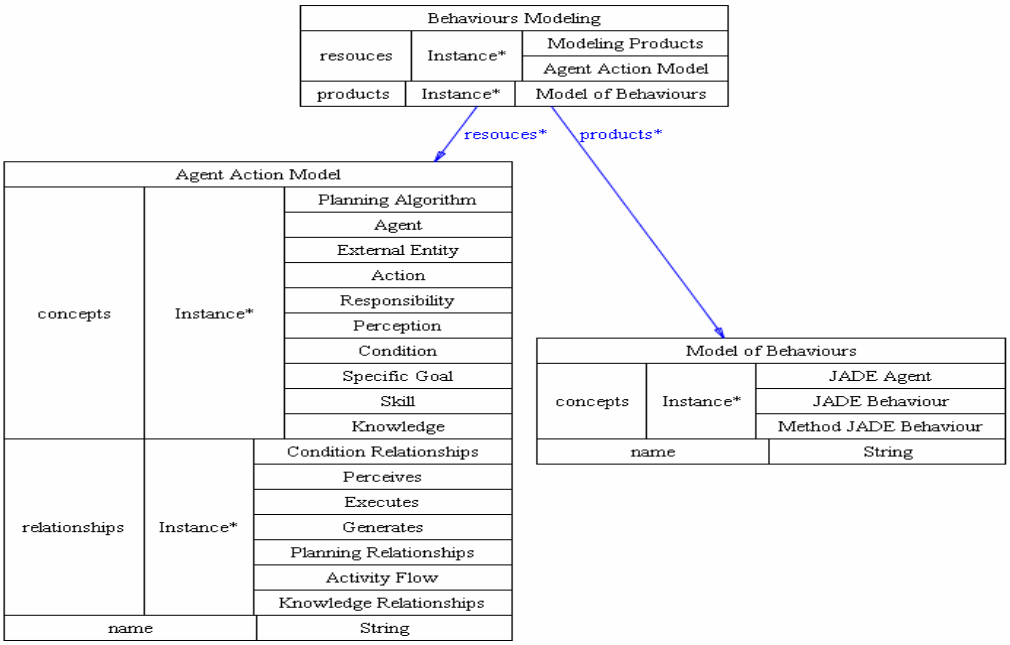

Figura 62 - Representação da subtarefa Modelagem de Comportamentos do processo MADAE-Pro na ONTORMAS

Na Figura 63 a rede semântica da modelagem de atos de comunicação é ilustrada. Nesse metamodelo o conceito “JADE Agent”, representa um agente JADE; “JADE Behaviour”, representa um comportamento JADE; “Method JADE Behaviour”, representa um método do comportamento JADE e “Performative”, representa uma performativa presente na especificação da linguagem de comunicação entre agentes FIPA-ACL.

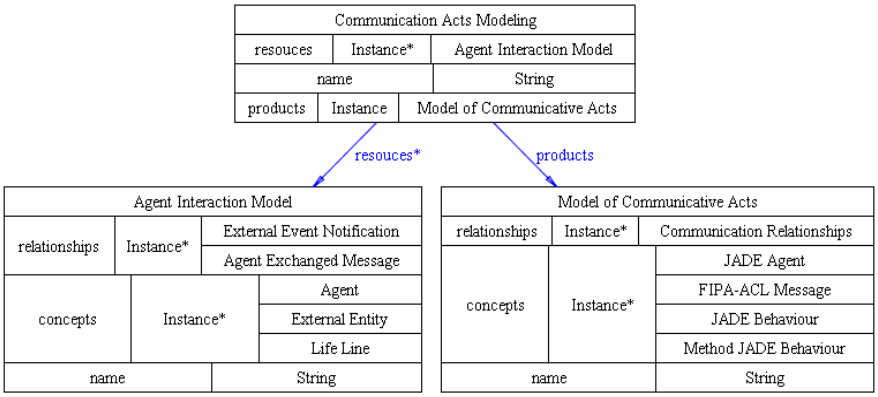

Figura 63 - Representação da subtarefa Modelagem dos Atos de Comunicação do processo MADAE-Pro na ONTORMAS

Para todo agente que pertence ao modelo de ações do agente, é adicionado este agente ao modelo de comportamento e ao modelo de atos de comunicação. Pelo assistente de modelagem visto na Figura 64 é gerado o modelo de comportamento através das regras 3.2 do apêndice A. O usuário deve selecionar o tipo de comportamento do Agente JADE e clicar em “Set”.

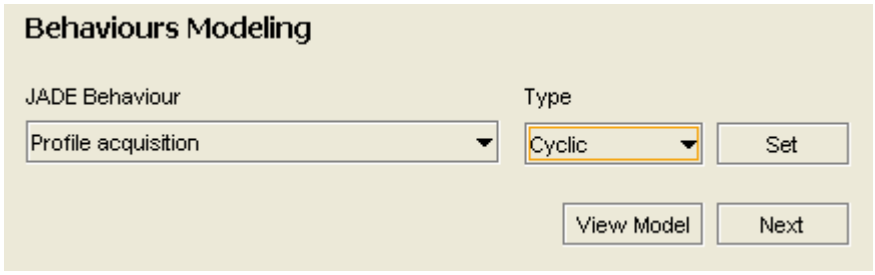

Figura 64 - Interface do ambiente MADAE-IDE para modelagem de comportamentos

Pelo assistente de modelagem visto na Figura 65 é gerado o modelo de atos de comunicação através das regras 3.3 do apêndice A. O usuário deve selecionar quais as mensagens trocadas entre os agentes e a quais os comportamentos JADE estão relacionados.

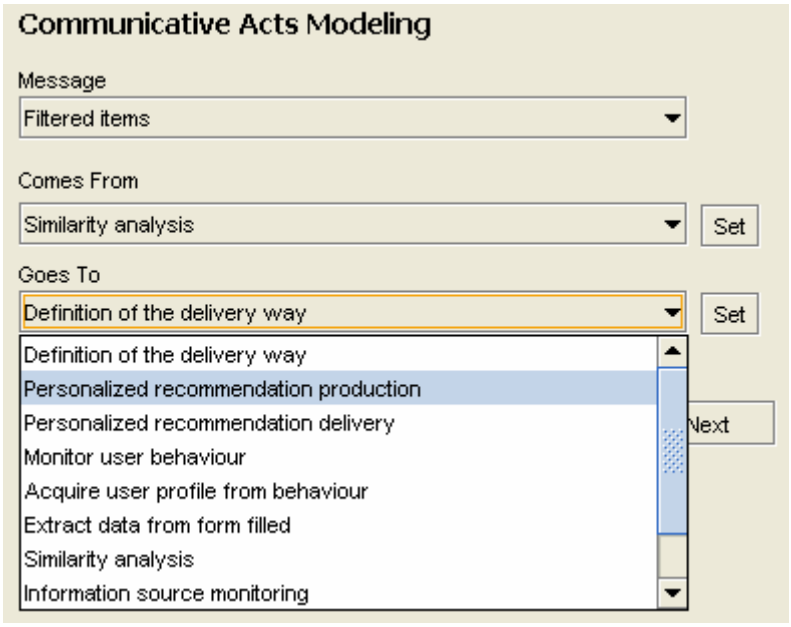

Figura 65 - Interface do ambiente MADAE-IDE para modelagem dos atos de comunicação

4.2.4.3.2 Implementação dos Agentes

A partir do Modelo de Comportamentos e do Modelo de Atos de Comunicação, a codificação de agentes genéricos a uma família de aplicações é realizada.

A

Tabela 9 descreve a correspondência entre os conceitos do modelo de implementação para a plataforma de agentes JADE/JESS. O agente, quando for do tipo reativo irá utilizar somente conceitos do framework JADE, enquanto que os do tipo deliberativo irão utilizar uma combinação dos frameworks JADE e JESS.

Tabela 9 - Correspondência entre conceitos de projeto e de implementação

	Design Concept	**JADE Concept**
Reative Agent	1 Agent	1 Agent Class
	1 Responsibility of each Agent	1 JADE Behaviour Class
	1 Action	Method of JADE Behaviour Class
	1 Multi-agent Knowledge Society Model	1 JADE Ontology (shared of all society agents)
	1 Knowledge instance of each Agent	1 Attribute of JADE Class
Deliberative Agent	**Design Concept**	**JADE/JESS Concept**
	1 Agent	1 Agent Class
	1 Responsibility of each Agent	1 JESS Rule
	1 Actions	JESS Rules

1 Multi-agent Knowledge Society Model	1 JADE Ontology (shared of all society agents)
1 Knowledge instance of each Agent	1 JESS Fact in JESS File

4.3 Considerações Finais

Considera-se que as principais características distintivas do ambiente MADAE-IDE é a agregação da ferramenta ONTORMAS ao ambiente e de regras de inferência para automatização de tarefas, reunindo as melhores soluções adotadas em apenas um ambiente. Sendo um plugin do Protégé, o ambiente é facilmente extensível e tem acesso direto à base de conhecimento ONTORMAS, podendo utilizar regras de inferência e busca semântica diretamente sobre suas instâncias.

Criar, associar e recuperar instâncias da ONTORMAS manualmente é um processo trabalhoso e suscetível a erros para o desenvolvedor que utiliza o processo MADAE-Pro, podendo tornar a base de conhecimento ONTORMAS inconsistente. Com a definição das regras de criação e de transformação de modelos, agregando o conhecimento sobre o ciclo de vida do processo MADAE-Pro, estas tarefas são automatizadas e os assistentes de modelagem garantem a sequencia correta de utilização do processo MADAE-Pro, produzindo assim modelos mais precisos na ontologia ONTORMAS e garantindo a qualidade de desenvolvimento, além de reduzir o tempo de modelagem, como será visto no próximo capítulo.

A instanciação e associação das fases com as tarefas e produtos de modelagem aconteceram de forma automática, sem intervenção do desenvolvedor. Antes estas ações eram de sua responsabilidade, o que poderia facilmente gerar inconsistências na ONTORMAS. Pôde ser verificado, também, o processo de transformação entre modelos, o que aumenta a produtividade e diminui o tempo de modelagem, pois evita retrabalho e a busca por instâncias na ONTORMAS de forma manual.

5 AVALIAÇÃO DO AMBIENTE MADAE-IDE

Este capítulo apresenta uma avaliação do ambiente MADAE-IDE através de sua utilização no desenvolvimento da família de sistemas multiagente ADVISE, para recomendação de informações baseada em conhecimento. A ONTOSERS (MARIANO et al, 2008) (MARIANO, 2008), uma família de sistemas de recomendações baseadas na tecnologia da Web Semântica, ou seja, uma sub-área do tópico recomendações baseadas em conhecimento, desenvolvida de forma manual, é utilizada como parâmetro de modelagem manual.

Inicialmente é feita uma comparação entre os artefatos criados em cada família. Em seguida, a avaliação é feita com base nas tarefas e produtos de modelagem instanciados considerando-se a produtividade e consistência. Para analisar-se a produtividade de desenvolvimento utilizando o ambiente MADAE-IDE, uma estatística foi extraída diretamente da ONTORMAS. Contabilizou-se o número de instâncias criadas e reusadas manual e automaticamente pelo MADAE-IDE na modelagem da família de aplicações ADVISE e o número de instancias criadas e reusadas manualmente na família ONTOSERS. A consistência foi analisada com base nas diretrizes de modelagem do MADAE-Pro (LEITE, 2009), através da verificação de instâncias de modelos, conceitos e relacionamentos semânticos na ONTORMAS. Ao fim do capítulo são apresentadas as considerações finais desta avaliação.

5.1 Instanciação das classes correspondentes às tarefas de modelagem

As instanciações das classes correspondentes às tarefas de modelagem são realizadas automaticamente pelo ambiente MADAE-IDE. Como exemplo, a Figura 66 ilustra a criação das instâncias das classes correspondentes à fase de análise de domínio, contendo as respectivas sub-tarefas e o produto da análise de domínio, com os respectivos subprodutos. Durante todo o desenvolvimento de uma família de aplicações multiagente pelo ambiente MADAE-IDE, os valores dos slots destas instâncias são obtidos através da interação do desenvolvedor com o ambiente.

Na modelagem da família de aplicações ONTOSERS isso não aconteceu. A fase de análise de domínio e suas sub-tarefas foram instanciadas, como pode ser visto na Figura 67, que exibe as instâncias de modelagem de domínio no repositório da ONTORMAS, mas os relacionamentos semânticos da fase com suas subtarefas não foram instanciados.

Por outro lado, a instanciação das classes correspondentes ao modelo de domínio, seus subprodutos e suas relações semânticas foi realizada, como visto na Figura 67. Percebe-se ainda que a classe correspondente ao produto da fase de prototipação da interface do usuário não foi instanciada. O mesmo ocorre com as instanciações das fases de projeto do domínio e implementação do domínio.

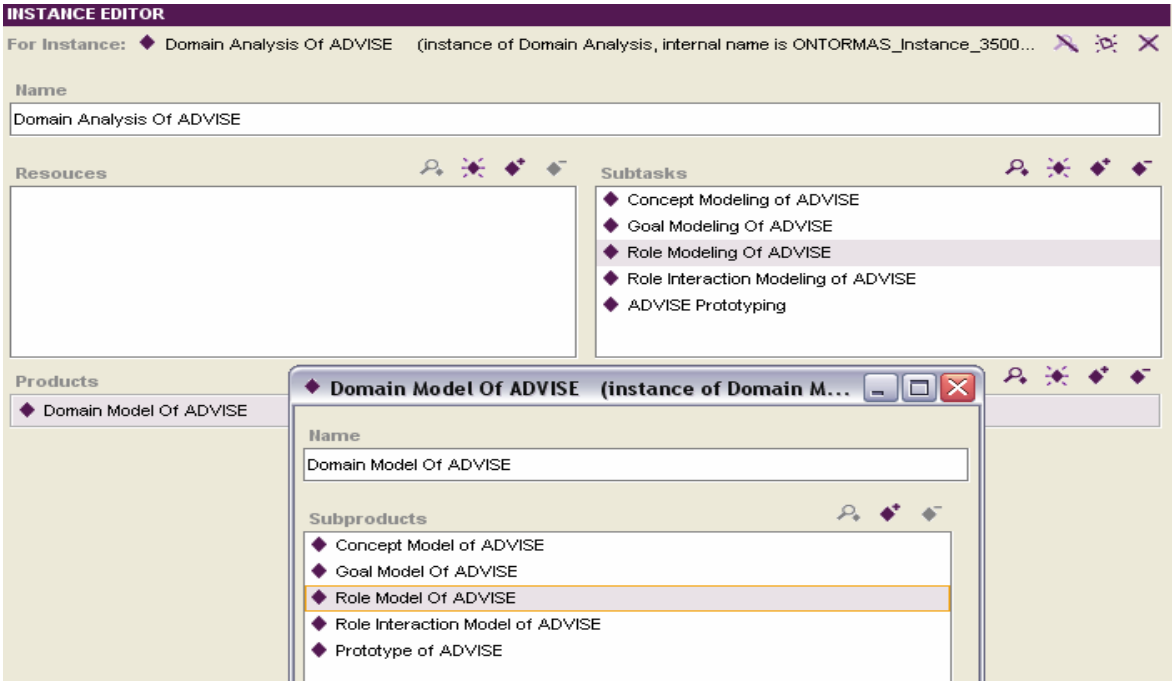

Figura 66 - Instâncias das classes correspondentes à Fase de Análise de Domínio da ADVISE

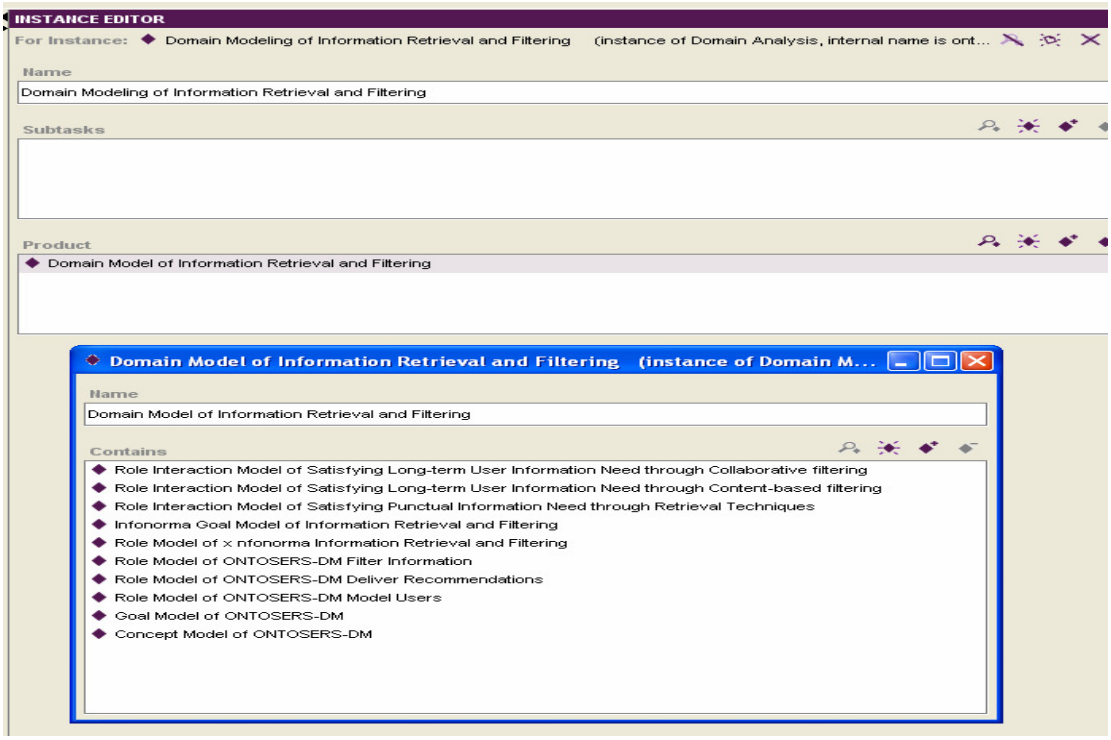

Figura 67 - Instâncias da Fase de Análise de Domínio da ONTOSERS

5.2 Geração e instanciação dos produtos de Modelagem

Na tarefa de modelagem de objetivos, o produto “Goal Model” da família ADVISE é gerado através da aplicação das regras de criação de modelos (seção 4.2.2.1) e da interação com o desenvolvedor através dos assistentes de modelagem. O modelo de objetivos da ONTOSERS é criado manualmente pelo desenvolvedor.

O objetivo geral de “Prover recomendações Usando a Tecnologia da Web Semântica” é atingido através dos objetivos específicos “Modelar Usuários”, “Filtrar Informação” e “Entregar as recomendações”.

Para atingir o objetivo específico “Modelar Usuário” é necessária a realização das responsabilidades “Aquisição Implícita dos Perfis”, “Aquisição Explícita dos Perfis” e “Criação e Atualização dos Modelos de Usuários Baseados em Instâncias de Ontologias”. A “Aquisição Implícita do Perfil” corresponde à obtenção do perfil do usuário sem que os seus interesses sejam explicitamente especificados, o que configuraria a “Aquisição Explícita dos Perfis”, a qual é uma forma intrusiva de obtenção do perfil do usuário. A “Criação e Atualização dos Modelos de Usuários Baseados em Instâncias de Ontologias” corresponde à manutenção da representação dos modelos de usuários.

O objetivo especifico “Modelar Usuários” tem um ponto de variação com grupo de responsabilidades para aquisição do perfil do usuário. Este ponto possui três variantes alternativas que correspondem às formas de adquirir o perfil do usuário: “Aquisição Implícita do Perfil”, “Aquisição Explícita do Perfil” ou ambas.

Para que o objetivo específico “Filtrar Informação” seja alcançado é preciso que se realize a responsabilidade “Criação e Atualização dos Modelos de Usuários em Instâncias de Ontologias”, que também contribui para atingir o objetivo específico de “Modelar Usuário”. Além dessa, são necessárias as responsabilidades “Agrupamento dos Modelos de Usuários”, “Representação em Instâncias Ontológicas dos Itens de Informação” e “Análise de Similaridade”. A responsabilidade “Agrupamento dos Modelos de Usuários” permite identificar grupos de usuários com interesses similares. A “Representação em Instâncias Ontológicas dos Itens de Informação” permite representar os itens de informação numa estrutura capaz de ser processada pelos agentes de software. A responsabilidade “Análise de Similaridade” consiste em determinar a relevância de um item para um determinado usuário.

Para alcançar o objetivo específico “Entregar recomendações” é pré-requisito que a responsabilidade “Análise de Similaridade” seja realizada, a fim de que haja itens de informação a serem entregues. Tendo sido realizado o pré-requisito há a execução das responsabilidades de “Produção de recomendações personalizadas” e “Entrega das recomendações Personalizadas”.

A Figura 68 e a Figura 69 ilustram os modelos das famílias de aplicações ADVISE e ONTOSERS, respectivamente.

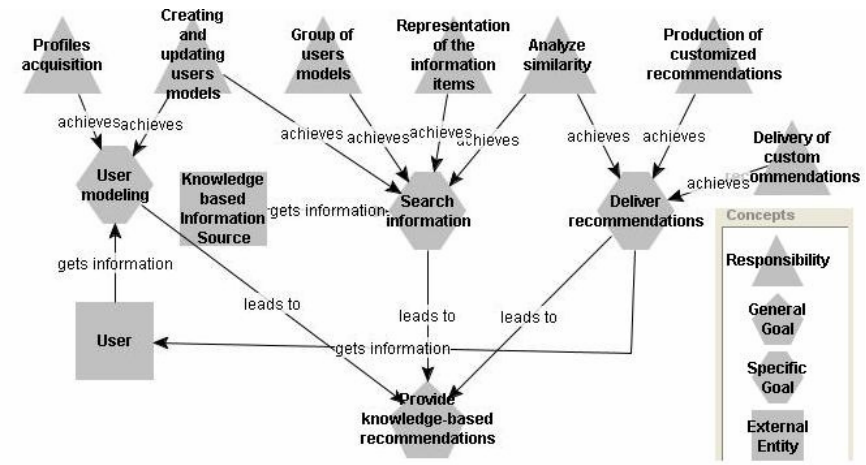

Figura 68 - Modelo de Objetivos da ADVISE

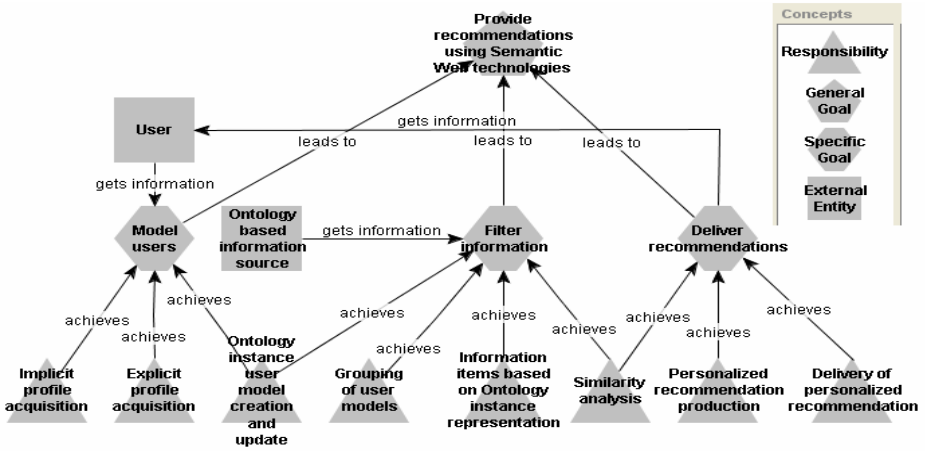

Figura 69 - Modelo de Objetivos da ONTOSERS

O produto “Role Model” da ADVISE é gerado parcialmente de forma automática através da aplicação das regras de transformação de modelos (Figura 70). Cabe ao desenvolvedor informar através dos assistentes de modelagem o papel, as pré e pós-condiçoes e os conhecimentos gerados e consumidos por cada responsabilidade.

A Figura 71 ilustra o modelo de papéis final para o objetivo específico modelagem de usuário, da família ADVISE, que pode ser comparado ao modelo da ONTOSERS equivalente, na Figura 72, criado manualmente pelo desenvolvedor.

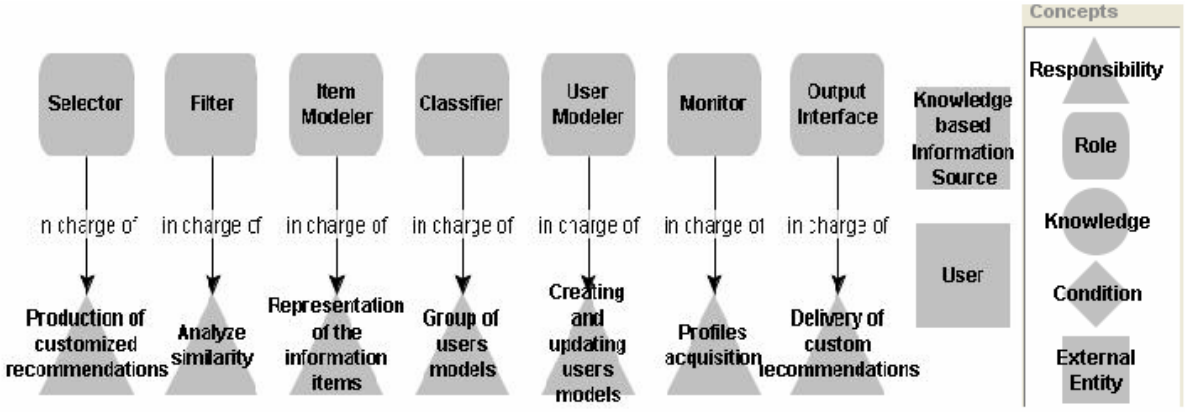

Figura 70 - Modelo de Papéis da ADVISE gerado pelo processo de transformação de modelos

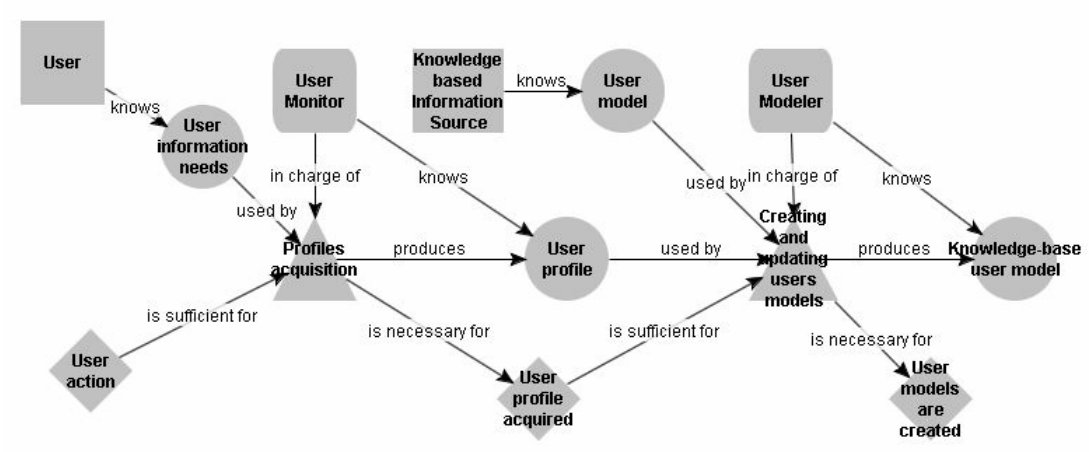

Figura 71 - Modelo de Papéis do Objetivo Específico "Modelagem de Usuário" da ADVISE

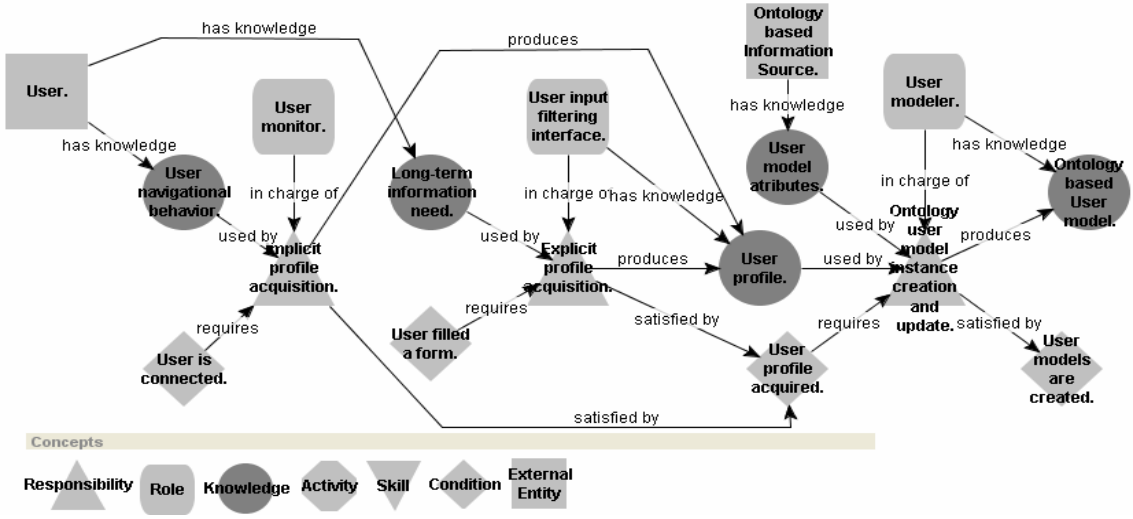

Figura 72 - Modelo de Papéis do Objetivo Específico "Modelar Usuário" da ONTOSERS

Os produtos da tarefa "Role Interaction Modeling" são totalmente gerados sem intervenção do usuário através da aplicação das regras de transformação de modelos do MADAE-IDE.

A Figura 73 exemplifica este modelo para a família ADVISE e a Figura 74 para a família ONTOSERS, este manualmente construído pelo desenvolvedor.

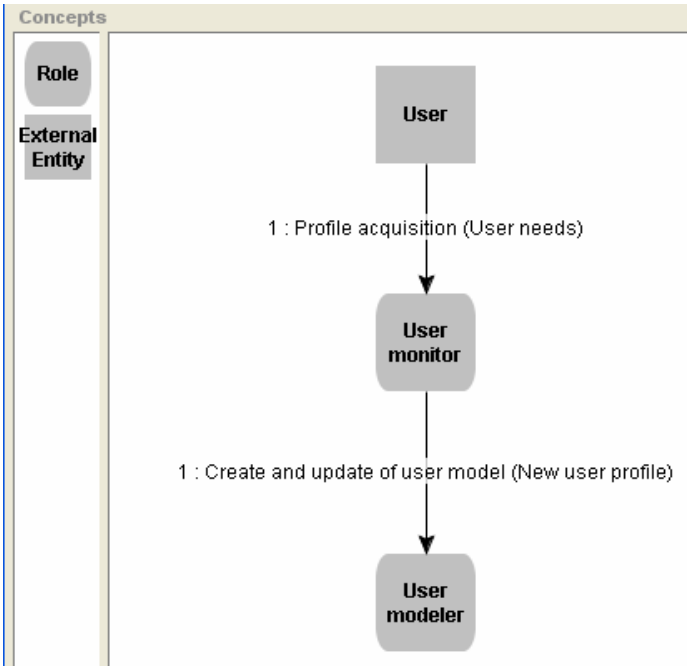

Figura 73 - Modelo de Interações entre Papéis do Objetivo Específico "Modelagem de Usuário" da ADVISE

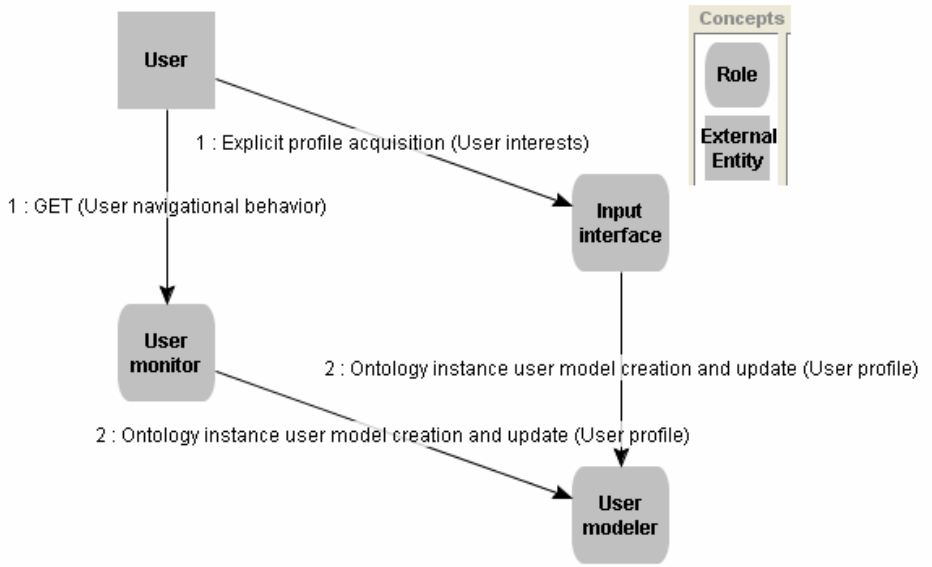

Figura 74 - Modelo de Interações entre Papéis do Objetivo Específico "Modelar Usuário" da ONTOSERS

Na fase projeto de domínio, o produto "Multi-agent Society Knowledge Model" é exibido com os conhecimentos já presentes, devendo o usuário montar a rede semântica. A Figura 75 ilustra o modelo da família ADVISE criado automaticamente através da aplicação das correspondentes regras de transformação, bastando o usuário criar as relações semânticas entre os conceitos

diretamente no modelo exibido para se obter o modelo equivalente visto na Figura 76, da família ONTOSERS, manualmente construído.

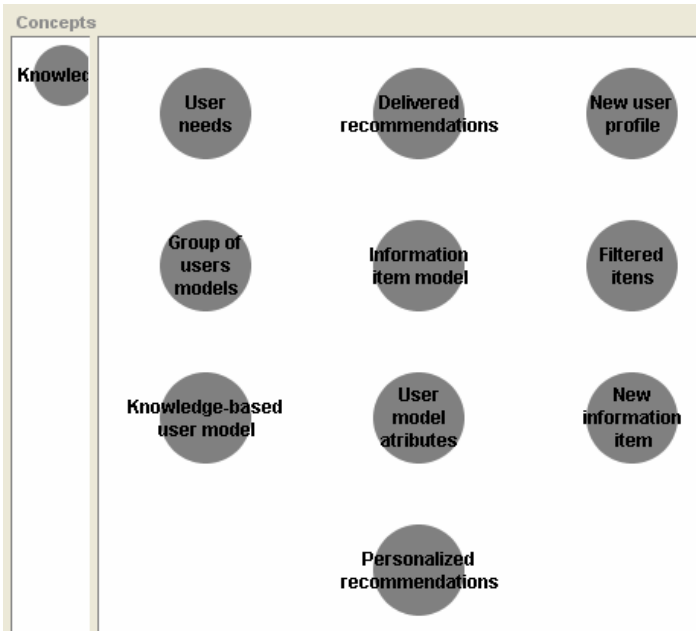

Figura 75 - Modelo de Conhecimento da Sociedade Multiagente da ADVISE

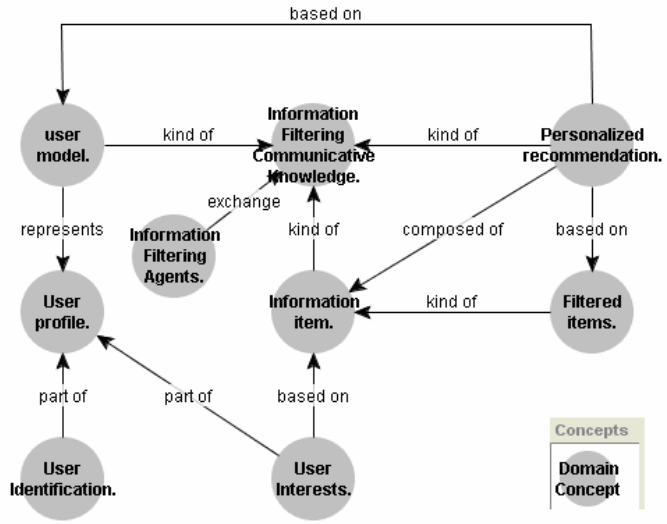

Figura 76 - Modelo de Conhecimento da Sociedade Multiagente da ONTOSERS

Na construção do “Multi-agent Society Model”, o desenvolvedor deve apenas informar os agentes e as destrezas relacionadas às responsabilidades. A Figura 77 ilustra o modelo gerado para a ADVISE e a Figura 78 o modelo criado manualmente para a ONTOSERS.

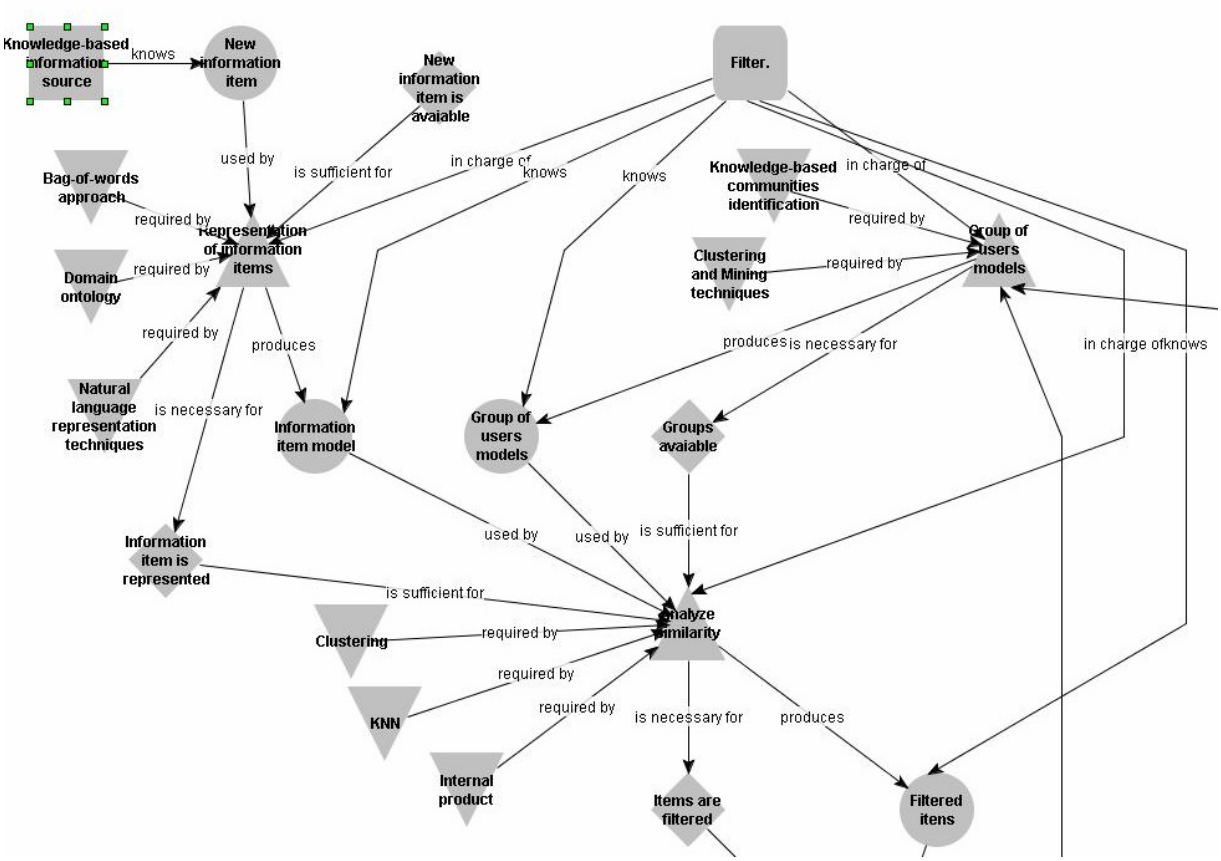

Figura 77 - Modelo da Sociedade Multiagente relacionadas ao agente "Filtrador" da ADVISE

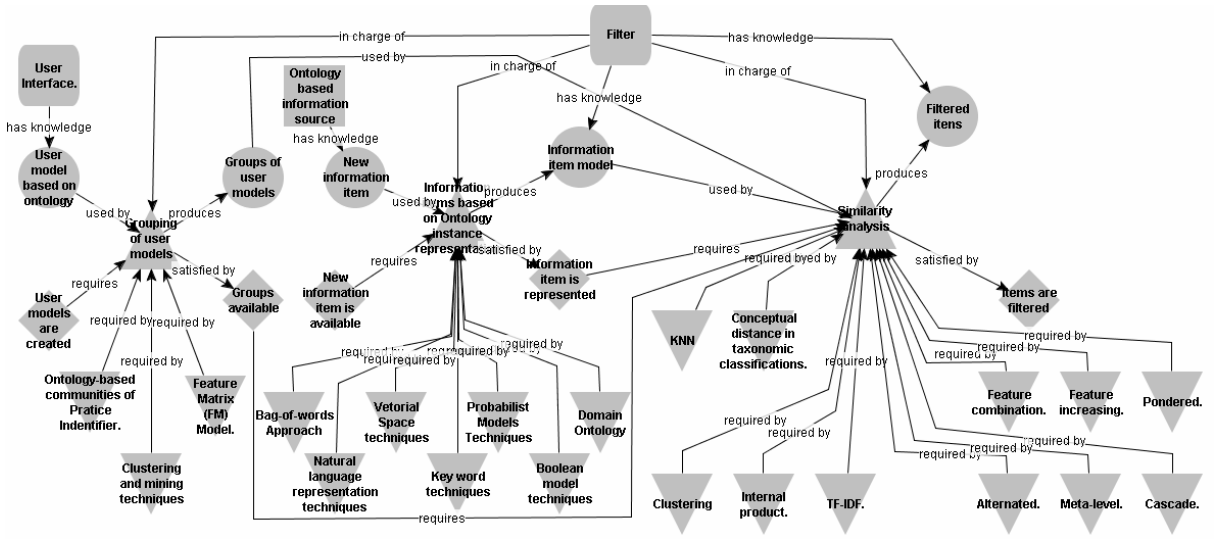

Figura 78 - Modelo da Sociedade Multiagente relacionadas ao agente "Filtrador" da ONTOSERS

O modelo de interação entre agentes é totalmente gerado sem intervenção do desenvolvedor. A Figura 79 exemplifica este modelo para a família ADVISE e a Figura 80 para a família ONTOSERS.

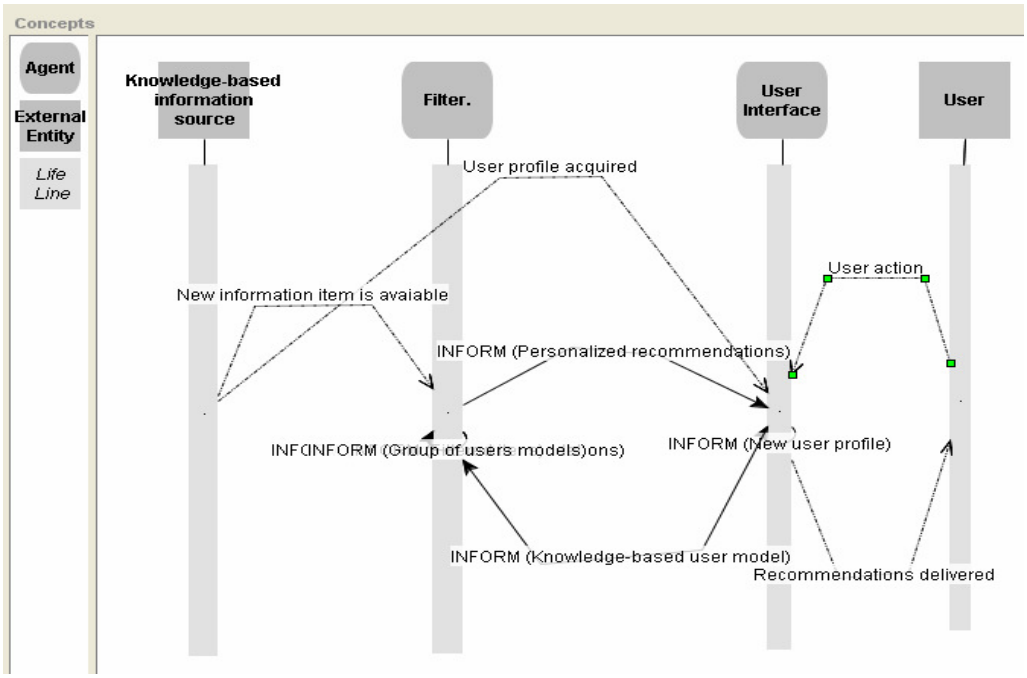

Figura 79 - Modelo de Interação entre Agentes da ADVISE

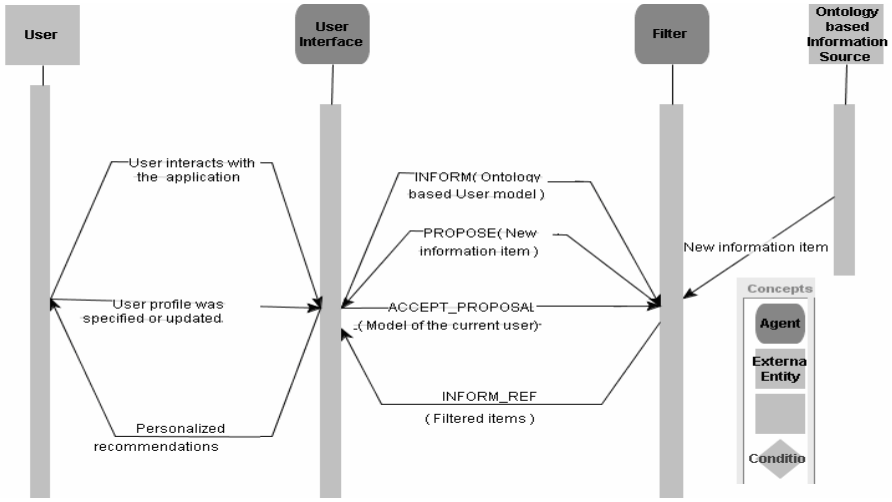

Figura 80 - Modelo de Interação entre Agentes da ONTOSERS

Para desenvolver o “Coordination and Cooperation Mechanisms Model”, o desenvolvedor informou que a arquitetura de cooperação e coordenação entre agentes é em camadas e a seguinte associou os agentes à sua respectiva camada. A seguinte, ele informa as relações semânticas existentes entre as camadas e entre as camadas e as entidades externas. A Figura 81 exemplifica este modelo para a família ADVISE e a Figura 82 para a família ONTOSERS, que foi desenvolvido manualmente.

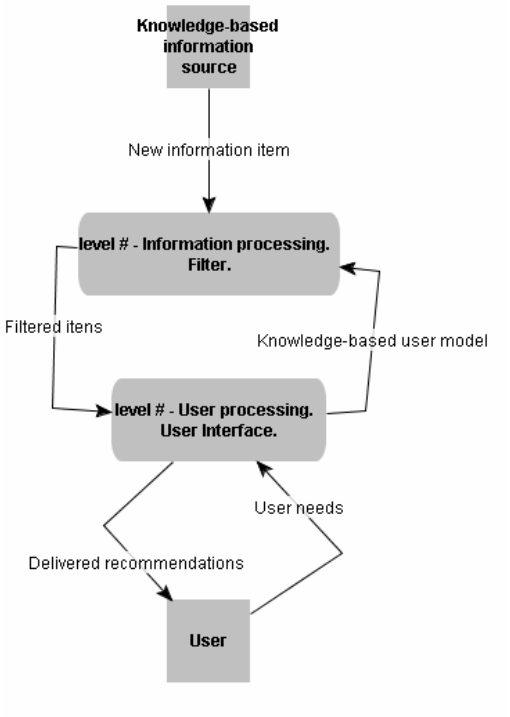

Figura 81 – Modelo dos Mecanismos de Coordanação e Cooperação da ADVISE

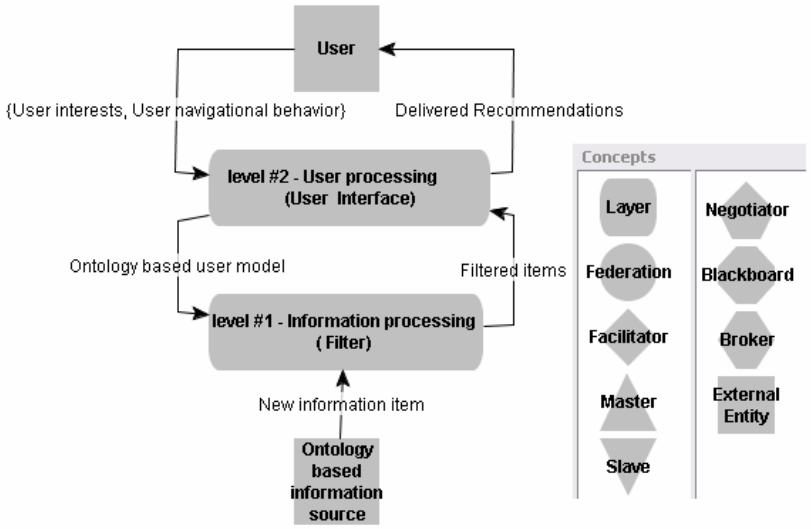

Figura 82 - Modelo dos Mecanismos de Coordanação e Cooperação da ADVISE

Para desenvolver o “Agent Knowledge Model”, o desenvolvedor cria a rede semântica da mesma forma que na construção do “Multi-agent Society Knowledge Model”. A Figura 83 exemplifica este modelo para a família ADVISE e a Figura 84 para a família ONTOSERS.

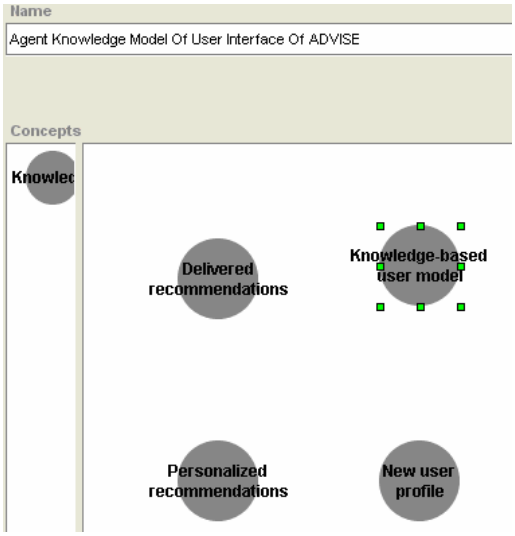

Figura 83 - Modelo do Conhecimento do Agente "User Interface" da ADVISE

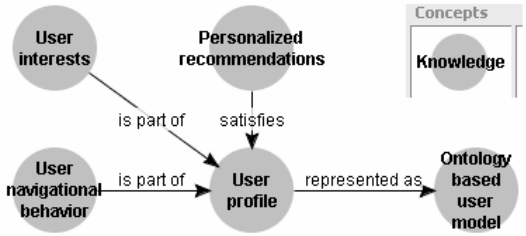

Figura 84 - Modelo do Conhecimento do Agente "User Interface" da ADVISE

O usuário deve agora informar o tipo de cada agente (reagente/deliberativo) e as ações relacionadas a cada responsabilidade em “Agent Actions Model”. Este modelo não existia quando da criação da família ONTOSERS. A Figura 85 ilustra este modelo.

Na Modelagem de Comportamentos é analisado o modelo de ações do agente, a fim de definir como os agentes atuarão. Nesta fase os agentes são mapeados para seus comportamentos, em classes do framework JADE. A partir do Modelo de Ações do Agente, cada agente foi mapeado para um agente JADE. Cada responsabilidade foi mapeada para uma classe JADE Behaviour e cada atividade para um método da classe Method JADE Behavior respectiva.

A Figura 86 ilustra o modelo de comportamentos do agente interface do usuário para a família ADVISE e a Figura 87 ilustra o mesmo modelo na família ONTOSERS.

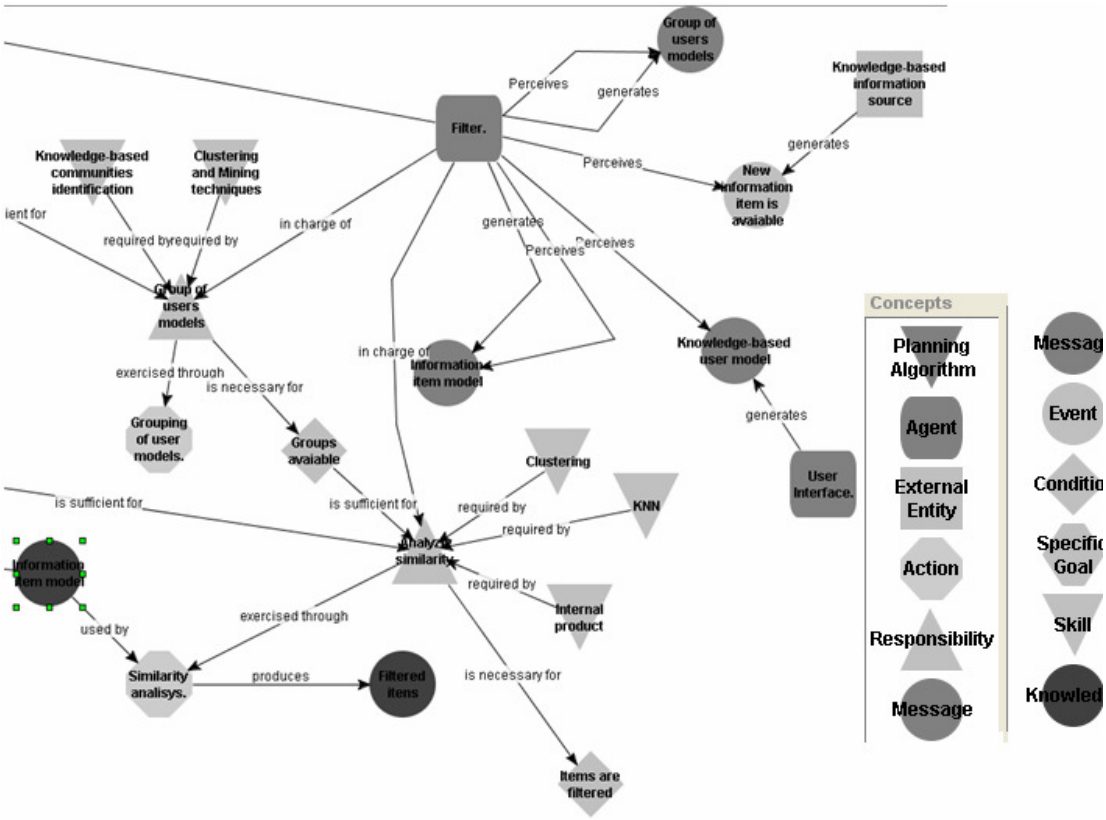

Figura 85 – Modelo de Ações do Agente Filtrador da ADVISE

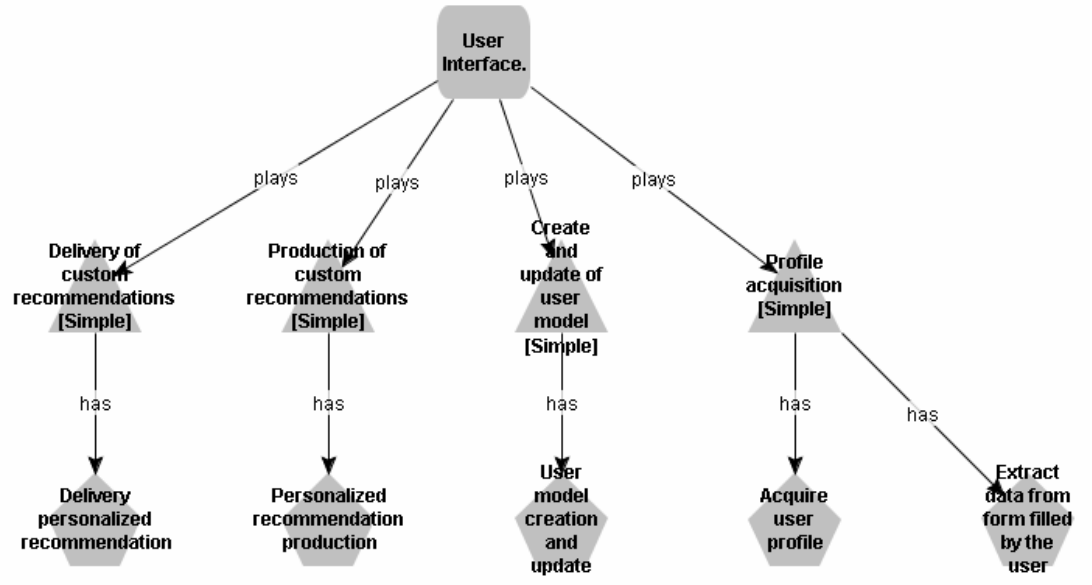

Figura 86 - Modelo de Agentes e Comportamentos do agente "Interface do Usuário" na ADVISE

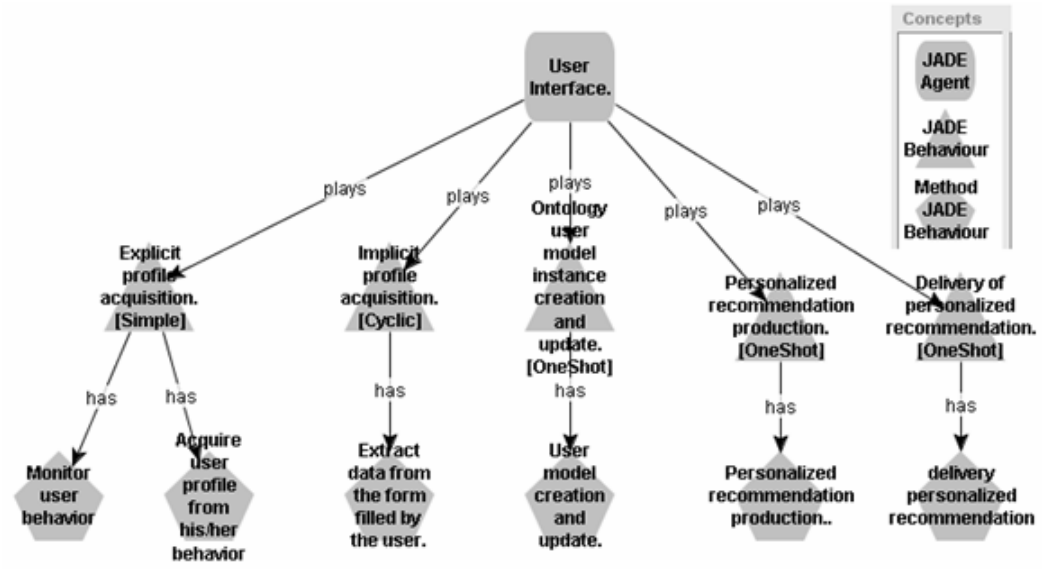

Figura 87 - Modelo de Agentes e Comportamentos do agente "Interface do Usuário" na ONTOSERS

Na modelagem dos atos de comunicação são especificadas as diretivas de comunicação trocadas entre os agentes, mapeados de cada Modelo de Interação do Agente. O Modelo de Atos de Comunicação determina as mensagens enviadas. Por exemplo, a mensagem "PROPOSE (Novo item de informação)" é gerada no método "Representação de novos itens informações como ontologia" e é recebida pelo método "Avaliar as novas itens de informações proposto", como mostra a Figura 88 da família ADVISE e Figura Figura 89 para a família ONTOSERS, manualmente construído.

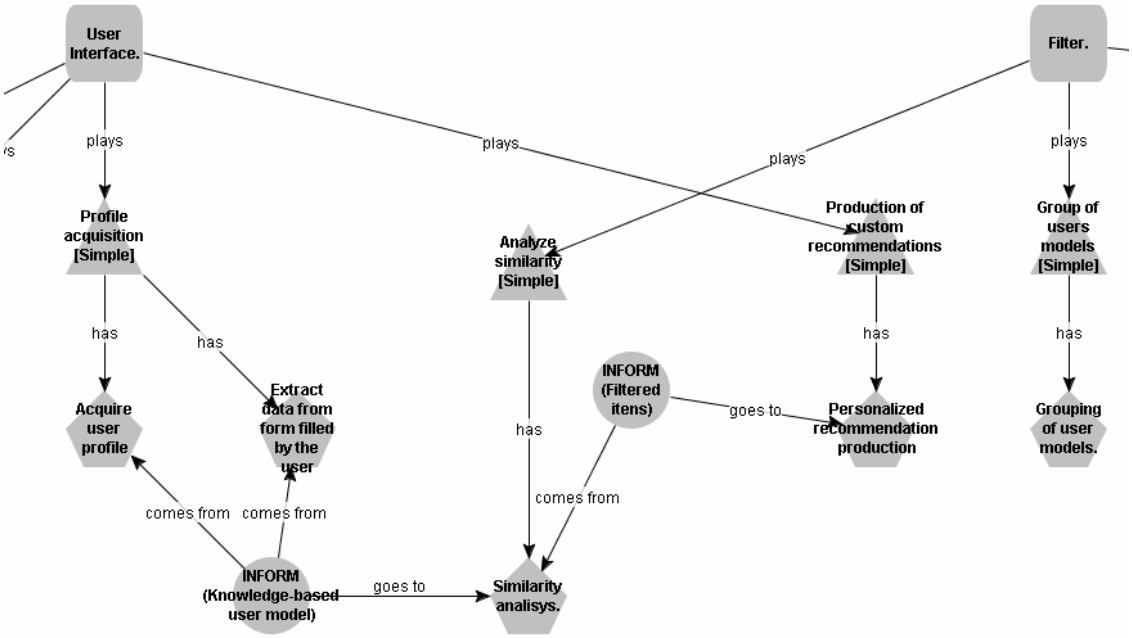

Figura 88 - Modelo de Atos de Comunicação da ADVISE

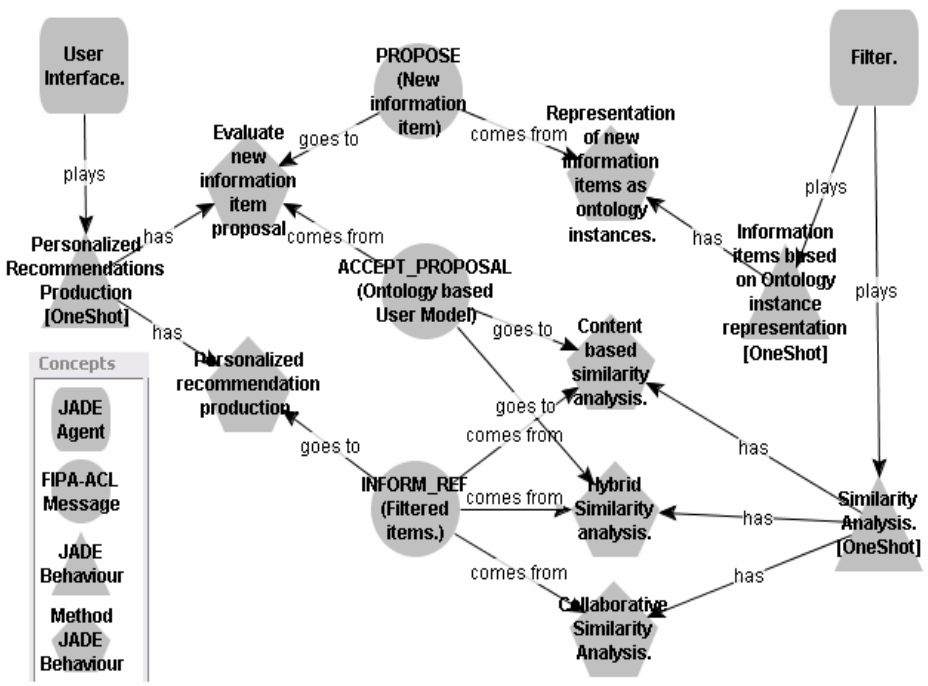

Figura 89 - Modelo de Atos de Comunicação da ONTORMAS

5.3 Análise dos ganhos de produtividade

Nos modelos iniciais do subprocesso de Engenharia de Domínio, os conceitos de modelagem e as relações semânticas nos modelos de conceitos e objetivos são totalmente capturados através da interação do desenvolvedor com o ambiente. Para estes modelos, na construção da família ADVISE, a informação é capturada através do respectivo assistente de modelagem e instanciado na ONTORMAS pela inferência de um conjunto de regras de criação (como visto no Capítulo 4). Logo, o esforço de desenvolvimento é equivalente àqueles requeridos para a construção destes modelos na família ONTOSERS, onde os conceitos de modelagem e relações semânticas são capturados diretamente do desenvolvedor para gerar as instâncias correspondentes, usando o editor gráfico da ONTORMAS.

A diminuição do esforço do desenvolvedor utilizando o MADAE-IDE é realmente alcançado a partir da modelagem de papéis, através da ativação das regras de transformação, no qual se consegue o reuso automático dos conceitos de modelagem e das relações semânticas dos modelos previamente criados.

A Tabela 10 resume o total de instâncias criadas e reusadas manual e automaticamente em cada fase de modelagem da família de aplicações ADVISE

utilizando-se o ambiente MADAE-IDE e a Tabela 11 o mesmo total para a ONTOSERS. Na fase de análise de domínio nota-se a predominância de maior esforço de modelagem por parte do desenvolvedor. Ao passar para as fases de projeto e implementação de domínio, o ambiente MADAE-IDE passa a criar e reusar automaticamente mais instâncias do que o desenvolvedor, de forma manual.

À medida que se avança nas fases de modelagem do MADAE-Pro utilizando o ambiente MADAE-IDE, o esforço do usuário na instanciação e reuso dos artefatos do processo diminui. Enquanto na família de aplicações ONTOSERS foram reusadas apenas 99 instâncias (24%), no ambiente MADAE-IDE foi efetivado o reuso 224 vezes, evidenciando o ganho de produtividade conseguido com a sua utilização em 90%.

Tabela 10 - Total de instâncias criadas e reusadas na modelagem da família de aplicações ADVISE

Fases	Manual		Automático		TOTAL	
	Instâncias Criadas	Instâncias Reusadas	Instâncias Criadas	Instâncias Reusadas	Instâncias Criadas	Instâncias Reusadas
Análise de Domínio	79	20	24	22	103	42
Projeto de Domínio	16	5	95	159	111	164
Implementação de Domínio	6	0	30	18	36	18
TOTAL (%)	101	25	149	199	250 (100%)	224 (90%)

Tabela 11 - Total de instâncias criadas e reusadas na modelagem da família de aplicações ONTOSERS

Fases	ONTOSERS	
	Instâncias Criadas	Instâncias Reusadas
Análise de Domínio	195	59
Projeto de Domínio	146	30
Implementação de Domínio	62	10
TOTAL (%)	403 (100%)	99 (24%)

5.4 Análise de consistência dos modelos

Vários problemas de consistência nos produtos integrantes da família de aplicações ONTOSERS foram detectados, resultantes de erros de modelagem na aplicação manual do processo MADAE-Pro, tornando a base de conhecimento ONTORMAS inconsistente. Na base de conhecimento só a classe correspondente à fase de Análise de Domínio foi instanciada, sem as instâncias correspondentes às tarefas de modelagem criadas e relacionadas (Figura 66).

Muitos conceitos de modelagem e relacionamento semânticos presentes na base de conhecimento não foram utilizados, como mostra a Figura 90, onde relacionamentos do tipo "Produced Knowledge" estão instanciados e não são utilizados.

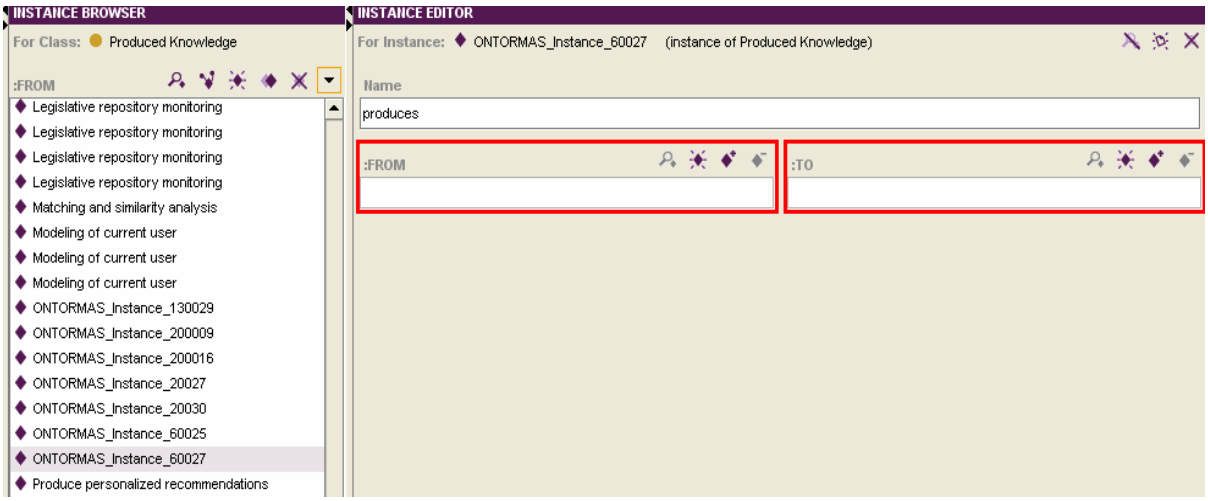

Figura 90 - Relacionamentos "Produced Knowledge" não utilizados

Foram também detectadas várias instâncias representando um mesmo conceito em diferentes modelos, enquanto apenas uma deveria ter sido criada e reutilizada. Por exemplo, como mostra a Figura 91, onde o conhecimento “Personalized recommendations” foi instanciado duas vezes. Este problema pode ser evidenciado na Tabela 11, pois na família ONTOSERS foram criadas 403 instâncias, e reusadas apenas 99, ao contrário da família ADVISE, onde o reuso foi efetivado 224 vezes e criadas 250 instâncias, apenas.

Este problema não existe na utilização do MADAE-IDE, pois o ambiente sempre realiza uma busca semântica antes da instanciação de um novo conceito de modelagem para descobrir se o mesmo já foi instanciado anteriormente.

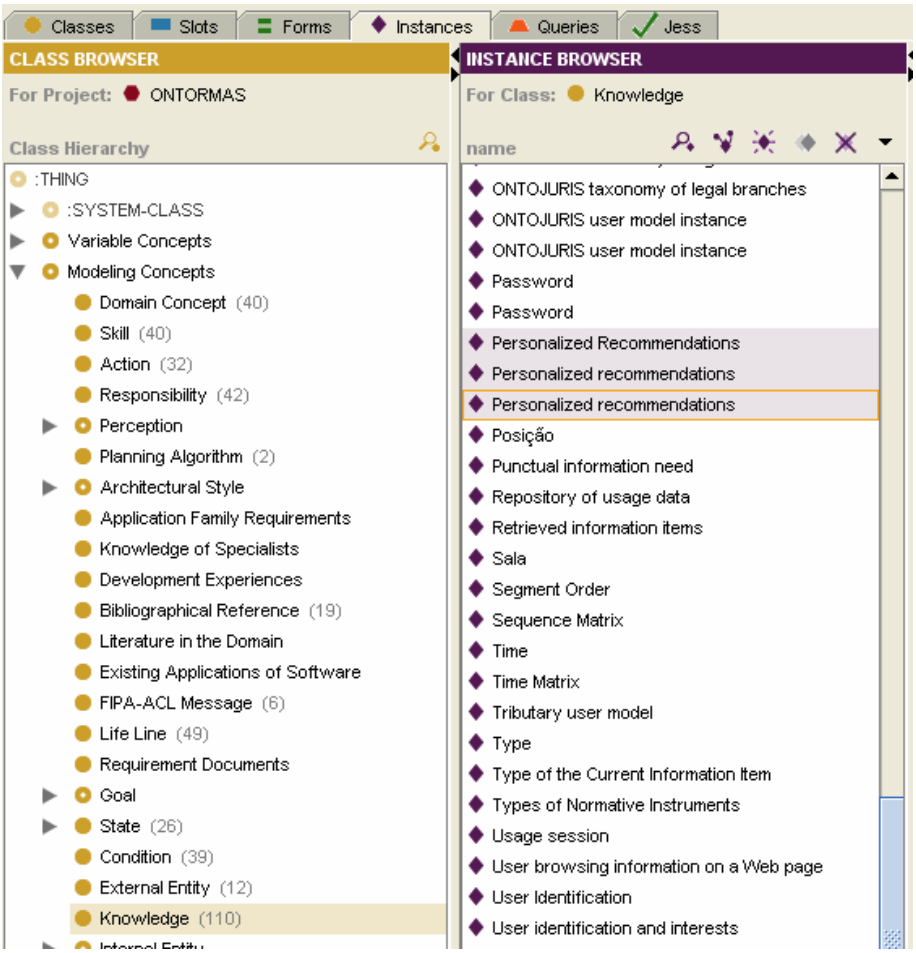

Figura 91 - Conhecimento "Personalized Recommendations" duplicado

Problemas de completude foram encontrados nos produtos de modelagem, como em “Role Model of ONTOSERS-DM Deliver Recommendations”. Os papéis “Item modeler”, “Content-based Filter”, “Colaborative Filter” e “Hybrid Filter” deveriam estar relacionados com os respectivos conhecimentos produzidos pelas responsabilidades a eles relacionados, como se vê na Figura 92.

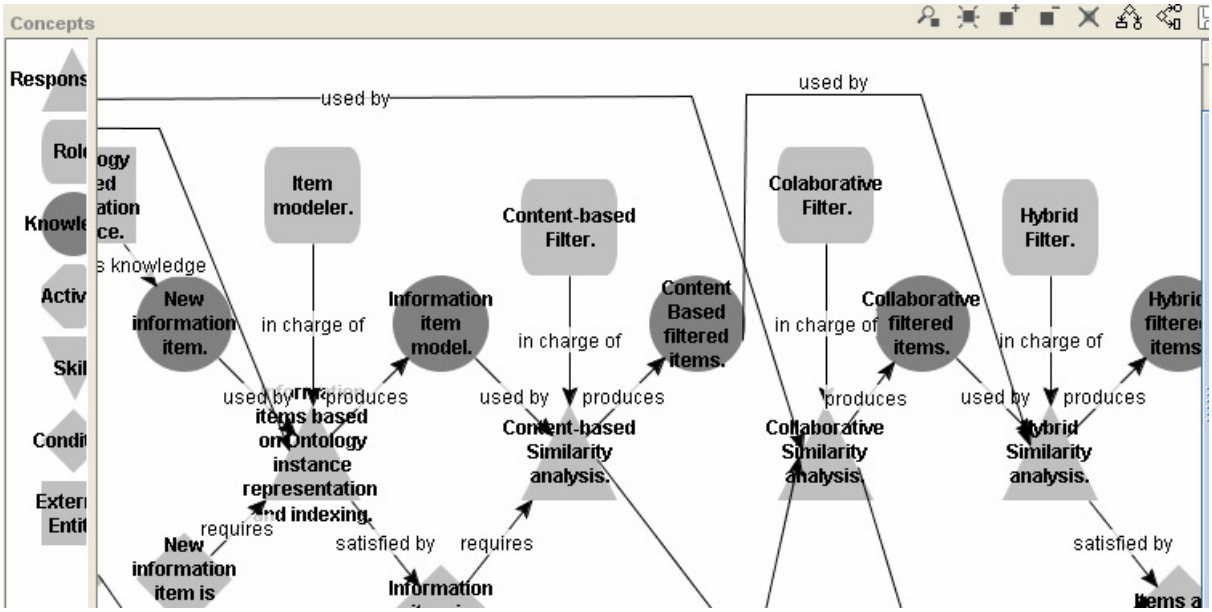

Figura 92 – Modelo de Papéis inconsistente da família ONTOSERS

5.5 Considerações finais

O desenvolvimento da família de aplicações permitiu uma primeira avaliação do ambiente MADAE-IDE. À medida que se avança na modelagem de uma família de aplicações utilizando-se o MADAE-IDE o desenvolver tem um ganho de produtividade, com vários modelos criados automaticamente e muitos conceitos de modelagem reusados.

A avaliação também revelou problemas de consistência e completeza por causa da modelagem manual da família ONTOSERS, que foram corrigidos com a utilização do MADAE-IDE.

6 CONCLUSÕES

Este trabalho descreveu o MADAE-IDE, um ambiente de desenvolvimento de software baseado no conhecimento para o desenvolvimento de sistemas multiagente. Os artefatos de software definidos no processo MADAE-Pro são representados como instâncias de classes da base de conhecimento ONTORMAS, servindo de repositório para os artefatos de software reusáveis para o ambiente MADAE-IDE.

Antes do desenvolvimento do MADAE-IDE, o processo de desenvolvimento de software multiagente MADAE-Pro era apoiado pela ferramenta ONTORMAS-T, composta de uma linguagem gráfica e um editor para modelagem das tarefas de forma manual. A criação, associação e recuperação manualmente das instâncias na ONTORMAS mostrou-se um processo trabalhoso e suscetível a erros por parte do desenvolvedor, tornando a base de conhecimento muitas vezes inconsistente.

Utilizando o ambiente MADAE-IDE, um estudo de caso foi desenvolvido, onde a ADVISE, uma família de aplicações multiagente para recomendações baseadas no conhecimento foi desenvolvida e seus resultados comparados com a família ONTOSERS, previamente desenvolvida na ONTORMAS de forma manual.

Através dos assistentes de modelagem e as regras de criação e transformação, o conhecimento do processo MADAE-Pro é disponibilizado no ambiente MADAE-IDE, automatizando sua aplicação. Isso permite a geração automática ou semi-automática de grande parte dos produtos de modelagem do processo, reduzindo o esforço de modelagem, o tempo de desenvolvimento e melhorando a qualidade dos produtos de software desenvolvidos.

6.1 Resultados e Principais Contribuições

Este trabalho contribui com a análise, projeto, implementação e avaliação de um ambiente de desenvolvimento de software multiagente, que:

- Fornece um guia automatizado de todas as fases do processo MADAE-Pro para o desenvolvedor, através de assistentes de modelagem,

viabilizando o desenvolvimento de famílias de aplicações e o seu reuso na construção de aplicações específicas;

- Aumenta a produtividade do desenvolvedor, não sendo sua responsabilidade instanciar tarefas de modelagem ou efetuar pesquisas para reuso de instâncias da base de conhecimento;
- Seleciona, integra e adapta conceitos de modelagem de forma automática;
- Mantém a consistência e completude da ontologia ONTORMAS seguindo as diretrizes do processo MADAE-Pro;
- Geração automática de modelos, através da exploração da riqueza semântica fornecida pela ontologia ONTORMAS;
- Permite continuar a modelagem de uma família de aplicações, de forma automatizada, a partir do último modelo construído.
- Submissão e aceitação de um artigo na International Conference on Information Technology (ITNG) do ano de 2010 publicado pelo IEEE, entitulado: *An Overview of the MADAE-IDE Multi-Agent System Development Environment*.

6.2 Limitações

A atual versão do MADAE-IDE possui algumas limitações:

- Falta suporte à modelagem de variabilidades, em momentos específicos durante a modelagem de uma família de aplicações;
- A Engenharia de Aplicações não está implementada;
- Não integração do ambiente MADAE-IDE com o framework JADE para execução de testes dos agentes ao fim da modelagem;
- Inexistência de suporte de gerência de projetos no MADAE-IDE. As versões de um mesmo produto de software não possuem mecanismos de controle de versões. Atualmente, é realizado somente “backups” dos arquivos quando é necessário fazer alguma modificação nos modelos.

6.3 Trabalhos Futuros

Para resolver as atuais limitações do MADAE-IDE propõe-se a realização dos seguintes trabalhos:

- Desenvolver um módulo de controle dos pontos de variação nos modelos. Estes pontos de variação nos modelos fornecem o mecanismo para se adaptarem os artefatos das famílias de aplicações na construção de uma aplicação específica;
- Estender o ambiente para fornecer suporte na construção de aplicações específicas, para assim contemplar a Engenharia de Aplicações Multiagente.
- Desenvolver um módulo de teste para os agentes, integrado ao framework JADE;
- Integrar uma ferramenta que apóie o gerenciamento do projetos de software no MADAE-IDE, para que possa-se acompanhar as versões sucessivas dos artefatos na base de conhecimento.
- Incluir um sistema de controle de versões no MADAE-IDE, que forneça um histórico das modificações realizadas nos projetos de software e possibilite armazenar versões estáveis do projeto, evitando assim, o retrabalho em caso de eventuais erros na versão atual do projeto;
- Avaliação e refinamento do ambiente MADAE-IDE através do desenvolvimento de novos estudos de casos na construção de famílias de sistemas e aplicações multiagente específicas.
- Disponibilização do ambiente para o público em geral, através da criação de um site na Web.

REFERÊNCIAS

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

BELLIFEMINE, F., CAIRE, G., POGGI, A., RIMASSA, G. **JADE A White Paper.** Exp v. 3 n. 3, Sept., 2003.

BERTOLLO, G., Ruy, F. B., MIAN, P. G., PEZZIN, J, SHWAMBACH, M.M., NATALI, A.C.C., FALBO, R.A.. **ODE - Um Ambiente de Desenvolvimento de Software Baseado em Ontologias**. Anais do XVI Simpósio Brasileiro de Engenharia de Software - SBES'2002. Caderno de Ferramentas, pp.438-443,Gramado - RS, Brasil, Outubro 2002.

BITTENCOURT, G.. **Inteligência Artificial, Ferramentas e Teorias**. Editora da UFSC, Florianópolis. 1998.

BRENNER, W.; ZARNEKOW, R.; WITTING, H. **Intelligent Software Agents, Foundations and Applications**. Berlin, Alemanha, Springer Verlag. 1998.

CAVALCANTE, U. A. S., GIRARDI, R. **An Overview of the MADAE-IDE Multi-Agent System Development Environment**. Proceedings of the 7th International Conference on Information Technology : New Generations, Ed. IEEE. Las Vegas, Nevada, USA, 2010. A ser publicado.

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

CLANCEY,W. J. Situated action: **A neuropsychological interpretation (Response to Vera and Simon)**. Cognitive Science 17(1): 87-107. 1993.

CZARNECKI, K. **Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models**, Ph.D. Thesis, Department of Computer Science and Automation, Technical University of Ilmenau, 1998.

DAI, W. A. **A Multiagent Architecture for Semantic Web Resources**. Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology. 2005.

DRUMOND, L., GIRARDI, R., LEITE, A. Architectural **Design of a Multi-Agent Recommender System for the Legal Domain**, Eleventh International Conference on ARTIFICIAL INTELLIGENCE and LAW, Ed. ACM. a ser publicado. 04 a 08 de junho de 2007.

FALBO, R. A., GUIZZARDI, G. NATALI, A. C. C. **Towards Semantic Software Engineering Environments**. Proc. of Int. Conference on Software Engineering and Knowledge Engineering, SEKE'2002. Ischia, Italy. 2002.

FALBO, R. **Integração de Conhecimento em um Ambiente de Desenvolvimento de Software**. Tese de Doutorado . Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ. Dezembro de 1998.

FALBO, R.A., NATALI, A. C. C., MIAN, P. G., BERTOLLO, G., RUY, F. B., IX ODE: **Ontology-based software Development Environment**. Congreso Argentino de Ciencias de la Computación, p. 1124-1135 La Plata, Argentina, Outubro 2003.

FIPA. **ACL Message Structure Specification**. Disponível em: <http://www.fipa.org/specs/fipa00061/SC00061G.html>. Acessado em: 16 fev 2008.

FIPA. **Foundation for Intelligent Physical Agents**. Disponível em: <http://www.fipa.org/>. Acessado em: 15 jan. 2008.

FRIEDMAN-HILL E. **JESS in Action**, Manning Publications, 2003.

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.

GIMENES, I.M.S. **O Processo de Engenharia de Software: Ambientes e Formalismos**, Caxambu-MG: SBC, 1994. Trabalho apresentado na Jornada de Atualização em Informática. 1994.

GIRARDI, R., FARIA, C. **An Ontology-Based Technique for the Specification of Domain and User Models in Multi-Agent Domain Engineering.** CLEI Electronic Journal, V. 7, N. 1, pp. 7, 2004.

GIRARDI, R., FARIA, C., MARINHO, L. **Ontology-based Domain Modeling of Multi-Agent Systems**. Proceedings of the Third International Workshop on Agent-Oriented Methodologies at International Conference on Object-Oriented

Programming, Systems, Languages and Applications (OOPSLA 2004), Ed. Cesar Gonzalez-Perez, pp. 51--62. Vancouver, Canada. October 24th to 28th, 2004.

GIRARDI, R., LEITE. **A Knowledge-based Tool for Multi-Agent Domain Engineering**, Knowledge-Based Systems Journal, v. 21, Ed. Elsevier, pp. 604-611, 2008.

GIRARDI, R., LINDOSO, A. **DDEMAS: A Domain Design Technique for Multi-agent Domain Engineering**, Lecture Notes in Computer Science, Perspectives in Conceptual Modeling: ER 2005 Workshops CAOIS, BP-UML, CoMoGIS, eCOMO, and QoIS, Volume 3770/2005, ISSN 0302-9743., Ed. Springer-Verlag GmbH, pp. 141-150. Klagenfurt, Austria, 2005.

GIRARDI, R., MARINHO, L. **A Domain Model of Web Recommender Systems based on Usage Mining and Collaborative Filtering**, Requirements Engineering Journal, vol.12 n. 1, Ed. Springer-Verlag, pp. 23-40. London, 2007.

GÓMEZ PÉREZ, A. F. (2004). **Ontological Engineering**. London: Springer-Verlag.

GRUBER, T. **Towards Principles for the Design of Ontologies Used for Knowledge Sharing**. In: *International Journal of Human-Computer Studies*,1995.

GUIZZARDI, G. ; ALMEIDA, J. P. A. ; GUIZZARDI, R. S. S. ; FALBO, R. A. . **Ontologias de Fundamentação e Modelagem Conceitual**. In: II Seminário de Pesquisa em Ontologia no Brasil, 2009, Rio de Janeiro. II Seminário de Pesquisa em Ontologia no Brasil, 2009.

GUIZZARDI, G., FALBO, R. A. and FILHO, J. G. P. **Using Objects and Patterns to Implement Domain Ontologies**. in Proc. of the 15th Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, 2001. HAPPEL, H. SEEDOF, S. Applications of Ontologies in Software Engineering, 2nd Int. Workshop on Semantic Web Enabled Software Engineering (SWESE 2006).

GUIZZARDI, Giancarlo ; FALBO, R. A. ; Guizzardi, R.S.S. . **Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology**. In: 11th Iberoamerican Workshop on Requirements Engineering and Software Environments, 2008, Recife. Proceedings of the 11th Iberoamerican Workshop on Requirements Engineering and Software Environments, 2008. p. 127-140.

JADE - **Java Agent DEvelopment Framework**, Endereço: http://jade.tilab.com/, Acessado em 23 de fevereiro de 2008.

KALYANPUR, A. J. **Automatic Mapping of OWL Ontologies into Java**. Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering. 2004.

KRUEGER, C. W. 1992. **Software reuse**. ACM Comput. Surv. 24, 2, 131-183. 1992.

LEITE, A. C., MADAE-Pro: **Um processo baseado no conhecimento para engenharia de domínio e de aplicações multiagente.** Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação) - Universidade Federal do Maranhão, São Luís, 2009.

LEITE, A. GIRARDI, R. CAVALCANTE, U.: **An Ontology for Multi-Agent Domain and Application Engineering**. In: The 2008 IEEE International Conference on Information Reuse and Integration (IEEE IRI-08), (pp. 98-103). Las Vegas. Proceedings of the 2008 IEEE International Conference on Information Reuse and Integration, 2008.

LEITE, A., GIRARDI, R. CAVALCANTE, U. **MAAEM: A Multi-agent Application Engineering Methodology,** Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), Knowledge Systems Institute, Redwood City, Boston, 2008b.

LEITE, A., GIRARDI, R., **A Process for Multi-Agent Domain and Application Engineering: the Domain Analysis and Application Requirements Engineering Phases**, Proceedings of the 11th International Conference on Enterprise Information Systems, Ed. INSTIIC, pp. 156-161. Milan, Italy, 2009.

LINDOSO, A. GIRARDI, R. **Uma Técnica baseada em Ontologias para o Reuso de Padrões de Software e de Frameworks no Projeto de Aplicações Multiagente**, First Workshop on Software Engineering for Agent-oriented Systems (SEAS 2005), 19º Simpósio Brasileiro de Engenharia de Software (XIX SBES). Uberlândia, Minas Gerais, Brasil, 2005.

LINDOSO, A. GIRARDI, R. **The SRAMO Technique for Analysis and Reuse of Requirements in Multi-agent Application Engineering**, IX Workshop on

Requirement Engineering, Cadernos do IME, v. 20, Ed. UERJ, pp. 41-50. Rio de Janeiro, 2006.

LINDOSO, A. **Uma metodologia baseada em ontologias para a engenharia de aplicações multiagente.** Dissertação (Mestrado em Engenharia de Eletricidade – Área de Ciência da Computação) - Universidade Federal do Maranhão, São Luís, 2006.

MARIANO, R., **Desenvolvimento de agentes de filtragem híbrida utilizando técnicas de aprendizagem de máquina e a tecnologia da Web semântica**. Dissertação de Mestrado, UFMA. 2008.

MARIANO, R., Girardi, R., Leite, A., Drumond, L. Maranhão, D. **A Case Study on Domain Analysis of Semantic Web Multi-agent Recommender Systems**. In: Proceedings 3th International Conference on Software and Data Technologies, Porto. Portugal, 2008.

MI, P. **A knowledge-based environment for modeling and simulating softwareengineering processes**. Knowledge and Data Engineering, IEEE Transactions on Volume 2, Issue 3, (pp. 283 – 289). 1990.

MIAN, P.G., FALBO, R.A., **Supporting Ontology Development with ODEd**, Proceedings of the 2nd Iberoamerican Symposium on Software Engineering and Knowledge Engineering, Salvador, Brazil, 2002.

MOTTA, E. **The knowledge modelling paradigm in knowledge engineering**. In: Handbook of Software Engineering and Knowledge Engineering (Vol. 1). In Chang, SK, Singapore: World Scientific. 2001.

PEZZIN, J., FALBO, R.A. **AgeODE: Uma Infra-estrutura para Apoiar a Construção de Agentes para Atuarem no Ambiente de Desenvolvimento de Software ODE**. Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, JIISIC'2004,Volumen II, pp. 389-404,Madrid, España, Noviembre 2004.

PRIETO-DÍAZ, R. **Status Report: Software Reusability**. *IEEE Softw.* 10, 3 (May. 1993), 61-66. 1993.

REIS, C. A. L. **Ambientes de Desenvolvimento de Software e seus Mecanismos de Execução de Processos de Software**, Orientador Daltro Nunes. Exame de Qualificação do Doutorado. PPGC-UFRGS. 2000.

REZENDE, S. **Sistemas inteligentes**. Barueri: Manole, 2003.

RUIZ, F. H. **Using Ontologies in Software Engineering and Technology**, chapter 2. In: Ontologies for Software Engineering and Software Technology. Springer. 2006.

RUSSELL, S., NORVIG, P. **Artificial Intelligence – A Modern Approach**. Prentice-Hall. 2a. edição. New Jersey, 2003.

THADDEUS, S., RAJA, K. S.V. **Ontology-driven Software Engineering Environment**. Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering, San Francisco, California, 337-342. July (2006).

USCHOLD, M. **Building Ontologies: Towards A Unified Methodology**. Proceedings Expert Systems 96. Cambridge, 1996.

W3C. http://www.w3.org/TR/owl-guide/. 2004

WARREN, P. **Knowledge Management and the Semantic Web: From Scenario to Technology**. IEEE Intelligent Systems, January/February , 53-59. 2006.

WATERMAN, D. A. **A Guide to expert systems**. Addison-Wesley Publishing Company,1986.

WOOLDRIDGE, M. **Agent-based software engineering.** IEE Proc. on Software Engineering, 144 (1) 26-37. 1997

WOOLDRIDGE, M. JENNINGS, N. R. **Agent-Oriented Software Engineering**. In 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World. 1999

YU, E. A. **Organization Modelling Environment**. Fonte: http://www.cs.toronto.edu/km/ome/. (Acesso em 30 de setembro de 2008). 2000.

APÊNDICES

APÊNDICE A – REGRAS DE INFERÊNCIA DO NÚCLEO DO AMBIENTE EM JESS

1. Análise de domínio

   1.1. Tarefa e produto da análise de domínio

```
(defrule DomainAnalysisTaskCreate
  ?DomainAnalysisTask <- (ModelingTask (task DomainAnalysis) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Domain Analysis" (name (str-cat "Domain Analysis Of " ?*project*))))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?DomainAnalysisTask (ref ?ref))
  (assert (ModelingProduct (product DomainModel) (name (str-cat "Domain Model Of " ?*project*))))
  (assert (ModelingTask (task GoalModeling) (name (str-cat "Goal Modeling Of " ?*project*))))
)

(defrule DomainModelCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil))
  ?DomainModel <- (ModelingProduct (product DomainModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Domain Model" (name ?name)))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?DomainModel (ref ?ref))
  (slot-set ?DomainAnalysisTask products ?ref)
)
```

   1.2. Modelagem de Conceitos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule ConceptModelingTaskCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil))
  ?ConceptModelingTask <- (ModelingTask (task ConceptModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Concept Modeling" (name (str-cat "Concept Modeling Of " ?*project*))))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?ConceptModelingTask (ref ?ref))
  (slot-insert$ ?DomainAnalysisTask subtasks 1 ?ref)
  (assert (ModelingProduct (product ConceptModel) (name (str-cat "Concept Model Of " ?*project*))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
```

```
(defrule ConceptModelCreate
  (ModelingProduct (product DomainModel) (ref ?DomainModel&~nil))
  (ModelingTask (task ConceptModeling) (ref ?ConceptModeling&~nil))
  ?ConceptModel <- (ModelingProduct (product ConceptModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Concept Model" (name ?name)))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?ConceptModel (ref ?ref))
  (slot-set ?ConceptModeling products ?ref)
  (slot-insert$ ?DomainModel subproducts 1 ?ref)
  (show ?ref)
)
```

1.3. Modelagem de objetivos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Tarefa Modelagem de Objetivos
(defrule GoalModelingTaskCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil)) ;; Tarefa de Modelagem - Analise de
Dominio
  ?GoalModelingTask <- (ModelingTask (task GoalModeling) (ref nil)) ;; Tarefa de Modelagem - Modelagem de
Objetivos
  =>
  (bind ?ref (make-instance of "Goal Modeling" (name (str-cat "Goal Modeling Of " ?*project*))))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?GoalModelingTask (ref ?ref))
  (slot-insert$ ?DomainAnalysisTask subtasks 1 ?ref)
  (assert (ModelingProduct (product GoalModel) (name (str-cat "Goal Model Of " ?*project*))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Produto Modelagem de Objetivos
(defrule GoalModelCreate
  (ModelingProduct (product DomainModel) (ref ?DomainModel&~nil))
  (ModelingTask (task GoalModeling) (ref ?GoalModeling&~nil))
  ?GoalModel <- (ModelingProduct (product GoalModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Goal Model" (name ?name)))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?GoalModel (ref ?ref))
  (slot-set ?GoalModeling products ?ref)
  (slot-insert$ ?DomainModel subproducts 1 ?ref)
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Conceitos Modelagem de Objetivos

(defrule GeneralGoalCreate
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  ?GeneralGoal <- (Concept (class GeneralGoal) (name ?name) (ref nil))
  =>
  ;; Apenas um Objetivo Geral
  (if (eq (count-query-results FindConcept "General Goal" ?name) 0) then
    (bind ?g (make-instance of "General Goal" (name ?name)))
    (bind ?*am_created* (+ 1 ?*am_created*))
  else
    (bind ?g (GetConcept "General Goal" ?name))
    (bind ?*am_reused* (+ 1 ?*am_reused*))
  )
  (slot-insert$ ?GoalModel concepts 1 ?g)
  (retract ?GeneralGoal)
)


(defrule SpecificGoalCreate
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  ?SpecificGoal  <- (Concept (class SpecificGoal) (name ?name) (related ?gg) (ref nil))
  =>
  ;; Objetivos EspecÃficos nÃ£o se repetem
  (bind ?GeneralGoal (GetConcept "General Goal" ?gg))
  (if (eq (count-query-results FindConcept "Specific Goal" ?name) 0) then
    (bind ?sg (make-instance of "Specific Goal" (name ?name) (leads\ to ?GeneralGoal)))
    (bind ?*am_created* (+ 1 ?*am_created*))
  else
    (bind ?sg (GetConcept "Specific Goal" ?name))
    (bind ?*am_reused* (+ 1 ?*am_reused*))
  )
  (slot-insert$ ?GoalModel concepts 1 ?sg)
  (retract ?SpecificGoal)
)


(defrule ResponsibilityCreate
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  ?Responsibility <- (Concept (class Responsibility) (name ?name) (related ?sg) (ref nil))
  =>
  (bind ?SpecificGoal (GetConcept "Specific Goal"  ?sg))
  ;; Se a Responsabilidade jÃ¡ existe pega sua instÃ¢ncia
```

```
  (if (eq (count-query-results FindConcept "Responsibility" ?name) 0) then
    (bind ?r (make-instance of "Responsibility" (name ?name) (achieves ?SpecificGoal)))
    (slot-insert$ ?GoalModel concepts 1 ?r)
    (bind ?*am_created* (+ 1 ?*am_created*))
   else
    (bind ?r (GetConcept "Responsibility" ?name))
    (slot-insert$ ?r achieves 1 ?SpecificGoal)
    (bind ?*am_reused* (+ 1 ?*am_reused*))
  )
  (retract ?Responsibility)
)


(defrule ExternalEntityCreate
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  ?ExternalEntity <- (Concept (class ExternalEntity) (name ?name) (kind ?kind) (related ?sg) (ref nil))
  =>
  ;; Entidades Externas nÃ£o se repetem
  (if (eq (count-query-results FindConcept "External Entity" ?name) 0) then
    (bind ?ee (make-instance of "External Entity" (name ?name)))
    (bind ?*am_created* (+ 1 ?*am_created*))
   else
    (bind ?ee (GetConcept "External Entity" ?name))
    (printout t " ### =========> " (slot-get ?ee name) crlf)
    (bind ?*am_reused* (+ 1 ?*am_reused*))
  )
  (slot-insert$ ?GoalModel concepts 1 ?ee)
  (retract ?ExternalEntity)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; RelaÃ§Ãµes Modelagem de Objetivos
(defrule GetsInformationCreate
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  ?Relationship <- (Relationship (class GetsInformation) (from ?from) (to ?to) (from-class ?from-class) (to-class
?to-class))
  =>
  ;; O relacionamento Ã© bi-direcional
  (bind ?f (GetConcept ?from-class  ?from))
  (bind ?t (GetConcept ?to-class  ?to))
  (bind ?r (make-instance of "Gets Information" (:FROM ?f) (:TO ?t)))
  (bind ?*am_created* (+ 1 ?*am_created*))
  (slot-insert$ ?GoalModel relationships 1 ?r)
  (retract ?Relationship)
)
```

1.4. Modelagem de papéis

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Tarefa Modelagem de Papï¿½is

(defrule RoleModelingTaskCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil))
  ?RoleModelingTask <- (ModelingTask (task RoleModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Role Modeling" (name (str-cat "Role Modeling Of " ?*project*))))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?RoleModelingTask (ref ?ref))
  (slot-insert$ ?DomainAnalysisTask subtasks 1 ?ref)
  (assert (ModelingProduct (product RoleModel) (name (str-cat "Role Model Of " ?*project*))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Produto Modelagem de PapÃ©is

(defrule RoleModelCreate
  (ModelingProduct (product DomainModel) (ref ?DomainModel&~nil))
  (ModelingTask (task RoleModeling) (ref ?RoleModeling&~nil))
  ?RoleModel <- (ModelingProduct (product RoleModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Role Model" (name ?name)))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (slot-set ?RoleModeling products ?ref)
  (slot-insert$ ?DomainModel subproducts 1 ?ref)
  (modify ?RoleModel (ref ?ref))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Produtos Modelagem de Papï¿½is - TRANFORMAÃ‡ÃƒO

(defrule GoalModelToRoleModel
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  (ModelingTask (task RoleModeling) (ref ?RoleModeling&~nil))
  (ModelingProduct (product RoleModel) (name ?name) (ref ?RoleModel&~nil))
  =>
  ;; Busca as Responsabilidades e Entidades Externas do Modelo de Objetivos
  (foreach ?concept (slot-get ?GoalModel concepts)
    (if (or (eq Responsibility (class ?concept))
            (eq External\ Entity (class ?concept))
```

```
      ) then
     (slot-insert$ ?RoleModel concepts 1 ?concept)
     (bind ?*aa_reused* (+ 1 ?*aa_reused*))
   )))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Conceitos Modelagem de Papï¿½is - FIM TRANFORMAÃ‡ÃƒO


(defrule RoleCreate
  (ModelingProduct (product RoleModel) (ref ?RoleModel&~nil))
  ?Role <- (Concept (class Role) (name ?rolename) (related ?resp) (ref nil))
  =>
  ;; Uma role para cada Responsabilidade
  (bind ?Responsibility (GetConcept "Responsibility" ?resp))
  (bind ?role (make-instance of "Role" (name ?rolename) (in\ charge\ of ?Responsibility)))
  (bind ?*am_created* (+ 1 ?*am_created*))
  (slot-insert$ ?RoleModel concepts 1 ?role)
  (retract ?Role)
)

(defrule KnowledgeCreate
  (ModelingProduct (product RoleModel) (ref ?RoleModel&~nil))
  ?Knowledge <- (Concept (class Knowledge) (name ?knowledge) (related ?rel) (kind ?kind))
  =>
  (if (eq (count-query-results FindConcept "Knowledge" ?knowledge) 0) then
    (bind ?know (make-instance of "Knowledge" (name ?knowledge)))
    (slot-insert$ ?RoleModel concepts 1 ?know)
    (bind ?*am_created* (+ 1 ?*am_created*))
   else
     (bind ?know (GetConcept "Knowledge" ?knowledge))
     (bind ?*am_reused* (+ 1 ?*am_reused*))
  )

  (if (eq ?kind OUT-RESP) then
    (bind ?resp (GetConcept "Responsibility" ?rel))
    (assert (Relationship (class ProducedKnowledge) (from ?resp) (to ?know)))
  )

  (if (eq ?kind IN-RESP) then
    (bind ?resp (GetConcept "Responsibility" ?rel))
    (assert (Relationship (class UsedKnowledge) (from ?know) (to ?resp)))
  )

  (if (eq ?kind IN-EE) then
```

```
    (bind ?ee (GetConcept "External Entity" ?rel))
    (assert (Relationship (class TurnsKnow) (from ?know) (to ?ee)))
  )

  (if (eq ?kind OUT-EE) then
    (bind ?ee (GetConcept "External Entity" ?rel))
    (assert (Relationship (class Knows) (from ?ee) (to ?know)))
  )
  (retract ?Knowledge)
)

(defrule ConditionCreate
  (ModelingProduct (product RoleModel) (ref ?RoleModel&~nil))
  ?Condition <- (Concept (class Condition) (name ?condition) (related ?rel) (kind ?kind))
  =>
 (if (eq (count-query-results FindConcept "Condition" ?condition) 0) then
   (bind ?cond (make-instance of "Condition" (name ?condition)))
   (slot-insert$ ?RoleModel concepts 1 ?cond)
   (bind ?*am_created* (+ 1 ?*am_created*))
  else
    (bind ?cond (GetConcept "Condition" ?condition))
    (bind ?*am_reused* (+ 1 ?*am_reused*))
  )

 (if (eq ?kind PRE-RESP) then
   (bind ?resp (GetConcept "Responsibility" ?rel))
   (assert (Relationship (class PreCondition) (from ?cond) (to ?resp)))
 )

 (if (eq ?kind POST-RESP) then
   (bind ?resp (GetConcept "Responsibility" ?rel))
   (assert (Relationship (class PostCondition) (from ?resp) (to ?cond)))
 )

 (if (eq ?kind POST-EE) then
   (bind ?ee (GetConcept "External Entity" ?rel))
   (assert (Relationship (class PostCondition) (from ?ee) (to ?cond)))
 )

 (retract ?Condition)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; RelaÃ§Ãµes Modelagem de PapÃ©is
(defrule ConditionRelationshipCreate
```

```
 (ModelingProduct (product RoleModel) (ref ?RoleModel&~nil))
 ?Relationship <- (Relationship (class ?class) (from ?from) (to ?to))
 =>
 (if (eq ?class PreCondition) then
   (bind ?r (make-instance of "Pre-condition" (:FROM ?from) (:TO ?to)))
   (bind ?*am_created* (+ 1 ?*am_created*))
   (slot-insert$ ?RoleModel relationships 1 ?r)
   (retract ?Relationship)
 )

 (if (eq ?class PostCondition) then
   (bind ?r (make-instance of "Post-condition" (:FROM ?from) (:TO ?to)))
   (bind ?*am_created* (+ 1 ?*am_created*))
   (slot-insert$ ?RoleModel relationships 1 ?r)
   (retract ?Relationship)
 )

 (if (eq ?class ProducedKnowledge) then
   (bind ?r (make-instance of "Produced Knowledge" (:FROM ?from) (:TO ?to)))
   (bind ?*am_created* (+ 1 ?*am_created*))
   (slot-insert$ ?RoleModel relationships 1 ?r)

   (foreach ?role (slot-get ?from performed\ by)
     (assert (Relationship (class Knows) (from ?role) (to ?to)))
   )
  (retract ?Relationship)
 )

(if (eq ?class UsedKnowledge) then
 (bind ?r (make-instance of "Used Knowledge" (:FROM ?from) (:TO ?to)))
 (bind ?*am_created* (+ 1 ?*am_created*))
 (slot-insert$ ?RoleModel relationships 1 ?r)
 (retract ?Relationship)
)

 (if (eq ?class Knows) then
   (if (eq Role (class ?from)) then
     (bind ?r (make-instance of "Knows" (:FROM ?from) (:TO ?to)))
     (bind ?*am_created* (+ 1 ?*am_created*))
     (slot-insert$ ?RoleModel relationships 1 ?r)
   )

  (if (eq External\ Entity (class ?from)) then
       (bind ?r (make-instance of "Knows" (:FROM ?from) (:TO ?to)))
       (bind ?*am_created* (+ 1 ?*am_created*))
```

```
      (slot-insert$ ?RoleModel relationships 1 ?r)
    )
   (retract ?Relationship)
  )

 (if (eq ?class TurnsKnow) then
  (bind ?r (make-instance of "Turns Know" (:FROM ?from) (:TO ?to)))
  (bind ?*am_created* (+ 1 ?*am_created*))
  (slot-insert$ ?RoleModel relationships 1 ?r)
  (retract ?Relationship)
 ))
```

1.5. Modelagem de interação entre papéis

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Tarefa Modelagem InteraÃ§Ã£o entre PapÃ©is

(defrule RoleIntModelingTaskCreate
  (ModelingTask (task DomainAnalysis) (ref ?DomainAnalysisTask&~nil))
  ?RoleIntModelingTask <- (ModelingTask (task RoleIntModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Role Interaction Modeling" (name (str-cat "Role Interaction Modeling Of "
?*project*))))
  (bind ?*aa_created* (+ 1 ?*aa_created*))
  (modify ?RoleIntModelingTask (ref ?ref))
  (slot-insert$ ?DomainAnalysisTask subtasks 1 ?ref)
  (assert (ModelingProduct (product RoleIntModel)))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; TransformaÃ§Ã£o Modelagem de InteraÃ§Ã£o entre PapÃ©is
(defrule RoleIntModelTransformation
  (ModelingProduct (product DomainModel) (ref ?DomainModel&~nil))
  (ModelingProduct (product GoalModel) (ref ?GoalModel&~nil))
  (ModelingProduct (product RoleModel) (ref ?RoleModel&~nil))
  (ModelingTask (task RoleIntModeling) (ref ?RoleIntModeling&~nil))
  ?RoleIntModel <- (ModelingProduct (product RoleIntModel))
  =>
  ;; Busca as Responsabilidades e Entidades Externas do Modelo de Objetivos
  (foreach ?concept (slot-get ?GoalModel concepts)
    (if (eq Specific\ Goal (class ?concept)) then
      (bind ?name (slot-get ?concept name))
      (bind ?modelname (str-cat "Role Interaction Model Of " ?name " Of " ?*project*))
      (bind ?model (make-instance of "Role Interaction Model" (name ?modelname)))
```

```
 (bind ?*aa_created* (+ 1 ?*aa_created*))
 (assert (ModelingProduct (product (str-cat RoleIntModel ?name)) (ref ?model)))

 (slot-insert$ ?RoleIntModeling products 1 ?model)
 (slot-insert$ ?DomainModel subproducts 1 ?model)

 (foreach ?resp (slot-get ?concept achieved\ by)
    (bind ?roles (slot-get ?resp performed\ by))
    (foreach ?role ?roles
       (if (eq Role (class ?role)) then
          (printout t ?role " - " (slot-get ?role name) crlf)
          (slot-insert$ ?model concepts 1 ?role)
          (bind ?*aa_reused* (+ 1 ?*aa_reused*))
       )
   )
 )

 (bind ?relations (slot-get ?GoalModel relationships))
 (foreach ?r ?relations
    (RoleIntModelAddExtEnt ?r ?name ?model)
 )

 (bind ?knows (slot-get ?RoleModel relationships))
 (bind ?used (slot-get ?RoleModel relationships))

(foreach ?ee (slot-get ?model concepts)
   (if (eq External\ Entity (class ?ee)) then
     (foreach ?know ?knows
       (if (eq Knows (class ?know)) then
         (CreateRoleInteractionEE_Knows ?know ?ee ?model))

       (if (eq Turns\ known (class ?know)) then
         (CreateRoleInteractionEE_TurnsKnow ?know ?ee ?model)))
    )
)

 (bind ?produced (slot-get ?RoleModel relationships))
 (foreach ?resp (slot-get ?concept achieved\ by)
    (bind ?nameresp (slot-get ?resp name))

    (foreach ?prod ?produced
     (if (and (eq Produced\ Knowledge (class ?prod)) (neq ?prod nil)) then
      (CreateRoleInteractionRoleAndRole ?prod ?nameresp ?model)))
 )
```

```
    (show ?model)
   )
  )
  (retract ?RoleIntModel)
)

(deffunction CreateRoleInteractionRoleAndRole (?prod ?nameresp ?model)
  (if (and (neq (slot-get ?prod :FROM) nil) (neq (slot-get ?prod :TO))) then
    (bind ?prodfrom (slot-get ?prod :FROM))
    (bind ?prodto (slot-get ?prod :TO))
        (printout t "FROM ===> " (slot-get ?prodfrom name) crlf)
        (printout t "TO ===> " (slot-get ?prodto name) crlf)
    (bind ?prodfname (slot-get ?prodfrom name))
    (bind ?prodtname (slot-get ?prodto name))

    (if (eq Responsibility (class ?prodfrom)) then
      (if (eq ?nameresp ?prodfname) then
        (foreach ?u ?used
          (if (eq Used\ Knowledge (class ?u)) then
           (bind ?ufrom (slot-get ?u :FROM))
           (bind ?uto (slot-get ?u :TO))
           (bind ?ufname (slot-get ?ufrom name))
           (bind ?utname (slot-get ?uto name))
           (if (eq ?prodtname ?ufname) then
             (foreach ?x (slot-get ?prodfrom performed\ by)
               (if (eq Role (class ?x)) then
                     (bind ?role_from ?x)))

             (foreach ?y (slot-get ?uto performed\ by)
               (if (eq Role (class ?y)) then
                 (bind ?role_to ?y)))

             (if (and (neq ?role_from nil) (neq ?role_to nil)) then
               (bind ?r (make-instance of "Role Interaction" (:FROM ?role_from) (:TO ?role_to) (method ?uto)
(parameter ?ufrom) (order 1)))
               (bind ?*aa_created* (+ 1 ?*aa_created*))
               (slot-insert$ ?model relationships 1 ?r)
             )))))
    )
  )
)

(deffunction CreateRoleInteractionEE_TurnsKnow (?knows ?ee ?model)
    (bind ?eename (slot-get ?ee name))
    (bind ?prodfrom (slot-get ?knows :FROM))
```

```
   (bind ?prodto (slot-get ?knows :TO))
   (bind ?prodfname (slot-get ?prodfrom name))
   (bind ?prodtname (slot-get ?prodto name))

   (if (and (eq External\ Entity (class ?prodto))
            (eq ?eename ?prodtname))  then

     (foreach ?u ?used
       (if (and (eq Produced\ Knowledge (class ?u)) (neq (slot-get ?u :FROM) nil) (neq (slot-get ?u :TO) nil)) then
         (bind ?ufrom (slot-get ?u :FROM))
         (bind ?uto (slot-get ?u :TO))
         (bind ?ufname (slot-get ?ufrom name))
         (bind ?utname (slot-get ?uto name))
           (if (and (eq ?prodfname ?utname) (neq (slot-get ?ufrom performed\ by) nil)) then
             (printout t "\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\ " ?eename crlf)
             (foreach ?x (slot-get ?ufrom performed\ by)
               (if (eq Role (class ?x)) then
                 (bind ?role ?x)))
             (bind ?r (make-instance of "Role Interaction" (:FROM ?role) (:TO ?ee) (method ?ufrom) (parameter
?uto) (order 1)))
             (bind ?*aa_created* (+ 1 ?*aa_created*))
             (slot-insert$ ?model relationships 1 ?r))
       ))
   )
)


(deffunction CreateRoleInteractionEE_Knows (?knows ?ee ?model)
   (bind ?eename (slot-get ?ee name))
   (bind ?prodfrom (slot-get ?knows :FROM))
   (bind ?prodto (slot-get ?knows :TO))
   (bind ?prodfname (slot-get ?prodfrom name))
   (bind ?prodtname (slot-get ?prodto name))

   (if (and (eq External\ Entity (class ?prodfrom)) (eq ?eename ?prodfname))  then
     (foreach ?u ?used
       (if (and (eq Used\ Knowledge (class ?u)) (neq (slot-get ?u :FROM) nil) (neq (slot-get ?u :TO))) then
         (bind ?ufrom (slot-get ?u :FROM))
         (bind ?uto (slot-get ?u :TO))
         (bind ?ufname (slot-get ?ufrom name))
         (bind ?utname (slot-get ?uto name))
           (if (eq ?prodtname ?ufname) then
             (foreach ?x (slot-get ?uto performed\ by)
               (if (eq Role (class ?x)) then
                 (bind ?role ?x)))
```

```
            (bind ?r (make-instance of "Role Interaction" (:FROM ?ee) (:TO ?role) (method ?uto) (parameter
?prodto) (order 1)))
            (bind ?*aa_created* (+ 1 ?*aa_created*))
            (slot-insert$ ?model relationships 1 ?r)))
        ))
)


(deffunction RoleIntModelAddExtEnt (?r ?name ?model)
    (bind ?from (slot-get ?r :FROM))
    (bind ?to (slot-get ?r :TO))
    (bind ?fname (slot-get ?from name))
    (bind ?tname (slot-get ?to name))

   (if (eq Specific\ Goal (class ?from)) then
      (if (eq ?fname ?name) then
        (bind ?ee ?to)
        (slot-insert$ ?model concepts 1 ?ee)
        (bind ?*aa_reused* (+ 1 ?*aa_reused*))
        (return ?ee)))

   (if (eq Specific\ Goal (class ?to)) then
      (if (eq ?tname ?name) then
        (bind ?ee ?from)
        (slot-insert$ ?model concepts 1 ?ee)
        (bind ?*aa_reused* (+ 1 ?*aa_reused*))
        (return ?ee)))
)
```

1.6. Prototipação da interface do usuário

2. Fase de projeto de domínio

2.1. Tarefa e produto do projeto de domínio

```
(defrule DomainDesignTaskCreate
  ?DomainDesignTask <- (ModelingTask (task DomainDesign) (ref nil))
  =>
  (bind ?ref (make-instance of "Domain Design" (name (str-cat "Domain Design of " ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (bind ?ArchitecturalDesignTask (make-instance of "Architectural Design" (name (str-cat "Architectural Design of
" ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
```

```
  (modify ?DomainDesignTask (ref ?ref))
  (slot-insert$ ?ref subtasks 1 ?ArchitecturalDesignTask)

  (assert (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask)))
  (assert (ModelingProduct (product MultiagentFramework)))
)

(defrule MultiagentFrameworkCreate
  (ModelingTask (task DomainDesign) (ref ?DomainDesignTask&~nil))
  (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask&~nil))
  ?MultiagentFramework <- (ModelingProduct (product MultiagentFramework) (ref nil))
  =>
  (bind ?ref (make-instance of "Multi-agent Framework" (name (str-cat "Multi-agent Framework of " ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (bind ?refard (make-instance of "Architectural Model" (name (str-cat "Architectural Model of " ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))

  (modify ?MultiagentFramework (ref ?ref))
  (slot-set ?DomainDesignTask products ?ref)

  (assert (ModelingProduct (product ArchitecturalModel) (name (str-cat "Architectural Model of " ?*project*)) (ref
?refard)))

  (slot-insert$ ?ref subproducts 1 ?refard)

  (slot-set ?ArchitecturalDesignTask products ?refard)
)
```

2.2. Modelagem arquitetural

2.2.1. Modelagem do conhecimento da sociedade multiagente

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Tarefa Modelagem Do Conhecimento da Sociedade Multiagente
;;
(defrule MASKCreate
  (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask&~nil))
  ?MAKSTask <- (ModelingTask (task MASKModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Multi-agent Society Knowledge Modeling" (name (str-cat "Multi-agent Society
Knowledge Modeling of " ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?MAKSTask (name (str-cat "Multi-agent Society Knowledge Modeling Of " ?*project*)) (ref ?ref))
  (slot-insert$ ?ArchitecturalDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product MASKModel)))
```

```
)

(defrule MASKModelTransformation
  (ModelingProduct (product ArchitecturalModel) (ref ?ArchitecturalModel&~nil))
  (ModelingTask (task MASKModeling) (ref ?MASKModeling&~nil))
  ?MASKModel <- (ModelingProduct (product MASKModel) (ref nil))
  =>
  (bind ?modelname (str-cat "Multi-agent Society Knowledge Model Of " ?*project*))
  (bind ?model (make-instance of "Multi-agent Society Knowledge Model" (name ?modelname)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?MASKModel (name ?modelname) (ref ?model))
  (slot-set ?MASKModeling products ?model)
  (slot-insert$ ?ArchitecturalModel subproducts 1 ?model)
  (bind ?GM (GetConcept "Goal Model" (str-cat "Goal Model Of " ?*project*)))
  (foreach ?concept (slot-get ?GM concepts)
    (if (eq Specific\ Goal (class ?concept)) then
      (bind ?sgname (slot-get ?concept name))
      (if (neq ?sgname nil) then
          (bind ?RIM (GetConcept "Role Interaction Model" (str-cat "Role Interaction Model Of " ?sgname " Of "
?*project*)))
            (foreach ?relationship (slot-get ?RIM relationships)
              (if (eq Role\ Interaction (class ?relationship)) then
                (bind ?know (slot-get ?relationship parameter))
                (bind ?knows (slot-get ?model concepts))
                (bind ?exists not)
                (foreach ?k ?knows
                  (if (and (eq Knowledge (class ?k)) (eq (slot-get ?k name) (slot-get ?know name))) then
                    (bind ?exists yes)
                  )
                )
                (if (and (neq ?know nil) (eq ?exists not)) then
                  (bind ?*pa_reused* (+ 1 ?*pa_reused*))
                  (slot-insert$ ?model concepts 1 ?know))
              ))
      )))
  (show ?model)
)
```

2.2.2. Modelagem da sociedade multiagente

```
(defrule MASCreate
  (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask&~nil))
  ?MASTask <- (ModelingTask (task MASModeling) (ref nil))
  =>
```

```
  (bind ?ref (make-instance of "Multi-agent Society Modeling" (name (str-cat "Multi-agent Society Modeling Of "
?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?MASTask (name (str-cat "Multi-agent Society Modeling Of " ?*project*)) (ref ?ref))
  (slot-insert$ ?ArchitecturalDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product MASModel)))
)


(defrule MASModelTransformation
  (ModelingProduct (product ArchitecturalModel) (ref ?ArchitecturalModel&~nil))
  (ModelingTask (task MASModeling) (ref ?MASModeling&~nil))
  ?MASModel <- (ModelingProduct (product MASModel) (ref nil))
  =>
  (bind ?modelname (str-cat "Multi-agent Society Model Of " ?*project*))
  (bind ?model (make-instance of "Multi-agent Society Model" (name ?modelname)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?MASModel (name ?modelname) (ref ?model))
  (slot-set ?MASModeling products ?model)
  (slot-insert$ ?ArchitecturalModel subproducts 1 ?model)
  (bind ?RM (GetConcept "Role Model" (str-cat "Role Model Of " ?*project*)))

  (foreach ?concept (slot-get ?RM concepts)
    (if (neq Role (class ?concept)) then
      (if (neq ?concept nil) then
        (bind ?*pa_reused* (+ 1 ?*pa_reused*))
        (slot-insert$ ?model concepts 1 ?concept))))

  (foreach ?relationship (slot-get ?RM relationships)
    (if (neq Knows (class ?relationship)) then
      (if (neq ?relationship nil) then
        (bind ?*pa_reused* (+ 1 ?*pa_reused*))
        (slot-insert$ ?model relationships 1 ?relationship)))

    (if (eq Knows (class ?relationship)) then
      (if (neq ?relationship nil) then
        (bind ?from (slot-get ?relationship :FROM))
        (if (neq Role (class ?from)) then
          (bind ?*pa_reused* (+ 1 ?*pa_reused*))
          (slot-insert$ ?model relationships 1 ?relationship))))
  )

  ;;(show ?model)

)
```

```
(defrule AgentCreate
  (ModelingProduct (product MASModel) (ref ?MASModel&~nil))
  ?Agent <- (Concept (class Agent) (name ?agname) (related ?resp) (ref nil))
  =>
  (bind ?Responsibility (GetConcept "Responsibility" ?resp))
  (if (eq (count-query-results FindConcept "Agent" ?agname) 0) then
      (bind ?agent (make-instance of "Agent" (name ?agname) (in\ charge\ of ?Responsibility)))
      (bind ?*pm_created* (+ 1 ?*pm_created*))
      (slot-insert$ ?MASModel concepts 1 ?agent)
   else
     (bind ?agent (GetConcept "Agent" ?agname))
     (slot-insert$ ?agent in\ charge\ of 1 ?Responsibility)
     (bind ?*pm_reused* (+ 1 ?*pm_reused*))
   )

  (bind ?produced (slot-get ?MASModel relationships))
  (foreach ?prod ?produced
     (if (eq Produced\ Knowledge (class ?prod)) then
        (bind ?from (slot-get ?prod :FROM))
        (bind ?to (slot-get ?prod :TO))
        (bind ?namefrom (slot-get ?from name))
        (bind ?nameto1 (slot-get ?to name))
        (if (eq Responsibility (class ?from)) then
           (if (eq ?resp ?namefrom) then
              (bind ?r (make-instance of "Knows" (:FROM ?agent) (:TO ?to)))
              (bind ?*pa_created* (+ 1 ?*pa_created*))
              (slot-insert$ ?MASModel relationships 1 ?r)
              (printout t " # PRODUCED KNOWLEDGE #### FROM "  ?namefrom " ===> " ?nameto1 crlf)
           )
        )
     )
  )

  (retract ?Agent)
)

(defrule SkillCreate
  (ModelingProduct (product MASModel) (ref ?MASModel&~nil))
  ?Skill <- (Concept (class Skill) (name ?sname) (related ?resp) (ref nil))
  =>
 (bind ?Responsibility (GetConcept "Responsibility" ?resp))
 (if (eq (count-query-results FindConcept "Skill" ?sname) 0) then
      (bind ?skill (make-instance of "Skill" (name ?sname) (required\ by ?Responsibility)))
      (bind ?*pm_created* (+ 1 ?*pm_created*))
```

```
    (slot-insert$ ?MASModel concepts 1 ?skill)
  else
    (bind ?skill (GetConcept "Skill" ?sname))
    (slot-insert$ ?skill required\ by 1 ?Responsibility)
    (bind ?*pm_reused* (+ 1 ?*pm_reused*))
  )
 (retract ?Skill)
)

(defglobal ?*v_role* = (create$))

(defrule FindRoles
  =>
  (bind ?RM (GetConcept "Role Model" (str-cat "Role Model Of " ?*project*)))

  (foreach ?concept (slot-get ?RM concepts)
     (if (eq Role (class ?concept)) then
        (bind ?role (slot-get ?concept name))
        (bind ?resp (slot-get (slot-get ?concept in\ charge\ of) name))
                    (store ROLES (bind ?*v_role*  (union$ ?*v_role* (create$ (str-cat ?role ";" ?resp)))))
     )
  )
)
```

2.2.3. Modelagem de interações entre agentes

```
(defrule AIMCreate
  (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask&~nil))
  ?MASTask <- (ModelingTask (task AIModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Agent Interaction Modeling" (name (str-cat "Agent Interaction Modeling Of "
?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?MASTask (name (str-cat "Agent Interaction Modeling Of " ?*project*)) (ref ?ref))
  (slot-insert$ ?ArchitecturalDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product AIModel)))
)

(defrule IAModelTransformation
  ;;(ModelingProduct (product ArchitecturalModel) (ref ?ArchitecturalModel&~nil))
  ;;(ModelingTask (task IAModeling) (ref ?IAModeling&~nil))
  ?IAModel <- (ModelingProduct (product AIModel) (ref nil))
  =>
  (bind ?modelname (str-cat "Agent Interaction Model Of " ?*project*))
  (bind ?model (make-instance of "Agent Interaction Model" (name ?modelname)))
```

```
(bind ?*pa_created* (+ 1 ?*pa_created*))
(modify ?IAModel (name ?modelname) (ref ?model))
;;(slot-set ?IAModeling products ?model)
;;(slot-insert$ ?ArchitecturalModel subproducts 1 ?model)

(bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))

(foreach ?concept (slot-get ?MAS concepts)
  (if (eq Agent (class ?concept)) then
    (if (neq ?concept nil) then
      (slot-insert$ ?model concepts 1 ?concept)
      (bind ?*pa_reused* (+ 1 ?*pa_reused*))
      (bind ?lifeline (make-instance of "Life Line" (associated\ with ?concept)))
      (bind ?*pa_created* (+ 1 ?*pa_created*))
      (slot-insert$ ?model concepts 1 ?lifeline)
      (printout t (str-cat "Agent -> " (slot-get ?concept name)) crlf)))

  (if (eq External\ Entity (class ?concept)) then
    (if (neq ?concept nil) then
      (slot-insert$ ?model concepts 1 ?concept)
      (bind ?*pa_reused* (+ 1 ?*pa_reused*))
      (bind ?lifeline (make-instance of "Life Line" (associated\ with ?concept)))
      (bind ?*pa_created* (+ 1 ?*pa_created*))
      (slot-insert$ ?model concepts 1 ?lifeline)))
)

(bind ?knows (slot-get ?MAS relationships))
(bind ?used (slot-get ?MAS relationships))
(bind ?prerel (slot-get ?MAS relationships))

(foreach ?know ?knows
   (if (and (eq Knows (class ?know)) (neq ?know nil) (neq nil (slot-get ?know :FROM))) then
      (bind ?cond "")
      (bind ?from1 (slot-get ?know :FROM))
      (bind ?to1 (slot-get ?know :TO))
      (bind ?namefrom (slot-get ?from1 name))
      (bind ?nameto1 (slot-get ?to1 name))
      (if (eq Knowledge (class ?to1))  then
         (foreach ?usedby ?used
            (if (eq Used\ Knowledge (class ?usedby)) then
               (bind ?from2 (slot-get ?usedby :FROM))
               (bind ?to2 (slot-get ?usedby :TO))
               (bind ?namefrom2 (slot-get ?from2 name))
               (bind ?nameto2 (slot-get ?to2 name))
               (if (eq ?nameto1 ?namefrom2) then
```

```
                (bind ?ags (slot-get ?to2 performed\ by))
                (foreach ?ag ?ags
                   (if (and (eq Agent (class ?ag)) (neq ?ag nil)) then
                      (bind ?nameto (slot-get ?ag name))
                      (bind ?lifelines (slot-get ?model concepts))
                      (bind ?linefrom nil)
                      (bind ?lineto nil)
                      (foreach ?life ?lifelines
                         (if (eq Life\ Line (class ?life)) then
                            (bind ?con (slot-get ?life associated\ with))
                            (bind ?namecon (slot-get ?con name))
                                        (if (eq ?namefrom ?namecon) then
                              (bind ?linefrom ?life))

                            (if (eq ?nameto ?namecon) then
                              (bind ?lineto ?life))))

                       (if (and (neq ?lineto nil) (neq ?linefrom nil)) then
                         (if (eq Agent (class ?from1)) then
                            (bind ?rel (make-instance of "Agent Exchanged Message" (content ?to1) (:FROM
?linefrom) (:TO ?lineto)))
                            (bind ?*pa_created* (+ 1 ?*pa_created*))
                            (slot-insert$ ?model relationships 1 ?rel)
                         )

                         (if (eq External\ Entity (class ?from1)) then
                            (foreach ?prer ?prerel
                               (if (and (eq Pre-condition (class ?prer)) (neq ?prer nil) (neq nil (slot-get ?prer :FROM))
(neq nil (slot-get ?prer :TO))) then
                                  (bind ?tox (slot-get ?prer :TO))
                                  (bind ?nametopre (slot-get ?tox name))
                                  (if (eq ?nametopre ?nameto2) then
                                    (bind ?cond (slot-get ?prer :FROM))
                                  )
                               )
                            )
                            (bind ?rel (make-instance of "External Event Notification" (content ?cond) (:FROM
?linefrom) (:TO ?lineto)))
                            (bind ?*pa_created* (+ 1 ?*pa_created*))
                            (slot-insert$ ?model relationships 1 ?rel)
                         )
                       )
                   )
                )
             )
```

```
            )
          )
        )
      )
    )

(foreach ?know ?knows
   (if (and (eq Turns\ known (class ?know)) (neq ?know nil) (neq nil (slot-get ?know :FROM))) then
      (bind ?cond "")
      (bind ?from1 (slot-get ?know :FROM))
      (bind ?to1 (slot-get ?know :TO))
      (bind ?namefrom (slot-get ?from1 name))
      (bind ?nameto1 (slot-get ?to1 name))
      (if (eq Knowledge (class ?from1))  then
          (foreach ?usedby ?used
            (if (eq Produced\ Knowledge (class ?usedby)) then
               (bind ?from2 (slot-get ?usedby :FROM))
               (bind ?to2 (slot-get ?usedby :TO))
               (bind ?namefrom2 (slot-get ?from2 name))
               (bind ?nameto2 (slot-get ?to2 name))
               (if (eq ?namefrom ?nameto2) then
                  (printout t ?namefrom crlf)
                  (bind ?ags (slot-get ?from2 performed\ by))
                  (foreach ?ag ?ags
                     (if (and (eq Agent (class ?ag)) (neq ?ag nil)) then
                        (bind ?nameto (slot-get ?ag name))
                        (bind ?lifelines (slot-get ?model concepts))
                        (bind ?linefrom nil)
                        (bind ?lineto nil)
                        (foreach ?life ?lifelines
                           (if (eq Life\ Line (class ?life)) then
                              (bind ?con (slot-get ?life associated\ with))
                              (bind ?namecon (slot-get ?con name))
                                          (if (eq ?nameto ?namecon) then
                                 (printout t "From -> " ?nameto crlf)
                                 (bind ?linefrom ?life))

                              (if (eq ?nameto1 ?namecon) then
                                 (printout t "To -> " ?nameto1 crlf)
                                 (bind ?lineto ?life))))

                        (if (and (neq ?lineto nil) (neq ?linefrom nil)) then
                           (foreach ?prer ?prerel
                              (if (and (eq Post-condition (class ?prer)) (neq ?prer nil) (neq nil (slot-get ?prer :FROM))
(neq nil (slot-get ?prer :TO))) then
```

```
                                (bind ?tox (slot-get ?prer :FROM))
                                (bind ?nametopre (slot-get ?tox name))
                                (if (eq ?nametopre ?namefrom2) then
                                  (bind ?cond (slot-get ?prer :TO))
                                )
                              )
                            )
                            (bind ?rel (make-instance of "External Event Notification"  (content ?cond) (:FROM
?linefrom) (:TO ?lineto)))
                            (bind ?*pa_created* (+ 1 ?*pa_created*))
                            (slot-insert$ ?model relationships 1 ?rel)
                          )

                      )
                    )
                  )
                )
              )
            )
          )
        )
      )
    )

 (show ?model)
)
```

2.2.4. Modelagem dos mecanismos de cooperação e coordenação

```
(defrule CCMCreate
  (ModelingTask (task ArchitecturalDesignTask) (ref ?ArchitecturalDesignTask&~nil))
  ?CCMTask <- (ModelingTask (task CCMModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Coordination and Cooperation Mechanisms Modeling" (name (str-cat "Coordination
and Cooperation Mechanisms Modeling Of " ?*project*))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?CCMTask (name (str-cat "Coordination and Cooperation Mechanisms Modeling Of " ?*project*)) (ref
?ref))
  (slot-insert$ ?ArchitecturalDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product CCMModel)))
)

(defrule CCMModelTransformation
  (ModelingProduct (product ArchitecturalModel) (ref ?ArchitecturalModel&~nil))
  (ModelingTask (task CCMModeling) (ref ?CCMModeling&~nil))
```

```
  ?CCMModel <- (ModelingProduct (product CCMModel) (ref nil))
  =>
  (bind ?modelname (str-cat "Coordination and Cooperation Mechanisms Model Of " ?*project*))
  (bind ?model (make-instance of "Coordination and Cooperation Mechanisms Model" (name ?modelname)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?CCMModel (name ?modelname) (ref ?model))
  (slot-set ?CCMModeling products ?model)
  (slot-insert$ ?ArchitecturalModel subproducts 1 ?model)

  (bind ?AIM (GetConcept "Agent Interaction Model" (str-cat "Agent Interaction Model Of " ?*project*)))

  (foreach ?concept (slot-get ?AIM concepts)
    (if (eq External\ Entity (class ?concept)) then
      (if (neq ?concept nil) then
        (bind ?*pa_reused* (+ 1 ?*pa_reused*))
        (slot-insert$ ?model concepts 1 ?concept)))
  )

 (show ?model)
)

(defrule AgentLayerCreate
  (ModelingProduct (product CCMModel) (ref ?CCMModel&~nil))
  ?AgentLayer <- (Concept (class AgentLayer) (kind ?level) (name ?layer) (related ?agent) (ref nil))
  =>
  (bind ?ag (GetConcept "Agent" ?agent))
  (if (eq (count-query-results FindConcept "Layer" ?layer) 0) then
      (printout t ?level " " ?layer " " ?agent crlf)
      (bind ?level (create$ ?level))
      ;;(bind ?l (make-instance of "Layer" (name ?layer) (includes ?ag) (abstraction\ level ?level)))
      (bind ?l (make-instance of "Layer" (name ?layer) (includes ?ag)))
      (bind ?*pm_created* (+ 1 ?*pm_created*))
      (slot-insert$ ?CCMModel concepts 1 ?l)
      (store LAYERS_EE (bind ?*v_layers_ee*  (union$ ?*v_layers_ee* (create$ ?layer))))
   else
     (bind ?l (GetConcept "Layer" ?layer))
     (slot-insert$ ?l includes 1 ?ag)
     (bind ?*pm_reused* (+ 1 ?*pm_reused*))
   )

  (show ?CCMModel)
  (retract ?AgentLayer)
)

(defrule SolutionRelCreate
```

```
  (ModelingProduct (product CCMModel) (ref ?CCMModel&~nil))
  ?SolutionRel <- (Relationship (class SolutionRel) (from ?from) (to ?to) (to-class ?know))
  =>
  (printout t ?from " - " ?know " - " ?to crlf)
  (if (neq (count-query-results FindConcept "Layer" ?from) 0) then
    (bind ?from (GetConcept "Layer" ?from))
  else
    (bind ?from (GetConcept "External Entity" ?from))
  )

  (if (neq (count-query-results FindConcept "Layer" ?to) 0) then
    (bind ?to (GetConcept "Layer" ?to))
  else
    (bind ?to (GetConcept "External Entity" ?to))
  )

  (bind ?know (GetConcept "Knowledge" ?know))
  (bind ?r (make-instance of "Solution Relationship" (concepts ?know) (:FROM ?from) (:TO ?to)))
  (bind ?*pm_created* (+ 1 ?*pm_created*))
  (slot-insert$ ?CCMModel relationships 1 ?r)
  (show ?CCMModel)
  (retract ?SolutionRel)
)



(defrule FindAgents
  (ModelingProduct (product CCMModel) (ref ?CCMModel&~nil))
  =>
  (bind ?AIM (GetConcept "Agent Interaction Model" (str-cat "Agent Interaction Model Of " ?*project*)))
  (foreach ?concept (slot-get ?AIM concepts)
     (if (eq Agent (class ?concept)) then
        (bind ?agent (slot-get ?concept name))
                    (store AGENTS (bind ?*v_agent*  (union$ ?*v_agent* (create$ ?agent))))))

  (bind ?MASK (GetConcept "Multi-agent Society Knowledge Model" (str-cat "Multi-agent Society Knowledge
Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MASK concepts)
       (if (eq Knowledge (class ?concept)) then
          (bind ?know (slot-get ?concept name))
          (store KNOWLEDGE (bind ?*v_know*  (union$ ?*v_know* (create$ ?know))))))

 (bind ?CCM (GetConcept "Coordination and Cooperation Mechanisms Model" (str-cat "Coordination and
Cooperation Mechanisms Model Of " ?*project*)))
 (foreach ?concept (slot-get ?CCM concepts)
```

```
    (if (eq Layer (class ?concept)) then
       (bind ?layer (slot-get ?concept name))
       (store LAYERS_EE (bind ?*v_layers_ee*  (union$ ?*v_layers_ee* (create$ ?layer))))))

  (foreach ?concept (slot-get ?CCM concepts)
    (if (eq External\ Entity (class ?concept)) then
       (bind ?ee (slot-get ?concept name))
                    (store LAYERS_EE (bind ?*v_layers_ee*  (union$ ?*v_layers_ee* (create$ ?ee))))))
)
```

2.3. Tarefa e produto do projeto do agente

```
(defrule AgentDesignTaskCreate
  (ModelingTask (task DomainDesign) (ref ?DomainDesignTask&~nil))
  ?AgentDesignTask <- (ModelingTask (task AgentDesign) (name nil) (ref nil))
  =>
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
    (if (eq Agent (class ?concept)) then
      (bind ?name (slot-get ?concept name))
      (bind ?modelname (str-cat "Agent Design Of " ?name " Of " ?*project*))
      (bind ?model (make-instance of "Agent Design" (name ?modelname)))
      (bind ?*pa_created* (+ 1 ?*pa_created*))
      (slot-insert$ ?DomainDesignTask subtasks 1 ?model)

      (bind ?modelname (str-cat ?name " Of " ?*project*))
      (assert (ModelingTask (task AgentDesignTask) (name ?modelname) (ref ?model)))
      (assert (ModelingProduct (product AgentModel) (name ?modelname)))
      (assert (ModelingTask (task AKMModeling) (name ?modelname)))
      (assert (ModelingTask (task AAMModeling) (name ?modelname)))

      (printout t ?modelname crlf)
    )
  )
  (retract ?AgentDesignTask)
)

(defrule AgentModelCreate
  (ModelingProduct (product MultiagentFramework) (ref ?MF&~nil))
  (ModelingTask (task AgentDesignTask) (ref ?AgentDesign&~nil) (name ?name))
  ?AgentModel <- (ModelingProduct (product AgentModel) (ref nil) (name ?name))
  =>
  (bind ?ref (make-instance of "Agent Model" (name (str-cat "Agent Model Of " ?name))))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
```

```
  (modify ?AgentModel (ref ?ref))
  (slot-set ?AgentDesign products ?ref)
  (slot-insert$ ?MF subproducts 1 ?ref)
)

(defrule FindResps
  =>
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
       (if (eq Responsibility (class ?concept)) then
           (bind ?resp (slot-get ?concept name))
           (store RESPS (bind ?*v_resp*  (union$ ?*v_resp* (create$ ?resp)))))
  )
)
```

2.3.1. Modelagem do conhecimento do agente

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule AKMCreate
  (ModelingTask (task AgentDesignTask) (ref ?AgentDesignTask&~nil) (name ?modelname))
  ?AKMTask <- (ModelingTask (task AKMModeling) (ref nil) (name ?modelname))
  =>
  (bind ?name (str-cat "Agent Knowledge Modeling Of " ?modelname))
  (bind ?ref (make-instance of "Agent Knowledge Modeling"  (name ?name)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?AKMTask (name ?modelname) (ref ?ref))
  (slot-insert$ ?AgentDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product AKMModel) (name ?modelname)))
)

(defrule AKModelTransformation
  (ModelingProduct (product AgentModel) (ref ?ArchitecturalModel&~nil) (name ?modelname))
  (ModelingTask (task AKMModeling) (ref ?AKMModeling&~nil) (name ?modelname))
  ?AKMModel <- (ModelingProduct (product AKMModel) (ref nil) (name ?modelname))
  =>
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
    (if (eq Agent (class ?concept)) then
      (bind ?name (slot-get ?concept name))
      (bind ?namex (str-cat ?name " Of " ?*project*))
      (if (eq ?modelname ?namex) then
         (bind ?modelname (str-cat "Agent Knowledge Model Of " ?name " Of " ?*project*))
         (printout t ?modelname crlf)
```

```
        (bind ?model (make-instance of "Agent Knowledge Model" (name ?modelname)))
        (bind ?*pa_created* (+ 1 ?*pa_created*))
        (assert (ModelingProduct (product (str-cat AKModel ?name)) (ref ?model)))

        (bind ?knows (slot-get ?MAS relationships))
        (foreach ?k ?knows
          (if (and (eq Knows (class ?k)) (neq ?k nil) (neq nil (slot-get ?k :FROM))) then
             (bind ?from (slot-get ?k :FROM))
             (bind ?to (slot-get ?k :TO))
             (bind ?fromname (slot-get ?from name))
             (bind ?toname (slot-get ?to name))
             (if (eq ?fromname ?name) then
                (slot-insert$ ?model concepts 1 ?to)
                (bind ?*pa_reused* (+ 1 ?*pa_reused*))
             )
          )
        )
        (slot-insert$ ?AKMModeling products 1 ?model)
        (slot-insert$ ?ArchitecturalModel subproducts 1 ?model)
        ;(show ?model)
     )
   )
  )
  (retract ?AKMModel)
)

(defrule FindAgents
  =>
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
       (if (eq Agent (class ?concept)) then
          (bind ?agent (slot-get ?concept name))
          (store AGENTS (bind ?*v_agent*  (union$ ?*v_agent* (create$ ?agent))))))
)
```

2.3.2. Modelagem das ações do agente

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule AAMCreate
  (ModelingTask (task AgentDesignTask) (ref ?AgentDesignTask&~nil) (name ?modelname))
  ?AAMTask <- (ModelingTask (task AAMModeling) (ref nil) (name ?modelname))
  =>
  (bind ?name (str-cat "Agent Action Modeling Of " ?modelname))
```

```
  (bind ?ref (make-instance of "Agent Action Modeling" (name ?name)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?AAMTask (name ?modelname) (ref ?ref))
  (slot-insert$ ?AgentDesignTask subtasks 1 ?ref)
  (assert (ModelingProduct (product AAMModel) (name ?modelname)))
)

(defrule AAModelTransformation
  (ModelingProduct (product AgentModel) (ref ?AgentModel&~nil) (name ?modelname))
  (ModelingTask (task AAMModeling) (ref ?AAMModeling&~nil) (name ?modelname))
  ?AAMModel <- (ModelingProduct (product AAMModel) (ref nil) (name ?modelname))
  =>
  (bind ?name (str-cat "Agent Action Model Of " ?modelname))
  (bind ?model (make-instance of "Agent Action Model" (name ?name)))
  (bind ?*pa_created* (+ 1 ?*pa_created*))
  (modify ?AAMModel (ref ?model))
  (slot-insert$ ?AAMModeling products 1 ?model)
  (slot-insert$ ?AgentModel subproducts 1 ?model)
  ;;(bind ?ag (first$ (explode$ ?modelname)))
  (bind ?ag (explode$ ?modelname))
  (bind ?x (create$))
  (bind ?stop n)
  (foreach ?a ?ag
    (if (eq ?stop n) then
        (if (eq ?a Of) then
            (bind ?stop y)
        else
           (bind ?x (create$ ?x ?a)))))
  (bind ?ag (implode$ ?x))
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)

      (if (and (eq Agent (class ?concept)) (eq ?ag (slot-get ?concept name))) then
          (bind ?name (slot-get ?concept name))
          (bind ?knows (slot-get ?MAS relationships))
          (foreach ?k ?knows
            (if (and (eq Knows (class ?k)) (neq ?k nil) (neq nil (slot-get ?k :FROM))) then
              (bind ?from (slot-get ?k :FROM))
              (bind ?to (slot-get ?k :TO))
              (bind ?fromname (slot-get ?from name))
              (bind ?toname (slot-get ?to name))
              (if (eq ?fromname ?name) then
                 (slot-insert$ ?model concepts 1 ?to)
                 (bind ?*pa_reused* (+ 1 ?*pa_reused*))
              )))
```

```
(slot-insert$ ?model concepts 1 ?concept)
(bind ?resp (slot-get ?concept in\ charge\ of))
(slot-insert$ ?model concepts 1 ?resp)
(foreach ?r ?resp
 (bind ?req (slot-get ?r requires))
 (slot-insert$ ?model concepts 1 ?req)

 (foreach ?relationship (slot-get ?MAS relationships)
  (if (eq Pre-condition (class ?relationship)) then
   (if (neq ?relationship nil) then
    (bind ?from (slot-get ?relationship :FROM))
    (bind ?to (slot-get ?relationship :TO))
    (if (and (eq Condition (class ?from)) (eq (slot-get ?r name) (slot-get ?to name))) then
     (bind ?*pa_reused* (+ 1 ?*pa_reused*))
     (bind ?exists not)
     (bind ?con (slot-get ?model concepts))
     (foreach ?x ?con
      (if (and (eq Condition (class ?x)) (eq (slot-get ?x name) (slot-get ?from name))) then
         (bind ?exists yes)))
     (if (eq ?exists not) then
        (slot-insert$ ?model concepts 1 ?from)
        (slot-insert$ ?model relationships 1 ?relationship)
     else
        (slot-insert$ ?model relationships 1 ?relationship)
     )))))


 (foreach ?relationship (slot-get ?MAS relationships)
  (if (eq Post-condition (class ?relationship)) then
   (if (neq ?relationship nil) then
    (bind ?from (slot-get ?relationship :FROM))
    (bind ?to (slot-get ?relationship :TO))
    (if (and (eq Condition (class ?to)) (eq (slot-get ?r name) (slot-get ?from name))) then
     (bind ?*pa_reused* (+ 1 ?*pa_reused*))
     (bind ?con (slot-get ?model concepts))
     (bind ?exists not)
     (foreach ?x ?con
      (if (and (eq Condition (class ?x)) (eq (slot-get ?x name) (slot-get ?to name))) then
         (bind ?exists yes)))
     (if (eq ?exists not) then
        (slot-insert$ ?model concepts 1 ?to)
        (slot-insert$ ?model relationships 1 ?relationship)
     else
        (slot-insert$ ?model relationships 1 ?relationship)
```

```
                )))))
        )

        (bind ?AIM (GetConcept "Agent Interaction Model" (str-cat "Agent Interaction Model Of " ?*project*)))
        (bind ?cs (slot-get ?AIM concepts))
        (bind ?rs (slot-get ?AIM relationships))
        (foreach ?concept ?cs
           (if (and (eq Life\ Line (class ?concept)) (eq ?ag (slot-get (slot-get ?concept associated\ with) name))) then
              (bind ?ll (instance-name ?concept))
              (foreach ?rel ?rs
                 (if (and (eq Agent\ Exchanged\ Message (class ?rel)) (eq ?ll (instance-name (slot-get ?rel :TO))))
then
                    (bind ?from (slot-get (slot-get ?rel :FROM) associated\ with))
                    (bind ?to (slot-get (slot-get ?rel :TO) associated\ with))
                    (bind ?mes (slot-get (slot-get ?rel content) name))
                    (printout t ?from " -> " ?ll " -> " ?ag " -> " ?mes crlf)

                     (bind ?exists not)
                     (bind ?con (slot-get ?model concepts))

                     (foreach ?x ?con
                       (if (and (eq Agent (class ?x)) (eq (slot-get ?x name) (slot-get ?from name))) then
                          (bind ?exists yes)))

                     (if (eq ?exists not) then
                        (slot-insert$ ?model concepts 1 ?from))

                     (foreach ?x ?con
                       (if (and (eq Agent (class ?x)) (eq (slot-get ?x name) (slot-get ?to name))) then
                          (bind ?exists yes)))

                     (if (eq ?exists not) then
                        (slot-insert$ ?model concepts 1 ?to))

                    (bind ?m (make-instance of "Message" (name ?mes) (content (slot-get ?rel content))))
                    (slot-insert$ ?model concepts 1 ?m)

                    (bind ?per (make-instance of "Perceives" (:FROM ?to) (:TO ?m)))
                    (bind ?gen (make-instance of "Generates" (:FROM ?from) (:TO ?m)))

                    (slot-insert$ ?model relationships 1 ?per)
                    (slot-insert$ ?model relationships 1 ?gen)
                 )

           (if (and (eq External\ Event\ Notification (class ?rel)) (eq ?ll (instance-name (slot-get ?rel :TO)))) then
```

```
                (bind ?from (slot-get (slot-get ?rel :FROM) associated\ with))
                (bind ?to (slot-get (slot-get ?rel :TO) associated\ with))
                (bind ?mes (slot-get (slot-get ?rel content) name))
                (printout t ?from " -> " ?ll " -> " ?ag " -> " ?mes crlf)

                 (bind ?exists not)
                 (bind ?con (slot-get ?model concepts))

                 (foreach ?x ?con
                   (if (and (eq External\ Entity (class ?x)) (eq (slot-get ?x name) (slot-get ?from name))) then
                      (bind ?exists yes)))

                 (if (eq ?exists not) then
                    (slot-insert$ ?model concepts 1 ?from))

                 (foreach ?x ?con
                   (if (and (eq Agent (class ?x)) (eq (slot-get ?x name) (slot-get ?to name))) then
                      (bind ?exists yes)))

                 (if (eq ?exists not) then
                    (slot-insert$ ?model concepts 1 ?to))

                (bind ?m (make-instance of "Event" (name ?mes)))
                (slot-insert$ ?model concepts 1 ?m)

                (bind ?per (make-instance of "Perceives" (:FROM ?to) (:TO ?m)))
                (bind ?gen (make-instance of "Generates" (:FROM ?from) (:TO ?m)))

                (slot-insert$ ?model relationships 1 ?per)
                (slot-insert$ ?model relationships 1 ?gen)
              )))
      )

   ))
)


(defrule SetAgentKind
  (ModelingProduct (product AgentModel) (ref ?AgentModel&~nil) (name ?modelname))
  (ModelingProduct (product AAMModel) (ref ?AAMModel&~nil) (name ?modelname))
  (ModelingTask (task AAMModeling) (ref ?AAMModeling&~nil) (name ?modelname))
  ?AgentKind <- (Concept (class AgentKind) (name ?modelname) (kind ?kind))
  =>
  (bind ?exists not)
  (bind ?con (slot-get ?AAMModel concepts))
```

```
  (foreach ?x ?con
    (if (eq Planning\ Algorithm (class ?x)) then
        (bind ?exists yes)
        (bind ?xx ?x)
        ))

  (slot-set ?AgentModel type ?kind)
  (if (eq ?kind "deliberative") then
      (if (eq ?exists not) then
         (bind ?m (make-instance of "Planning Algorithm" (name "Rule Engine")))
         (slot-insert$ ?AAMModel concepts 1 ?m))
  else
      (if (eq ?exists yes) then
         (unmake-instance ?xx))
  )
  (retract ?AgentKind)
)

(defrule CreateAction
  (ModelingProduct (product AAMModel) (ref ?AgentModel&~nil) (name ?modelname))
  ?Action <- (Concept (class Action) (name ?name) (kind ?modelname) (related ?resp))
  =>
  (printout t ?name crlf)
  (printout t ?modelname crlf)
  (printout t ?resp crlf)

  (bind ?ac (make-instance of "Action" (name ?name)))
  (slot-insert$ ?AgentModel concepts 1 ?ac)

  (bind ?r (GetConcept "Responsibility" ?resp))
  (slot-insert$ ?r exercised\ through 1 ?ac)
  (retract ?Action)
)

(defrule FindAgs
  =>
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
       (if (eq Agent (class ?concept)) then
          (bind ?agent (slot-get ?concept name))
          (store AGENTS (bind ?*v_agent*  (union$ ?*v_agent* (create$ ?agent))))))
)

(defrule FindRespsAgs
  ?x <- (FindResp (related ?agent))
```

```
  =>
  (bind ?*v_resp* (create$))
  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
      (if (and (eq Agent (class ?concept)) (eq (slot-get ?concept name) ?agent)) then
         (printout t (slot-get ?concept name) crlf)
         (printout t ?agent crlf)
         (bind ?resps (slot-get ?concept in\ charge\ of))
         (foreach ?resp ?resps
            (printout t (slot-get ?resp name) crlf)
            (store RESPSAG (bind ?*v_resp*  (union$ ?*v_resp* (create$ (slot-get ?resp name)))))
         )
      )
  )
  (retract ?x)
)
```

3. Fase de implementação de domínio

3.1. Tarefa e produto de implementação de domínio

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;; Implementação do Domínio

(defrule DomainImplementationTaskCreate
  ?DomainImplementationTask <- (ModelingTask (task DomainImplementation) (ref nil))
  =>
  (bind ?ref (make-instance of "Domain Implementation" (name (str-cat "Domain Imeplementation Of "
?*project*))))
  (bind ?*ia_created* (+ 1 ?*ia_created*))
  (modify ?DomainImplementationTask (ref ?ref))
  (assert (ModelingProduct (product ImplementationModel) (name (str-cat "Implementation Model of the Multi-
agent Society Of " ?*project*))))
  (assert (ModelingTask (task BehaviourModeling)))
  (assert (ModelingTask (task CommModeling)))
)

(defrule ImplementationModelCreate
  (ModelingTask (task DomainImplementation) (ref ?DomainImplementationTask&~nil))
  ?ImplementationModel <- (ModelingProduct (product ImplementationModel) (name ?name) (ref nil))
  =>
  (bind ?ref (make-instance of "Implementation Model of the Multi-agent Society" (name ?name)))
  (bind ?*ia_created* (+ 1 ?*ia_created*))
  (modify ?ImplementationModel (ref ?ref))
  (slot-set ?DomainImplementationTask products ?ref)
```

```
)
```

3.2. Modelagem de comportamentos

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule BehaviourModelingTaskCreate
  (ModelingTask (task DomainImplementation) (ref ?DomainImplementationTask&~nil))
  ?BehaviourModelingTask <- (ModelingTask (task BehaviourModeling) (ref nil))
  ?ComActModelingTask <- (ModelingTask (task CommModeling) (ref nil))
  =>
  (bind ?ref (make-instance of "Behaviours Modeling" (name (str-cat "Behaviours Modeling Of " ?*project*))))
  (bind ?*ia_created* (+ 1 ?*ia_created*))
  (modify ?BehaviourModelingTask (ref ?ref))
  (slot-insert$ ?DomainImplementationTask subtasks 1 ?ref)
  (assert (ModelingProduct (product BehaviourModel)))

  (bind ?ref (make-instance of "Communication Acts Modeling" (name (str-cat "Communication Acts Modeling Of "
?*project*))))
  (bind ?*ia_created* (+ 1 ?*ia_created*))
  (modify ?ComActModelingTask (ref ?ref))
  (slot-insert$ ?DomainImplementationTask subtasks 1 ?ref)
  (assert (ModelingProduct (product CommModel)))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule BehaviourModelTransform
  (ModelingProduct (product ImplementationModel) (ref ?ImplementationModel&~nil))
  (ModelingTask (task BehaviourModeling) (ref ?BehaviourModeling&~nil))
  (ModelingTask (task CommModeling) (ref ?CommModeling&~nil))
  ?BehaviourModel <- (ModelingProduct (product BehaviourModel) (ref nil))
  ?CommModel <- (ModelingProduct (product CommModel) (ref nil))
  =>
  (bind ?name (str-cat "Model Of Behaviours Of " ?*project*))
  (bind ?model (make-instance of "Model of Behaviours" (name ?name)))
  (bind ?*im_created* (+ 1 ?*im_created*))
  (modify ?BehaviourModel (ref ?model))
  (slot-set ?CommModeling products ?model)
  (slot-insert$ ?ImplementationModel subproducts 1 ?model)

  (bind ?name (str-cat "Model Of Communicative Acts Of " ?*project*))
  (bind ?model2 (make-instance of "Model of Communicative Acts" (name ?name)))
```

```
  (bind ?*im_created* (+ 1 ?*im_created*))
  (modify ?CommModel (ref ?model2))
  (slot-set ?CommModeling products ?model2)
  (slot-insert$ ?ImplementationModel subproducts 1 ?model2)

  (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
  (foreach ?concept (slot-get ?MAS concepts)
   (if (and (eq Agent (class ?concept))) then
       (bind ?ja (make-instance of "JADE Agent" (name (slot-get ?concept name))))
       (slot-insert$ ?model concepts 1 ?ja)
       (slot-insert$ ?model2 concepts 1 ?ja)

       (foreach ?resp (slot-get ?concept in\ charge\ of)
          (bind ?jb (make-instance of "JADE Behaviour" (name (slot-get ?resp name))))
          (slot-insert$ ?ja plays 1 ?jb)
          (slot-insert$ ?model concepts 1 ?jb)
          (slot-insert$ ?model2 concepts 1 ?jb)

          (foreach ?ac (slot-get ?resp exercised\ through)
             (bind ?jm (make-instance of "Method JADE Behaviour" (name (slot-get ?ac name))))
             (slot-insert$ ?jb has 1 ?jm)
             (slot-insert$ ?model concepts 1 ?jm)
             (slot-insert$ ?model2 concepts 1 ?jm)
          )))
  )

   (bind ?AIM (GetConcept "Agent Interaction Model" (str-cat "Agent Interaction Model Of " ?*project*)))
   (bind ?rs (slot-get ?AIM relationships))
   (foreach ?rel ?rs
      (if (eq Agent\ Exchanged\ Message (class ?rel)) then
         (bind ?name (slot-get (slot-get ?rel content) name))
         (bind ?mes (make-instance of "FIPA-ACL Message" (name ?name) (object\ content (slot-get ?rel
content))))
         (slot-insert$ ?model2 concepts 1 ?mes)
      )
   )
)

(defrule SetBehType
  ?BehType <- (Concept (class BehaviourType) (related ?behav) (kind ?kind))
  =>
  (bind ?name (str-cat "Model Of Behaviours Of " ?*project*))
  (bind ?BM (GetConcept "Model of Behaviours" ?name))
  (foreach ?concept (slot-get ?BM concepts)
      (printout t ?behav " -> " ?kind " " (slot-get ?concept name) crlf)
```

```
      (if (and (eq JADE\ Behaviour (class ?concept)) (eq ?behav (slot-get ?concept name))) then
         (slot-set ?concept type ?kind)))
  (retract ?BehType)
)

;;(Concept (class ComesFrom) (ref " + "" + mes + "" + ") (related " + "" + from + "" + "))
(defrule SetComesFrom
  ?CommAct <- (Concept (class ComesFrom) (ref ?mes) (related ?met))
  (ModelingProduct (product CommModel) (ref ?CommModel&~nil))
  =>
   (printout t ?met " -> " ?mes crlf)
  (bind ?mes (GetConcept "FIPA-ACL Message" ?mes))
  (bind ?met (GetConcept "Method JADE Behaviour" ?met))
  (bind ?cm (make-instance of "Outgoing Communicative Act" (:FROM ?mes) (:TO ?met)))
  (slot-insert$ ?CommModel relationships 1 ?cm)
  (retract ?CommAct)
)

(defrule SetGoesTo
  ?CommAct <- (Concept (class GoesTo) (ref ?mes) (related ?met))
  (ModelingProduct (product CommModel) (ref ?CommModel&~nil))
  =>
  (printout t ?met " -> " ?mes crlf)
  (bind ?mes (GetConcept "FIPA-ACL Message" ?mes))
  (bind ?met (GetConcept "Method JADE Behaviour" ?met))
  (bind ?cm (make-instance of "Incoming Communicative Act" (:FROM ?mes) (:TO ?met)))
  (slot-insert$ ?CommModel relationships 1 ?cm)
  (retract ?CommAct)
)

(defrule FindBehaviours
  (ModelingProduct (product BehaviourModel) (ref ?BM&~nil))
  (ModelingProduct (product CommModel) (ref ~nil))
  =>
  (foreach ?concept (slot-get ?BM concepts)
      (if (eq JADE\ Behaviour (class ?concept)) then
         (bind ?beh (slot-get ?concept name))
         (store BEHAVIOURS (bind ?*v_beh*  (union$ ?*v_beh* (create$ ?beh))))))
)



(defrule FindMess
  (ModelingProduct (product CommModel) (ref ~nil))
  =>
  (bind ?name (str-cat "Model Of Communicative Acts Of " ?*project*))
```

```
 (bind ?CommModel (GetConcept "Model of Communicative Acts" ?name))
 (foreach ?concept (slot-get ?CommModel concepts)
      (if (eq FIPA-ACL\ Message (class ?concept)) then
         (bind ?mes (slot-get (slot-get ?concept object\ content) name))
         (store MESSAGES (bind ?*v_mes*  (union$ ?*v_mes* (create$ ?mes))))))

 (foreach ?concept (slot-get ?CommModel concepts)
      (if (eq Method\ JADE\ Behaviour (class ?concept)) then
         (bind ?metbeh (slot-get ?concept name))
         (store METBEHAVIOURS (bind ?*v_metbeh*  (union$ ?*v_metbeh* (create$ ?metbeh))))))
```

3.3. Modelagem dos atos de comunicação

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;
(defrule BehaviourModelingTaskCreate
 (ModelingTask (task DomainImplementation) (ref ?DomainImplementationTask&~nil))
 ?BehaviourModelingTask <- (ModelingTask (task BehaviourModeling) (ref nil))
 =>
 (bind ?ref (make-instance of "Behaviours Modeling" (name (str-cat "Behaviours Modeling Of " ?*project*))))
 (bind ?*ia_created* (+ 1 ?*ia_created*))
 (modify ?BehaviourModelingTask (ref ?ref))
 (slot-insert$ ?DomainImplementationTask subtasks 1 ?ref)
 (assert (ModelingProduct (product BehaviourModel)))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;
(defrule BehaviourModelTransform
 (ModelingProduct (product ImplementationModel) (ref ?ImplementationModel&~nil))
 (ModelingTask (task BehaviourModeling) (ref ?BehaviourModeling&~nil))
 ?BehaviourModel <- (ModelingProduct (product BehaviourModel) (ref nil))
 =>
 (bind ?name (str-cat "Model Of Behaviours Of " ?*project*))
 (bind ?model (make-instance of "Model of Behaviours" (name ?name)))
 (bind ?*im_created* (+ 1 ?*im_created*))
 (modify ?BehaviourModel (ref ?model))
 (slot-set ?BehaviourModeling products ?model)
 (slot-insert$ ?ImplementationModel subproducts 1 ?model)

 (bind ?MAS (GetConcept "Multi-agent Society Model" (str-cat "Multi-agent Society Model Of " ?*project*)))
 (foreach ?concept (slot-get ?MAS concepts)
   (if (and (eq Agent (class ?concept))) then
      (bind ?ja (make-instance of "JADE Agent" (name (slot-get ?concept name))))
      (slot-insert$ ?model concepts 1 ?ja)
```

```
      (foreach ?resp (slot-get ?concept in\ charge\ of)
         (bind ?jb (make-instance of "JADE Behaviour" (name (slot-get ?resp name))))
         (slot-insert$ ?ja plays 1 ?jb)
         (slot-insert$ ?model concepts 1 ?jb)

         (foreach ?ac (slot-get ?resp exercised\ through)
            (bind ?jm (make-instance of "Method JADE Behaviour" (name (slot-get ?ac name))))
            (slot-insert$ ?jb has 1 ?jm)
            (slot-insert$ ?model concepts 1 ?jm)
         )))
  )
)

(defrule SetBehType
  ?BehType <- (Concept (class BehaviourType) (related ?behav) (kind ?kind))
  =>
  (bind ?name (str-cat "Model Of Behaviours Of " ?*project*))
  (bind ?BM (GetConcept "Model of Behaviours" ?name))
  (foreach ?concept (slot-get ?BM concepts)
      (printout t ?behav " -> " ?kind " " (slot-get ?concept name) crlf)
      (if (and (eq JADE\ Behaviour (class ?concept)) (eq ?behav (slot-get ?concept name))) then
         (slot-set ?concept type ?kind)))
  (retract ?BehType)
)


(defrule FindBehaviours
  =>
  (bind ?name (str-cat "Model Of Behaviours Of " ?*project*))
  (bind ?BM (GetConcept "Model of Behaviours" ?name))
  (foreach ?concept (slot-get ?BM concepts)
      (if (eq JADE\ Behaviour (class ?concept)) then
         (bind ?beh (slot-get ?concept name))
         (store BEHAVIOURS (bind ?*v_beh*  (union$ ?*v_beh* (create$ ?beh))))))
)
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo