UNIVERSIDADE FEDERAL DO MARANHÃO
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE
ÁREA: CIÊNCIA DA COMPUTAÇÃO

UMA TÉCNICA E UMA FERRAMENTA PARA O PROJETO DE DOMÍNIO GLOBAL E DETALHADO DE SISTEMAS MULTIAGENTE

Steferson Lima Costa Ferreira

São Luís, Ma
2004

UMA TÉCNICA E UMA FERRAMENTA PARA O PROJETO DE DOMÍNIO GLOBAL E DETALHADO DE SISTEMAS MULTIAGENTE

Steferson Lima Costa Ferreira

Bacharel em Ciência da Computação

Universidade Federal do Maranhão, 2002

Dissertação apresentada ao curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica na área de Ciência da Computação.

Orientadora*: Profa. Dra. Rosario Girardi*

São Luís, Ma
2004

UMA TÉCNICA E UMA FERRAMENTA PARA O PROJETO DE DOMÍNIO GLOBAL E DETALHADO DE SISTEMAS MULTIAGENTE

Steferson Lima Costa Ferreira

Dissertação aprovada em / /

_____

Profª. Drª. Maria del Rosario Girardi
Universidade Federal do Maranhão
(Orientadora)

_____

Prof. Dr. Zair Abdelouahab
Universidade Federal do Maranhão

_____

Prof. Dr. Evandro de Castro Barros
Universidade Federal de Alagoas

A meus pais e à minha avó querida.

"Se você já conseguiu tudo o que planejou, você não planejou o suficiente."

Desconhecido

AGRADECIMENTOS

Agradeço a todos aqueles que de alguma forma contribuíram para a elaboração deste trabalho, de modo especial:

Aos meus pais Ana e Estevam pelo amor e pela presença em toda minha vida.

À Maria de Jesus Miranda Lima pelo afeto e carinho dispensados durante os anos.

À João Ribeiro Lima (in memorian) pelos inesquecíveis momentos compartilhados e valiosos exemplos de honestidade.

Aos meus familiares, pelos exemplos de persistência em busca de um objetivo.

Aos amigos e companheiros, em especial a Ismênia, Carla, Geovanne, Alisson, Leandro e Ivo, que em vários momentos desta caminhada me ajudaram.

À Professora Rosário pelo apoio e dedicação indispensável para a realização deste trabalho.

RESUMO

Este trabalho propõe a DDEMAS, uma técnica para a construção de frameworks multiagente baseados em padrões e ontologias. A técnica é uma nova abordagem para o Projeto de Domínio na Engenharia de Domínio Multiagente.

A técnica consiste de três fases, através das quais é especificado o projeto de uma família de aplicações multiagente em ordem crescente de detalhamento: *modelagem de agentes e interações*, *projeto global* e *projeto detalhado*. A primeira fase visa a definição dos componentes agentes e suas interações. Na segunda, é feito o projeto global da família de sistemas, com os agentes organizados segundo mecanismos de coordenação e cooperação apropriados. Na ultima fase, cada agente é analisado individualmente em termos comportamentais e de conhecimento.

É também proposta a ONTODD, uma ferramenta que modela o conhecimento da técnica DDEMAS, permitindo a aplicação da técnica e guiando o processo de criação de frameworks multiagente.

A construção de frameworks é feita através da instanciação das meta-classes relativas aos conceitos da modelagem e produtos da modelagem da ONTODD.

A DDEMAS e a ONTODD foram avaliadas através do desenvolvimento de um estudo de caso, onde um framework baseado em ontologias para o acesso a informação foi construído.

Palavras - chave: Frameworks, Sistemas Multiagente, Ontologias, Projeto de Domínio, Engenharia de Domínio.

Fonte: Ferreira, Steferson L. C. Uma técnica e uma ferramenta para o Projeto de Domínio global e detalhado de sistemas multiagente. 2004. 143 p. Dissertação (Mestrado em Engenharia de Eletricidade), Universidade Federal do Maranhão, São Luís.

ABSTRACT

This work proposes DDEMAS, a technique for the construction of multi-agent frameworks based on patterns and ontologies. DDEMAS is a new approach for Domain Design in Multi-agent Domain Engineering.

The technique consists of three phases, through which the architectural and detailed design of a family of multi-agent applications is specified in a growing order of detail: modelling of agents and interactions, global design and detailed design. The first phase looks for the definition of agents and interactions. On the second one, it is made the global design of the family of systems, with the agents organized according to coordination and cooperation mechanisms. In the last phase, each agent is analyzed individually in terms of behavior and knowledge.

The work also proposes ONTODD, an ontology-based tool which encodes the knowledge of DDEMAS, thus allowing the application of the technique for the creation of multi-agent frameworks.

The frameworks construction is made through the instanciation of the metaclasses corresponding to the concepts of modelling and products of modelling of ONTODD.

DDEMAS and ONTODD have been evaluetad through the development of a case study where an ontology-based framework for building information access applications has been constructed.

Keywords: Frameworks, Multi-agent Systems, Ontologies, Domain Design, Domain Engineering

Source: Ferreira, Steferson L. C. A technique and a tool for global and detailed Domain Project of multiagent systems. 143 p. Thesis (Graduation in Eletrical Engineering), Universidade Federal do Maranhão, São Luís.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

ABREVIATURAS E SÍMBOLOS

AIP	Agents interaction protocol
APSMAD	Architectural pattern system for multi-agent application development
APSUMAS	Agent-based pattern system for user modeling and adaptation of systems
AUML	Agent Unified Modeling Language
AWT	Abstract Windowing Toolkit
CFRP	Conceptual Framework for reuse process
DARTS	Design approach for real-time systems
DDEMAS	Domain design for multi-agent systems
FODA	Feature Oriented Domain Analysis
GAIA	Generic Architecture for Information Availabity
GESEC	Grupo de pesquisa em Engenharia de Software e Engenharia de Conhecimento
GRAMO	Generic Requirements Analysis Method based on Ontologies
MaAE	Methodology based on Agents for the Development of Software
MADS	Multi-agent Application Engineering
MaSE	Multi-agent Systems Engineering
ONTODD	Generic Ontology for Domain Design
ONTODD_INFO	Ontology for Domain Design
ONTODUM	Generic Ontology for Domain and User Modelling
ONTOINFO	Ontology for Domain and User Modelling in informartion access
OSI	Open systems interconnection
PASSI	Process for Agent Societies Specification and Implementation
SMA	Sistemas multiagente
UML	Unified Modeling Language

1. Introdução

Na Engenharia de Software existem vários modelos de desenvolvimento: clássico, interativo incremental, baseado na reutilização, entre outros. A necessidade de diminuição do tempo de construção do software, bem como a redução dos custos, fez com que abordagens que se baseiam na reutilização crescessem, tendo lugar de destaque entre as várias abordagens. A reutilização pode ser realizada nas diferentes fases do ciclo de desenvolvimento de um produto de software. A utilização de padrões de análise é um exemplo de reuso na fase de análise. O desenvolvimento baseado em frameworks e padrões de projeto caracterizam o reuso na fase de projeto, entre outras práticas de reutilização.

Um framework é uma estrutura de componentes inter-relacionados, que corresponde à implementação de uma arquitetura de software incompleta para uma família de sistemas de um domínio específico. Os frameworks facilitam a reutilização tanto no nível de projeto quanto no nível de implementação, reduzindo o tempo e o esforço na construção de software. Os padrões de projeto fornecem soluções bem sucedidas para a organização estrutural e detalhada de software e pode ser uma excelente fonte para a construção de frameworks.

A construção de software baseada na reutilização é geralmente feita de forma sistemática objetivando redução do tempo de desenvolvimento, aumento de produtividade, confiabilidade e qualidade, objetivos os quais por estar-se construindo software alicerçado em artefatos de software já desenvolvido e testado são mais facilmente alcançados.

A reutilização está centrada em duas linhas. Uma delas foca a construção de artefatos de software reutilizáveis, ou seja, desenvolvidos PARA serem aplicados e adequados em sistemas semelhantes. A outra linha de reutilização existente é a construção de software baseando-se em artefatos reutilizáveis, ou seja, a construção COM reuso, onde após a análise de artefatos reutilizáveis disponíveis, estes são aplicados numa situação específica.

A reutilização existe na análise, com a reutilização de padrões de análise e/ou especificações genéricas de requisitos; no projeto, com reutilização de soluções computacionais e no nível de implementação com a reutilização de linhas de código ou lógica programática. Porém, o reuso nas fases de análise e projeto mostra-se mais significativo, pois traz consigo um conjunto integrado de artefatos que compõem uma determinada solução computacional, ao invés de artefatos isolados.

O desenvolvimento baseado em agentes, assim como a reutilização, é uma abordagem de grande importância na Engenharia de Software, pelas suas características que viabilizam a resolução de problemas complexos. Inúmeras pesquisas estão sendo desenvolvidas nesta área, definições de conceitos, formalizações, protocolos, técnicas e métodos para aplicação deste tipo de abordagem na concepção de software [WOOLDRIDGE, 2000][DILEO, 2002][MYLOPOULOS, 2000][COSSENTINO, 2002].

1.1. Contexto de Pesquisa

MaAE (Multi-agent Application Engineering) é um projeto desenvolvido no contexto do grupo de pesquisa GESEC (Grupo de pesquisa em Engenharia de Software e Engenharia de Conhecimento) [GIRARDIc, 2003], que busca a sistematização de metodologias de desenvolvimento para a Engenharia de Domínio Multiagente e para a reutilização de seus produtos no desenvolvimento de aplicações específicas.

A Figura 1 mostra o modelo proposto para a Engenharia de Domínio Multiagente[GIRARDI, 2004], suas fases (Análise de Domínio, Projeto de Domínio e Implementação de Domínio) e os produtos gerados em cada fase. O conhecimento das técnicas utilizadas em cada fase bem como os produtos gerados em cada uma delas são representados em ontologias.

Na Figura 1, os produtos gerados em cada fase são ilustrados considerando seu nível de abstração (do mais abstrato ao mais concreto) e seu nível de dependência de um domínio de aplicação (do mais dependente ao mais independente): Modelos de Domínio, Modelos de Usuário, Padrões arquiteturais e de projeto detalhado, Frameworks multiagente e Agentes genéricos.

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

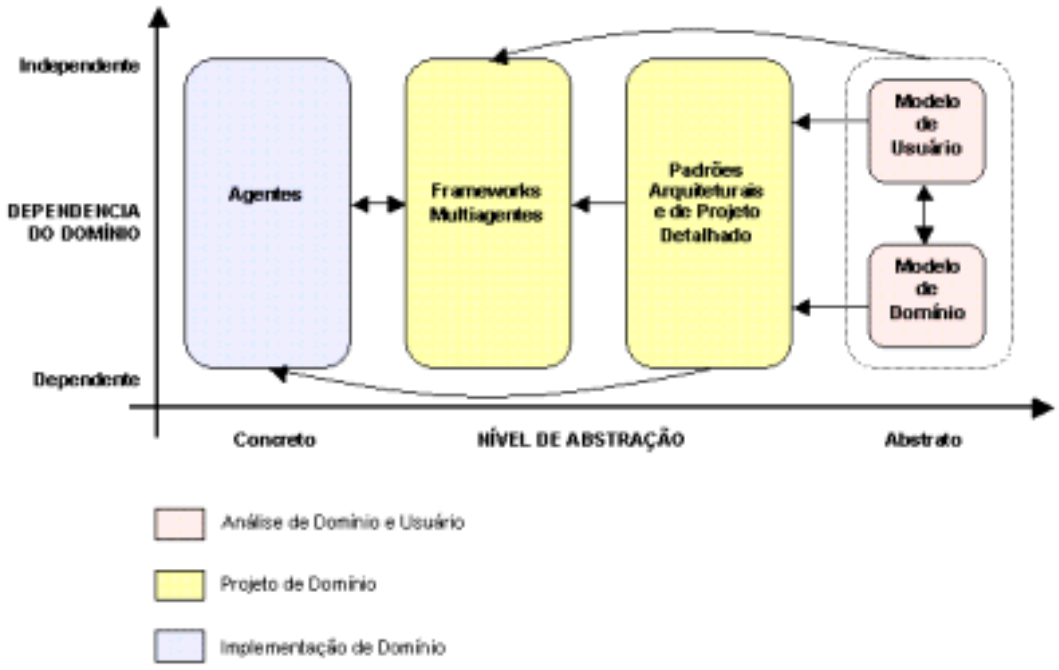

Figura 1. Fases e produtos da Engenharia de Domínio Multiagente

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

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

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

Este projeto trata-se de um empreendimento realizado em parceria entre professores e pesquisadores das Universidades Federal de Alagoas (UFAL), do Maranhão (UFMA) e de Santa Catarina (UFSC) que integra experiências interdisciplinares vindas das áreas da Engenharia de Software e da Inteligência Artificial e propõe aplicações concretas de resultados já obtidos de projetos de pesquisa em andamento. A proposta está sendo avaliada através do desenvolvimento de vários estudos de caso envolvendo a tecnologia multiagente, como o Aceso à Informação [GIRARDIc, 2003].

1.2. Objetivo do trabalho

Este trabalho tem como objetivo principal a elaboração de uma técnica e uma ferramenta para o projeto de domínio global e detalhado de sistemas multiagente, enfatizando a reutilização de padrões de projeto e permitindo a construção de frameworks multiagente através da utilização de ontologias como forma de especificação.

1.3. Organização do manuscrito

A presente dissertação é constituída por seis capítulos.

O segundo capítulo apresenta os conceitos básicos do projeto de software, das arquiteturas de software e dos estilos arquiteturais. A visão geral da Engenharia de domínio é também apresentada, enfatizando os métodos relevantes para a fase de Projeto de domínio, o uso de framework e padrões de software.

O terceiro capítulo aborda o projeto de sistemas multiagente, mostrando os conceitos de agentes e sistemas multiagente. São analisadas as técnicas de projeto existentes para o desenvolvimento de sistemas multiagente e mostrados exemplos de aplicação dos mesmos.

O quarto capítulo apresenta a técnica DDEMAS para o projeto de domínio de sistemas multiagente, detalhando e exemplificando suas fases e atividades.

No quinto capítulo é feita a especificação da ferramenta ONTODD, uma ontologia para o projeto de sistemas multiagente que modela o conhecimento da técnica DDEMAS.

No sexto capítulo é apresentado um estudo de caso, o qual objetiva avaliar a técnica DDEMAS e a ferramenta ONTODD através da construção de um framework para o acesso à informação.

No sétimo e último capítulo são apresentadas as considerações finais do trabalho, destacando os resultados obtidos e os trabalhos futuros que poderão ser desenvolvidos a partir desta pesquisa.

2. O Projeto de software

O processo de desenvolvimento de software consiste de etapas que vão da análise de requisitos até sua implantação (Figura 2). Um dos pontos decisivos nesse caminho é o projeto do software, pois é nesta fase que é definida uma solução computacional ao problema especificado na fase de análise de requisitos.

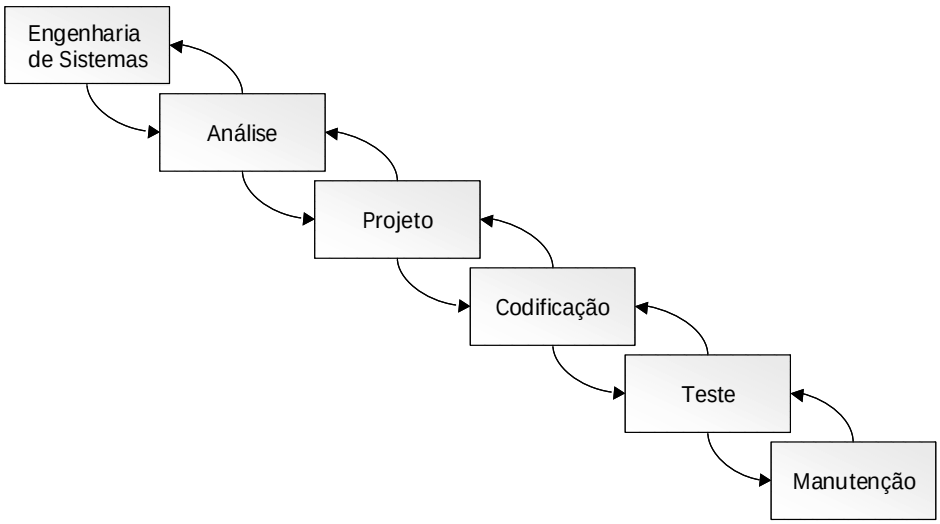

Figura 2. Ciclo de desenvolvimento de software [YOURDON, 1990]

Durante a fase de análise o problema é dividido em partes menores que, na fase de projeto serão computacionalmente resolvidos através da construção de um conjunto de módulos de software. O conjunto destes módulos e seus mecanismos de cooperação e coordenação compõem a *Arquitetura do software*. No projeto detalhado, esta arquitetura é analisada em detalhe em termos de processamento e dados internos de cada módulo.

2.1. Arquiteturas de software

As arquiteturas de software surgiram como uma evolução natural das abstrações de projeto, na busca de novas formas de construir sistemas de software maiores e mais complexos [SHAW, 1996].

A arquitetura do sistema é muito importante no processo de construção de sistemas por fornecer uma visão do funcionamento do sistema; visões sobre performance, confiabilidade, portabilidade, escalabilidade e interoperabilidade [GARLAN, 1994]; e, por isso, fornece uma garantia de que o software irá satisfazer o objetivo global que ele pretende alcançar.

Em uma definição formal uma arquitetura de software é a principal parte do projeto, mostrando como as partes que compõem o sistema interagem; onde ocorrem as interações e quais são as principais propriedades das partes, dando assim uma descrição que serve para a análise e avaliação do sistema.

A arquitetura tem vários papéis dentro do processo de desenvolvimento de sistemas, entre eles [GARLAN, 1994] [FERREIRA, 2002]:

- Entendimento: facilita e simplifica a compreensão do sistema, pois o exibe como uma abstração de alto nível.
- Reuso: pode-se fazer o reuso em vários níveis, reuso de concepções de módulos, simples módulos ou frameworks, nos quais os módulos estão integrados.
- Construção: fornece um esqueleto inicial para o desenvolvimento, mostrando os principais componentes e as dependências entre eles.
- Evolução: a arquitetura pode expor de forma mais clara a dimensão do sistema, separando os conceitos de funcionalidade dos meios pelos quais os componentes estão conectados, distinguindo explicitamente componentes e mecanismos. Esta clara separação facilita a evolução da arquitetura, pois permite mudanças em mecanismos de conexão, sem perda de performance, interoperabilidade.
- Análise: permite checagem de consistência, como por exemplo a conformidade das restrições impostas pelo estilo arquitetural ou análise de dependência.

A importância das arquiteturas de software está na constatação de que determinadas estruturas de software são adequadas para determinados tipos de problema de projeto. Por exemplo, o estilo arquitetural canais e filtros [SHAW, 1996], em que fluxos de dados são transportados linearmente através de um conjunto de componentes de software, tem se mostrado adequado e por isso, vem sendo usado para a construção de compiladores. Cada componente construtor de um compilador recebe um conjunto de dados, realiza transformações nestes dados e disponibiliza o resultado. Este tipo de experiência no desenvolvimento de software produziu várias estruturas organizacionais de software que são tratadas como *estilos arquiteturais.*

2.2. Estilos arquiteturais

Um estilo arquitetural define uma topologia de conexão específica, isto é, uma forma de interconectar os componentes de uma arquitetura, o que restringe as possibilidades de interação entre os componentes da topologia. Os principais elementos de uma arquitetura de software são os componentes que a constituem. O que caracteriza um estilo arquitetural, porém, é a forma como os componentes estão conectados e como interagem. A seguir, são apresentados alguns estilos arquiteturais [SHAW, 1996] [BUSCHMANN, 1996].

2.2.1. Canais e filtros

A Figura 3 ilustra o estilo arquitetural *Canais e filtros*. Os elementos que executam processamento são os filtros, que são componentes de software com entradas, por onde recebem fluxos de dados, e saídas, por onde disponibilizam fluxos de dados resultantes de sua atuação sobre os fluxos de entrada. Canais interligam entradas e saídas de filtros, estabelecendo uma topologia. A forma de interação característica deste estilo arquitetural é o fluxo de dados unidirecional.

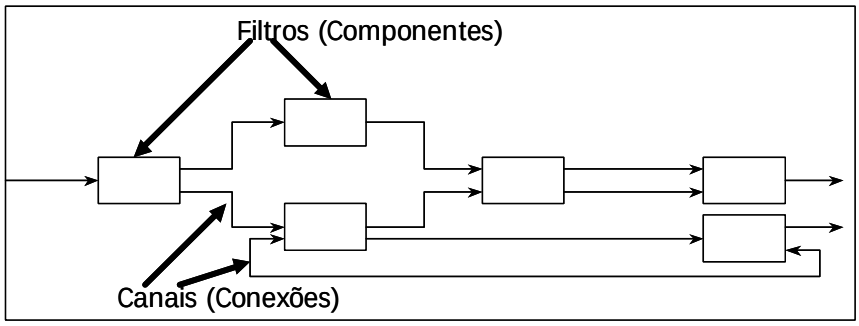

Figura 3. Estilo arquitetural canais e filtros

Um dos aspectos importantes neste estilo é a condição de que os filtros devem ser independentes e não conhecem a identidade dos outros filtros no processo.

Algumas das vantagens deste estilo são: permite ao projetista entender o comportamento global das entradas e saídas de um sistema como a composição simples de filtros individuais; suporta o reuso; os sistemas se tornam de fácil manutenção e de expansão, pois filtros podem ser inseridos e/ou filtros podem ser melhorados e suporta a execução concorrente.

Por outro lado têm-se algumas desvantagens. Este estilo freqüentemente se direciona a uma organização do tipo *batch*, não sendo ideal para sistemas interativos; outra desvantagem é que se tem que manter a correspondência entre filtros independentes que possuem fluxos comuns.

Um dos exemplos mais conhecidos do uso de *Canais* e f*iltros* são os programas escritos no Unix Shell [BACH, 1986]. Unix dá suporte a este estilo provendo uma notação para conectar os componentes (representados como processos do Unix) e provendo mecanismos de tempo de execução para implementar os Canais. Um outro exemplo de aplicação conhecido são os compiladores. Tradicionalmente os compiladores têm sido vistos como sistemas *pipeline* (entretanto as fases não são freqüentemente incrementais). Os estágios no *pipeline* incluem análise lexica, *parsing*, análise semântica e geração de código. Outros exemplos de *canais e filtros* existem em domínios de processamento de sinais [DELISLE, 1990], programação funcional [KAHN, 1974] e sistemas distribuídos [BARBACCI, 1988].

2.2.2. Sistemas em camadas

Um sistema em camadas é organizado hierarquicamente, onde cada camada provendo serviço à camada acima e servindo como um cliente à camada abaixo. As conexões estão definidas pelos protocolos que determinam como as camadas irão interagir. Restrições deste estilo incluem a limitação de interações somente entre camadas adjacentes. A Figura 4 ilustra este estilo.

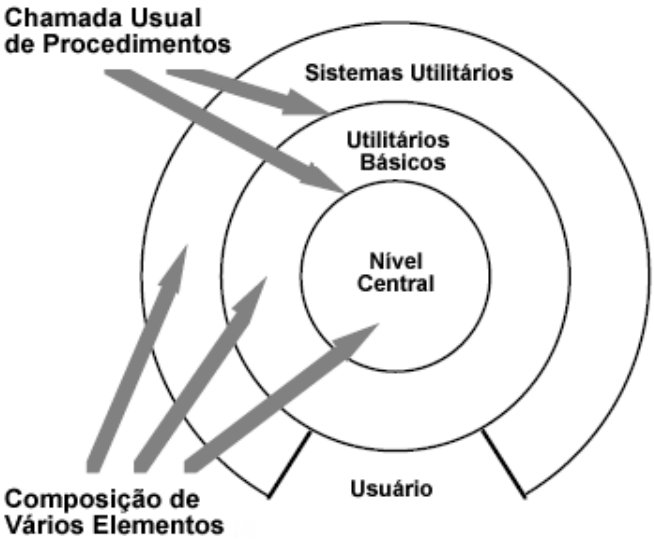

Figura 4. Exemplo de sistema em camadas

Os exemplos mais conhecidos deste tipo de estilo arquitetural são os protocolos de comunicação em camadas [MCCLAIN, 1991]. Nesta área de aplicação cada camada provê um substrato para a comunicação em algum nível de abstração. Os níveis mais baixos definem áreas com um número reduzido de interações, com o mais baixo dos níveis tipicamente tratando conexões de hardware. Outras áreas de aplicação para este estilo incluem sistemas de banco de dados e sistemas operacionais [FRIDRICH, 1985] [BATORY, 1991] [LAUER,1979].

Os sistemas em camadas possuem várias vantagens. Primeiro, eles dão suporte ao projeto em níveis crescentes de abstração. Isto permite aos desenvolvedores dividir um problema complexo em uma sucessão de passos incrementais. Segundo, eles dão suporte ao melhoramento contínuo, como nos "*pipelines*", pois cada camada interage com no máximo duas camadas, as camadas acima e abaixo; mudanças em uma determinada camada afetam no máximo duas outras camadas. Terceiro, eles dão suporte ao reuso, como os tipos abstratos de dados, onde diferentes implementações da mesma camada podem ser reutilizadas, contanto que eles dêem suporte às mesmas interfaces. Isto conduz à possibilidade de definir uma camada de interface padrão na qual diferentes desenvolvedores podem utilizar diferentes implementações. Um bom exemplo é o modelo OSI ("*Open systems interconnection*") [ZIMMERMANN, 1980].

Mas os sistemas em camadas também têm desvantagens. Nem todos os sistemas são estruturados facilmente em forma de camadas. E até mesmo se um sistema pode ser estruturado logicamente como camadas, considerações de desempenho podem requerer uma junção mais próxima das funções de alto nível e as suas implementações de baixo-nível. Adicionalmente, pode ser bastante difícil achar os níveis certos de abstração. Isto é particularmente verdade para modelos em camadas padronizados. Um exemplo de dificuldade encontrada pela comunidade pesquisadora em comunicações, foi mapear os protocolos existentes no framework ISO [ZIMMERMANN, 1980]: muitos desses protocolos atravessam várias camadas.

Os sistemas em camadas são semelhantes ao *pipeline*, em que componentes comunicam no máximo com um outro componente em qualquer lado. Mas em vez de um protocolo leitura/escrita dos canais, os sistemas em camadas podem prover formas mais ricas de interação. Isto dificulta a definição de camadas independentes, como nos filtros, pois uma camada tem que suportar os protocolos específicos a seus limites superiores e inferiores. Mas

eles também permitem uma interação mais próxima entre as camadas, e a transmissão de informações nos dois sentidos.

2.2.3. Sistemas baseados em invocação implícita

Tradicionalmente, em um sistema no qual as interfaces de componentes provêem uma coleção de procedimentos e funções, os componentes interagem entre si invocando essas rotinas explicitamente. No estilo invocação implícita, a comunicação é baseada na propagação ("*event broadcasting*") de eventos. Os componentes podem estar ou não habilitados para reagir à ocorrência de um evento. A reação consiste na execução de algum procedimento. Quando um componente divulga um evento, os componentes habilitados a reagirem àquele evento executam seus respectivos procedimentos. O componente iniciador do evento não invoca procedimentos de outros componentes diretamente, mas a ocorrência do evento acarreta em invocações, daí o nome de *invocação implícita*.

Um problema neste estilo é que, quando ocorre um evento, não se sabe quais componentes são por ele afetados, ficando dúvidas sobre se processamentos ocorreram ou quais processamentos irão ocorrer após tal evento. Por este motivo, este estilo também se utiliza em algumas situações do método explícito de invocação.

Um benefício deste estilo é que ele provê o reuso e ainda facilita a evolução do sistema, pois componentes podem ser inseridos, apenas necessitando seu registro nos eventos do sistema, ou substituídos sem afetar a interface dos outros componentes no sistema. A desvantagem primária é que os componentes renunciam ao controle, dando-o para o sistema.

São vários os exemplos de Sistemas baseados em invocação implícita [GARLAN, 1988]. Eles são usados em ambientes de programação, para integrar ferramentas [GERETY, 1989] [REISS, 1990]; em sistemas de administração de banco de dados para assegurar restrições de consistência [HEWITT, 1969] [BALZER, 1986]; em interfaces de usuário para separar a apresentação de dados da aplicação que administra os dados [KRASNER, 1988] [SHAW, 1983]; e por editores dirigidos à sintaxe, para suportar a verificação semântica incremental [HABERMANN, 1986] [HABERMANN, 1991].

2.2.4. Sistemas baseados em repositórios

A Figura 5 ilustra o estilo arquitetural baseado em repositórios, também conhecido como o quadro-negro ou *blackboard*. Neste estilo subsistemas independentes (fontes de conhecimento) compartilham um repositório de dados. Os subsistemas não interagem diretamente entre si. A forma de interação possível entre subsistemas consiste na manipulação das informações compartilhadas através do repositório. A interação entre subsistemas e o repositório pode ocorrer através de compartilhamento de área de memória. Em aplicações mais complexas, o repositório pode ser um banco de dados ou alguma estrutura mais complexa. Uma vantagem dos sistemas baseados em repositórios é sua flexibilidade. Na medida em que os subsistemas que compartilham um repositório não têm dependências entre si, torna-se mais fácil alterar o sistema através de substituição, remoção, inclusão ou alteração de subsistemas sem que os demais sejam afetados.

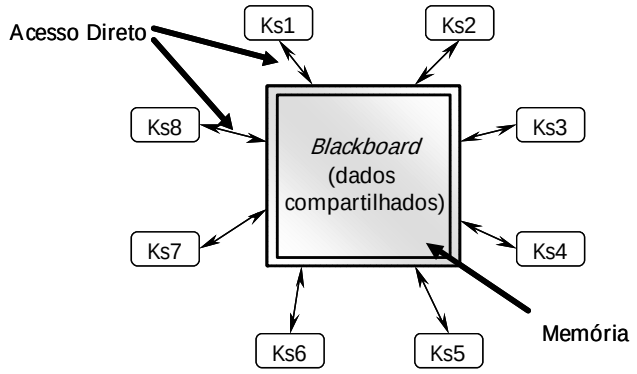

Figura 5. Exemplo de um sistema baseado em repositórios

Sistemas de quadro-negro têm sido tradicionalmente usados em aplicações que requerem interpretações complexas de processamento de sinais, como reconhecimento de padrões e de fala. Eles também apareceram em outros tipos de sistemas que envolvem acesso compartilhado a dados com agentes livremente agrupados [AMBRIOLA, 1990].

Existem muitos outros exemplos de sistemas de quadro-negro. Sistemas Batch seqüenciais com bancos de dados compartilhados são um caso especial. Ambientes de programação são freqüentemente organizados como uma coleção de ferramentas juntamente com um repositório compartilhado de programas e fragmentos de programa. Até mesmo aplicações vistas tradicionalmente como arquiteturas *pipeline*, podem ser analisadas mais

precisamente como sistemas de quadro-negro. Como por exemplo, uma arquitetura de um compilador vista tradicionalmente como *pipeline*, as “fases” da maioria dos compiladores modernos operam em uma base de informação compartilhada (tabelas de símbolo, árvores abstratas de sintaxe).

2.2.5. Comparativo dos estilos arquiteturais

Na Tabela 1 são apresentados os vários estilos arquiteturais analisados, descrevendo-os de forma comparativa de acordo com suas vantagens, desvantagens e aplicações.

Estilo	Vantagens	Desvantagens	Aplicações
Canais e filtros	- Permite melhor entendimento do comportamento global das entradas e saídas do sistema como uma composição simples de filtros individuais; - Suporta o reuso; - Os sistemas são de fácil manutenção e expansão.	- Se direciona a uma organização do tipo *batch*, não sendo ideal para sistemas interativos; - Dificuldade de manter a correspondência entre filtros independentes quem possuem fluxos comuns.	- Programas escritos no Unix Shell; - Compiladores; - Processamento de sinais; - Programação funcional ; - Sistemas distribuídos..
Sistemas em camadas	- Suporte ao projeto em níveis crescentes de abstração; - Suporte ao melhoramento contínuo; - Suporte ao reuso.	- Nem todos os sistemas são estruturados facilmente em forma de camadas; - Dificuldade em achar os níveis certos de abstração.	- Protocolos de comunicação em camadas; - Sistemas de banco de dados ; - Sistemas operacionais.
Sistemas baseados em invocação implícita	- Suporta o reuso ; - Facilita a evolução do sistema.	- Ao ocorrer um evento, não se sabe quais componentes são afetados por ele; - Os componentes renunciam ao controle, dando-o para o sistema.	- Ferramentas de desenvolvimento; - Sistemas de administração de banco de dados; - Interfaces de usuário; - Editores dirigidos à sintaxe.
Sistemas baseados em repositório	- Flexibilidade; - Fácil alteração do sistema através de substituição, remoção, inclusão ou alteração de subsistemas. - Suporte ao reuso das fontes de conhecimento.	- Documentação e suporte limitados.	- Sistemas *batch* seqüenciais com bancos de dados compartilhados; - Ambientes de programação.

Tabela 1. Tabela comparativa dos estilos arquiteturais

2.3. O reuso no projeto de software

O objetivo da reutilização na construção de software é auxiliar nas várias fases e etapas do processo de desenvolvimento, reutilizando componentes previamente desenvolvidos. O reuso permite a aplicação de abordagens gerativas, que possuem, geralmente, três características muito importantes para o atendimento da demanda de mercado:

- Qualidade;
- Baixo custo;
- Rapidez no seu desenvolvimento.

A prática da reutilização implica em um aumento da produtividade da equipe de desenvolvimento, da qualidade e confiabilidade do produto, além de contribuir para a diminuição dos custos. Porém, esta prática ainda não é tida como parte indispensável no desenvolvimento de software, resultando, assim, em uma baixa produtividade no desenvolvimento e em produtos de baixa qualidade. As principais razões para esse fato podem ser divididas em dois grupos de fatores [GUIZZARDI, 2000]:

a) *Fatores tecnológicos*: para que os componentes produzidos possam ser reutilizados, é fundamental que eles possam ser encontrados com facilidade. Por isso se faz necessário a construção de ambientes de desenvolvimento, que possam agrupar características que auxiliem os desenvolvedores a localizar componentes a serem utilizados; entender os serviços oferecidos, pré-requisitos e conseqüências da utilização de um determinado componente e identificar oportunidades de reuso, entre outras coisas. Por outro lado, os ambientes devem facilitar a classificação e catalogação de componentes desenvolvidos para reuso e de suas respectivas informações. Além disso, deve haver uma política bem definida para a catalogação de componentes, de modo a evitar a proliferação de componentes com pouca possibilidade de reuso.

b) *Fatores culturais/econômicos*: Poulin [POULIN,1997] defende a crença de que, do ponto de vista tecnológico, a área já atingiu maturidade suficiente para que o reuso possa ocorrer e salienta o fato de que, na verdade, esse processo é guiado por questões organizacionais. Ele critica a falta de trabalhos científicos que abordem esses aspectos não tecnológicos e prega a disseminação do conhecimento por parte dos pesquisadores para que as empresas comecem a conhecer e aplicar estes princípios.

Segundo Poulin, enquanto não pudermos quantificar os benefícios do reuso em termos concretos como tempo e dinheiro economizados, a prática sistemática de reuso simplesmente não irá acontecer. Zand [ZAND, 1997] complementa essa idéia, afirmando que uma estratégia clara do ponto de vista organizacional é hoje o maior obstáculo para a adoção de um desenvolvimento para/com reuso em escala industrial e que as organizações geralmente desistem de um projeto de implantação da cultura de reuso sempre que têm que se confrontar com problemas orçamentários. Para ele, organizações que não possuem um plano sistemático de reuso, não possuem um sistema de desenvolvimento holístico e desconhecem as vantagens estratégicas a longo prazo, podendo ter um benefício apenas parcial do reuso de software.

Além dos aspectos que impedem a plena aplicação da reutilização, para que ela seja viável, o desenvolvimento de software precisa passar por uma etapa anterior, onde são analisadas, projetadas e implementadas as características comuns em relação ao domínio em questão. Esse processo de desenvolvimento chama-se a Engenharia de Domínio [MAGNAN,2001].

Dessa forma, podemos definir dois pontos de vista para o desenvolvimento de software: o desenvolvimento PARA reuso e o desenvolvimento COM reuso [MOORE, 1991]. No desenvolvimento para reuso são utilizadas algumas técnicas, entre elas está a Engenharia de domínio, que se preocupa com a geração de componentes reutilizáveis em um determinado domínio. No desenvolvimento com reuso são utilizadas também algumas técnicas, entre elas está Engenharia de aplicações, que se preocupa em construir aplicações reutilizando os componentes criados na Engenharia de domínio (Figura 6). A Engenharia de aplicações serve também como agente motivador para a construção de novos componentes inexistentes na base de componentes do domínio [MAGNAN,2001].

Um dos grandes problemas da reutilização de software é justamente a criação de componentes reutilizáveis. Um dos processos de desenvolvimento que busca a construção de componentes reutilizáveis é a Engenharia de domínio.

2.4. A Engenharia e o Projeto de domínio

A Engenharia de domínio representa um enfoque sistematizado da análise, projeto e implementação de domínio, sob uma perspectiva voltada para a construção de componentes reutilizáveis, criando-se um processo completo para sua especificação [WERNER, 2000].

O conhecimento do domínio é um fator muito importante a ser levado em consideração no desenvolvimento de software, pois se o desenvolvedor conhece a área de aplicação na qual está desenvolvendo irá fazer uma boa especificação de requisitos e, em conseqüência, gerar um software de boa qualidade.

A Engenharia de domínio tem como principal objetivo disponibilizar artefatos reutilizáveis que possam ser utilizados no desenvolvimento de novas aplicações. A Figura 6 mostra as três etapas da Engenharia de domínio [FOREMAN, 1996] [WERNER, 2000].

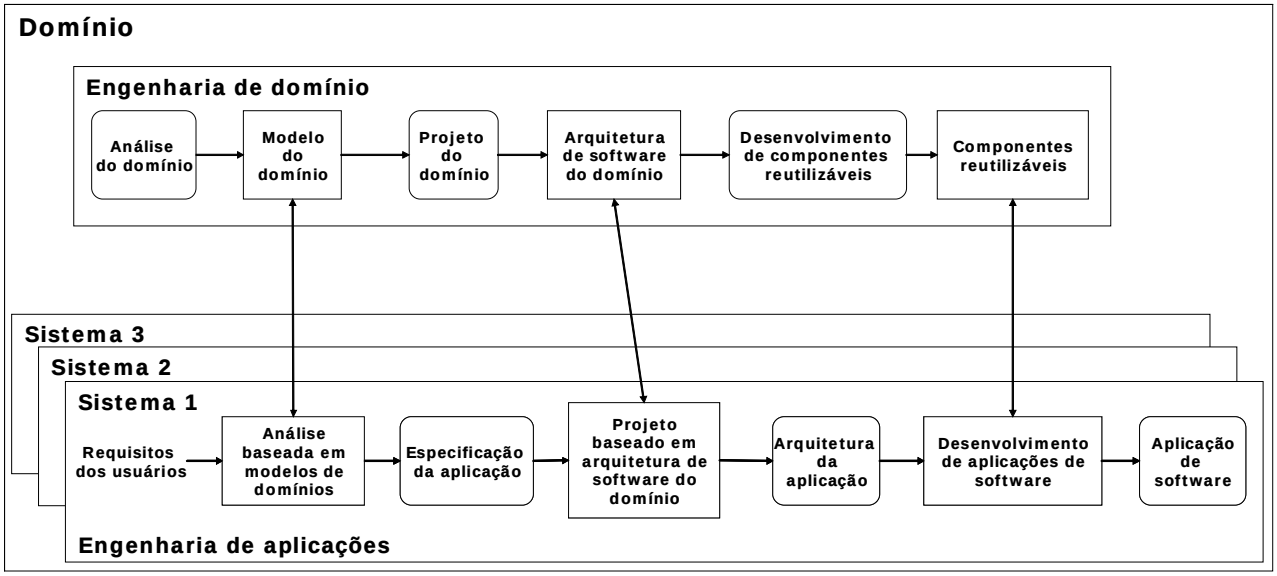

Figura 6. As fases da Engenharia de domínio e da Engenharia de aplicações

A fase de Análise de domínio tem como objetivo identificar as oportunidades para a reutilização e determinar os requisitos comuns de uma família de sistemas. O domínio do problema representa um conjunto de elementos de informação presentes em um certo contexto do mundo real, inter-relacionados de forma bastante coesa, e que despertam o interesse de uma certa comunidade. O produto desta fase é o modelo de domínio.

A fase de Projeto de domínio objetiva o refinamento das oportunidades de reutilização identificadas na fase de análise de domínio e identificar e generalizar soluções para os requisitos comuns, através da especificação de um framework (uma arquitetura de software reutilizável) e da especificação de padrões de projeto para o domínio. Os produtos desta fase são frameworks e padrões de projeto.

A fase de Implementação de domínio tem como objetivo transformar as oportunidades de reutilização e as soluções que foram identificadas na fase de Projeto de

domínio em uma implementação, que inclui os serviços de identificação, reengenharia e/ou construção e manutenção dos componentes reutilizáveis que suportem estes requisitos e soluções de projeto.

2.4.1. Frameworks

Um framework é uma estrutura de componentes inter-relacionados, que corresponde à implementação de uma arquitetura de software incompleta para um conjunto ou família de sistemas de um domínio específico. Esta estrutura de classes deve ser adaptada para a geração de aplicações específicas.

Um *framework* pode ser definido como [JOHNSON, 1997]:

- Um projeto reutilizável de uma ou todas as partes de um sistema, que é representado por um conjunto de módulos e pela maneira como eles interagem.
- Um esqueleto formado por elementos abstratos de uma família de aplicações, passível de ser adaptado para atender necessidades de uma aplicação específica.

As definições, na verdade, não são conflitantes: a primeira descreve a estrutura de um *framewor*k, enquanto a segunda descreve seu propósito. Johnson [JOHNSON, 1997] compara *frameworks* fazendo uma analogia a geradores de aplicações, no sentido de que ambos compilam uma linguagem de alto nível, específica de domínio, em uma estrutura concreta. Nesse sentido, um *framework* é também uma arquitetura específica de domínio, capaz de estruturar entidades e relações comuns ao domínio e, conseqüentemente, de prover facilidade de comunicação, uniformidade e padronização.

A relação entre *frameworks* e componentes de código decorre do fato de serem tecnologias complementares. Em primeiro lugar, *frameworks* fornecem um contexto de reuso para componentes de código, provendo uma maneira padronizada para manipulação de erros e eventos, para troca de dados e para troca de mensagens entre eles. Por outro lado, *frameworks* facilitam o desenvolvimento de novos componentes.

Por fim, *frameworks* se diferenciam de bibliotecas de classes por possibilitarem o reuso também de projeto, em um nível mais alto de abstração, e não artefatos de software isolados, sem interligações definidas.

No sentido de ser um artefato reutilizável de projeto, *frameworks* representam um compromisso entre simplicidade e poder. Geralmente possuem interfaces complexas e requerem um período de compreensão e aprendizado antes de serem aplicados. Por outro lado, constituem um tipo de componente que, quando bem concebidos, são flexíveis e poderosos, e podem reduzir o esforço de desenvolvimento de aplicações adaptadas em ordens de magnitudes. Vários frameworks têm sido implementados (AWT, JavaBeans [HORSTMANN,1997]), quase sempre focando domínios horizontais como: interfaces gráficas com o usuário, acesso ao sistema de arquivos, acesso a repositórios de dados e desenvolvimento de aplicações distribuídas. Isso se deve ao fato de que muitos desses domínios já foram amplamente estudados e compreendidos. O mesmo não é verdade para domínios verticais como Aviação, Telecomunicações, Direito e Medicina, entre outros.

Por exemplo Johnson [JOHNSON, 1997], comenta a alta complexidade envolvida na criação de componentes reutilizáveis (e conseqüentemente os altos custos e longos períodos de desenvolvimento), e prega que a academia deve se concentrar na exploração de domínios verticais e a sua concretização através de frameworks. Apesar disso, várias empresas, como Motorola [MEEKEL, 1997], Verilog [TROY, 1993] e Hewlett Packard [CORNWELL, 1996], têm relatado experiências bem sucedidas de redução do ciclo de desenvolvimento de três a quatro vezes, ao construírem arquiteturas abstratas a partir de famílias de produtos, e posteriormente promoverem a especialização de aplicações, a partir de adaptações dessas arquiteturas. A sua principal diferença em comparação a outras técnicas de reuso de projeto de alto nível, é o fato dos *frameworks* serem geralmente expressos em uma linguagem de programação, ou seja, além de representarem o projeto de software em um nível mais alto de abstração, são também artefatos de código. Essa característica traz vantagens e desvantagens. Por um lado, o fato de serem programas faz com que sejam mais fáceis de serem compreendidos e aplicados por programadores, além de não haver necessidade de outras ferramentas além das já utilizadas. Entretanto, tendem a ser específicos a uma linguagem de programação.

2.4.2. Padrões de software

Uma outra abordagem que está sendo bastante disseminada na busca da reutilização é o uso de padrões de software, uma prática recomendável para o desenvolvimento de software baseado na reutilização.

Um padrão apresenta uma solução bem sucedida a um problema recorrente. Ele mostra não só a solução, como também as suas restrições e o contexto em que se deve aplicar essa solução. Um padrão possui uma forma ou estrutura identificável e reconhecida em diferentes domínios que ajudam a diminuir a complexidade do software em várias fases do seu ciclo de vida [BUSCHMANN, 1996].

2.4.2.1 Coletâneas de padrões

As diferentes categorias de padrões podem ser agrupadas em coletâneas que têm por objetivo reunir padrões segundo algum critério. As coletâneas podem ser divididas em:

Coleções de padrões: é uma coletânea qualquer com padrões que não possuem nenhum vínculo entre si e em geral, não possuem nenhuma padronização no formato de apresentação.

Catálogos de padrões: é uma coletânea de padrões relacionados, são relacionados apenas fracamente ou informalmente e pode oferecer um esquema de classificação e recuperação [APPLETON, 2002].

Sistemas de padrões: é um conjunto coeso de padrões co-relacionados que trabalham juntos para apoiar a construção e evolução de arquiteturas completas [APPLETON, 2002] [BUSCHMANN, 1996].

Linguagem de padrões: é uma coleção de padrões que trabalham juntos para resolver um problema complexo dentro de uma solução ordenada de acordo com uma meta pré-definida [APPLETON, 2002]. Coplien [COPLIEN, 2003] define uma linguagem de padrões como uma coleção de padrões e regras para combiná-los em um estilo arquitetural. A linguagem de padrões descreve frameworks ou famílias de sistemas relacionados.

2.4.2.2 Atributos de um padrão

Um padrão é descrito por vários atributos que podem variar de acordo com os critérios de descrição adotados. Porém, existem atributos essenciais que devem ser claramente identificáveis ao se ler um padrão [APPLETON, 2002] [SILVA, 2003]:

Nome: todo padrão deve ter um nome significativo. Pode ser uma única palavra ou frase curta que se refira ao padrão e ao conhecimento ou estrutura descritos por ele.

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

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

Forças: devem mostrar quais são os fatores que, frente ao contexto especificado e ao problema estabelecido, influenciam a solução proposta e como ela é implementada. Também pode apresentar limitações e restrições da solução, desvantagens em utilizar o padrão ou o porquê da solução não utilizar outras estratégias.

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

Padrões relacionados: são relacionamentos estáticos e dinâmicos entre este padrão e outros dentro da mesma linguagem de padrões ou sistema de padrões. Eles também têm um contexto inicial ou resultante que é compatível com o contexto inicial ou resultante de outros padrões. Tais padrões podem ser padrões predecessores cuja aplicação conduz a este padrão; padrões sucessores cuja aplicação segue deste padrão; padrões alternativos que descrevem uma solução diferente para o mesmo problema mas sob diferentes forças e restrições; e padrões co-dependentes que podem (ou devem) ser aplicados simultaneamente com este padrão.

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

2.4.3. Metodologia para a Engenharia de domínio

Nesta seção é apresentado um exemplo de metodologia existente para a Engenharia de domínio enfatizando principalmente a fase de Projeto de domínio.

2.4.3.1 FODA

FODA (“*Feature-Oriented Domain Analysis*”) é uma metodologia que busca a análise de domínio identificando as características relevantes, particularidades e diferenças de uma classe de sistemas [KYO, 1990]. A metodologia FODA é fundada em dois conceitos: abstração e refinamento. Abstração é o princípio usado para criar produtos de domínio de

aplicações específicas no domínio. Estes produtos genéricos de domínio abstraem funcionalidades das aplicações em um domínio. São desenvolvidas aplicações específicas no domínio como refinamentos dos produtos de domínio.

O processo na FODA é dividido em três fases [KYO, 1990]:

- *Análise de contexto*: O propósito da análise de contexto é definir o escopo de um domínio. São analisadas as relações entre o domínio e elementos externos (ambiente, requisitos, etc.), então as variações são avaliadas. Os resultados são documentados em um *modelo de contexto*.
- *Modelo de domínio*: De posse do escopo do domínio, a fase de modelagem do Domínio provê passos para analisar as semelhanças e diferenças entre as aplicações no domínio, então após essa análise são produzidos alguns modelos do domínio. A modelagem do domínio tem três atividades principais: *análise das características,* onde as características gerais que descrevem o contexto de aplicações de domínio são definidas; *análise das informações,* na qual são definidos e analisados o conhecimento do domínio e os requisitos de dados para implementar aplicações no domínio; e por fim a *análise operacional*, onde as características de comportamento das aplicações em um domínio são identificadas.
- *Modelagem arquitetural*: Esta fase provê uma solução de software para aplicações no domínio. É desenvolvido um *modelo arquitetural* que representa um desenho de alto nível para aplicações em um domínio,.

A fase de Projeto de domínio na metodologia FODA [KYO, 1990] é composto pela fase de construção do *modelo arquitetural*, pelo *detalhamento do modelo* e pela *construção dos componentes*. Tem-se como objetivo a construção de produtos reutilizáveis, através da construção de projetos em um determinado domínio, focados na identificação de processos simultâneos, módulos comuns ao domínio e na alocação das características, funções, e objetos definidos no modelo de domínio (Análise de domínio) para estes processos e módulos.

2.4.3.1.1 Descrição do *modelo arquitetural*

Um *modelo arquitetural* deve direcionar os problemas definidos no modelo de domínio de uma maneira que possa ser adaptado a futuras mudanças nos diversos problemas e tecnologias. Esta adaptação é alcançada através da separação em camadas da arquitetura onde:

- A definição da arquitetura é feita em vários níveis de abstração de forma que o reuso possa acontecer no nível apropriado para uma determinada aplicação;
- A definição das funções e objetos de domínio é feita separadamente das técnicas de implementação de forma que:
  - Decisões de implementação possam ser separadas da funcionalidade;
  - A reutilização dos módulos possa ser aumentada;
  - O impacto para o resto do sistema durante mudanças nas técnicas de implementação possam ser localizados.

Nesta técnica propõe-se que uma aplicação seja definida como uma hierarquia de módulos constituídos de funções e objetos de dados, e definidos através de um modelo de fluxo de dados, identificando os módulos comuns orientados ao domínio, buscando com isso aumentar a reusabilidade [KYO, 1990].

As decisões de implementação estão baseadas em várias técnicas de implementação, como comunicação e mecanismos de sincronização, métodos de programação de processo, sistemas de gerenciamento de banco de dados e linguagens de programação.

Uma arquitetura em camadas dos sistemas pode ser definida como mostrado na Figura 7 [KYO, 1990]. No topo, a *camada de arquitetura de domínio*, que é representada através de um modelo mostrando os processos simultâneos do domínio e interconexões entre eles. Este modelo é chamado de modelo de interações de processo que é representado usando a metodologia DARTS ("*Design approach for real-time systems*") [GOMMA, 1984]. A camada seguinte, a *camada de utilidades do domínio*, mostra o conjunto de funções e dados dentro dos módulos e interconexões entre eles, chamado de modelo de estrutura de quadros, representado usando notações de Estrutura de Quadros [YOURDON, 1978], seguindo a metodologia DARTS. Os módulos orientados ao domínio que são comuns entre as aplicações em um domínio também são identificados.

A *camada de utilidades comuns* contém módulos que podem ser usados por diferentes domínios. Módulos para comunicação inter-processos, sincronização e administração de dados pertencem a esta camada. Qualquer utilidade fornecida pelos sistemas operacionais e linguagens de programação pertencem à *camada de sistemas*.

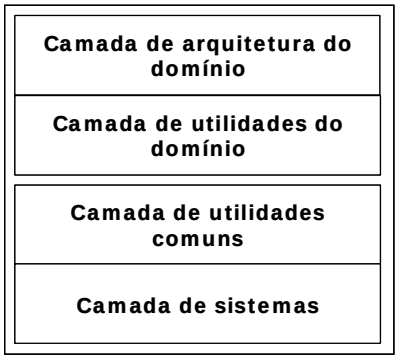

Figura 7. Camadas arquiteturais na FODA

A metodologia FODA focaliza-se nas duas camadas do topo, ou seja, no desenvolvimento de arquiteturas orientadas ao domínio, um projeto de alto nível onde o agrupamento de funções e objetos em módulos de software é o objetivo primário. São identificadas tarefas concorrentes, comunicações e sincronizações entre as tarefas. Cada tarefa é projetada como um programa seqüencial, construindo as funções específicas e dados usando o Projeto Estruturado. Nenhuma decisão sobre a implementação de comunicação e mecanismos de sincronização é tomada neste nível, eles devem ser feitos posteriormente para completar o projeto.

Os componentes de cada uma das camadas arquiteturais podem ser construídos com base em níveis de *modelos conceituais* definidos para os componentes. Por exemplo, no domínio de Sistemas de administração de janelas, um projeto em camadas pode ser desenvolvido como mostrado na Figura 8. A camada no nível mais baixo, *driver de dispositivo virtual*, provê um *modelo conceitual* em termos das operações em nível de *pixel* escondendo as peculiaridades de dispositivos particulares do resto do sistema. A próxima camada, acima do *driver de dispositivo virtual*, provê um modelo conceitual ao nível de "gráficos" onde tipos diferentes de linhas e formas estão definidas e as operações para criar, mover, e destruir as mesmas. A camada acima da camada de gráficos está definida com base no conceito de janelas e operações de janela. Uma janela tem uma forma, contém informações e tem atributos de exibição. As operações são as de criação de janelas, exibição de conteúdos, e os valores de seus atributos podem variar. A camada no topo contém tipos diferentes de janelas (tipicamente chamadas *widgets*) incluindo botões, barras de rolagem, menus, formulários, entre outros, os quais podem ser reunidos para criar janelas mais complexas.

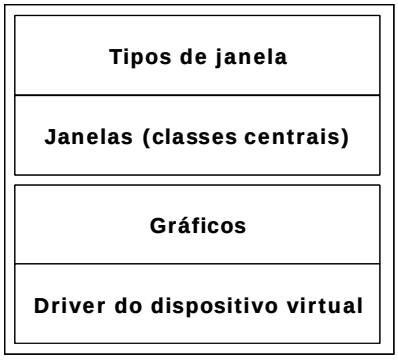

Figura 8. Estrutura de projeto de um sistema de administração de janelas

Nota-se pelo exemplo anterior que:

- Cada camada está definida com base em um *modelo conceitual*;
- Um *modelo conceitual* em níveis inferiores é mais comum a vários domínios que modelos em níveis mais altos (i.e., cada camada é uma especialização da camada abaixo).

Projetar um módulo baseado em modelos conceituais é importante, por que esta prática provê uma base para a definição de objetos e operações, e caso haja uma boa compreensão do modelo, fica mas fácil relacioná-lo a outros problemas.

Embora modelos de baixo nível sejam mais genéricos e reutilizáveis que os modelos de alto nível, o aumento de produtividade com reuso de modelos de alto nível é maior que o de modelos de baixo nível. Então, na modelagem arquitetural varias camadas devem ser definidas para aumentar a produtividade e reutilização. Porém, o desempenho poderia ser degradado com muitas camadas e deve-se levar em consideração todos estes fatores.

2.5. Considerações Finais

Neste capítulo foram apresentados conceitos sobre o projeto de software segundo as características do seu principal produto, a arquitetura do software, detalhando-a e exemplificando-a através dos estilos arquiteturais.

Foi apresentada uma visão geral da Engenharia de domínio, enfatizando os métodos relevantes para a fase de Projeto de domínio, o uso de framework e padrões de software, finalizando com a uma análise de uma metodologia para a Engenharia de Domínio, a FODA.

3. O Projeto de sistemas multiagente

3.1. Conceito de agente

Existe uma série de sistemas que se baseiam em uma tecnologia chamada de agente. Existem diversas definições para agentes. Cada autor tem sua própria definição e concepção do que é um agente. Geralmente, estas definições estão associadas a diferentes pontos de vista e dependem muito das características e funcionalidades apresentadas pelo agente em questão.

Um agente caracteriza-se também como uma entidade que percebe seu ambiente através de sensores e atua neste ambiente através de executores. Por exemplo, nos agentes humanos, os olhos e os ouvidos são sensores e as mãos e a boca são executores. A Figura 9 mostra um agente genérico [RUSSELL,1995].

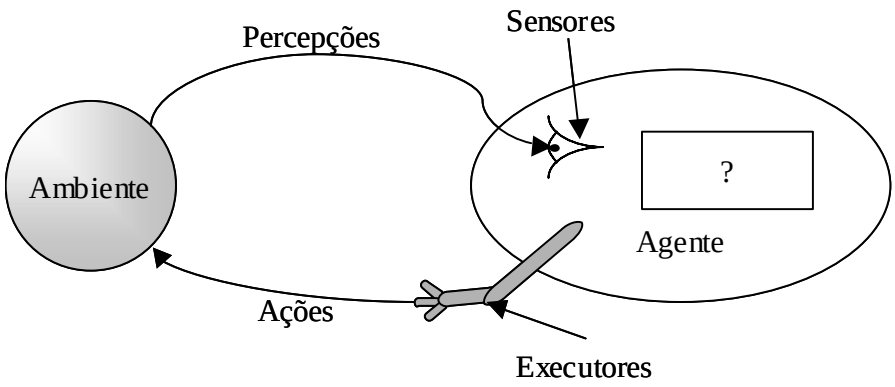

Figura 9. Agente genérico

Um agente difere-se de um objeto por ter um comportamento ativo, tendo autonomia no seu comportamento, um objeto é uma entidade passiva que reage a partir de estímulos recebidos. Abaixo estão listadas as diferenças mais significativas entre agentes e objetos[FERBER, 1999]:

- Agentes diferem de objetos por serem autônomos. Objetos têm autonomia somente sobre seu estado interno, mas não exibem controle sobre seu comportamento. Dessa forma, objetos têm controle sobre como as coisas são feitas, porém nenhum poder de decidir se determinada solicitação vai ou não ser atendida.

- Agentes apresentam um comportamento autônomo flexível (reativo, pró-ativo, social). Os modelos existentes no paradigma orientado a objetos não especificam nenhuma maneira de representar estes tipos de comportamento.
- Quando inseridos em um sistema, cada agente possui, necessariamente, sua própria linha de execução. Assim sendo, cada um dos agentes de uma sociedade pode ser visto como sendo um processo computacional e não apenas uma estrutura composta por métodos e estado interno.

Algumas destas diferenças não são aplicáveis quando se consideram objetos distribuídos.

Os agentes podem ser identificados por possuírem as seguintes características principais [GRENN, 2002] [HERMANS, 2001] [WOOLDRIDGE, 1995]:

Autonomia: um agente pode funcionar sem intervenção humana e executar ações pela sua própria iniciativa segundo o conhecimento que ele possui do seu ambiente e sua racionalidade;

Habilidade social: um agente interage com outros agentes através de protocolos de interação sofisticados, como cooperação, competição e negociação;

Reatividade: um agente deve ser capaz de perceber mudanças em seu ambiente e atuar de acordo com estas mudanças;

Pró-atividade: os agentes não respondem apenas ao ambiente, mas perseguem um objetivo, ou seja, buscam alcançar uma meta;

Capacidade de aprendizagem: um agente é capaz de aprender a partir das ações realizadas, considerando sucessos e fracassos, de forma a melhorar seu desempenho.

3.2. Sistemas multiagente

Um Sistema multiagente (SMA) pode ser definido como uma sociedade de agentes que interagem entre si ou com outros sistemas de software para a resolução de um problema comum, que está além das capacidades individuais dos agentes. O surgimento e crescimento desta abordagem deve-se principalmente ao fato de que, em relação a um sistema centralizado, um SMA inclui habilidades para [GRENN, 2002]:

- Resolver problemas que, por serem muito complexos e/ou por haver limitações de recursos, não poderiam ser solucionados por um único agente;
- Aumentar a velocidade de processamento de um sistema (através do processamento paralelo);
- Prover maior flexibilidade aos sistemas, pois agentes com diferentes habilidades cooperam entre si para resolver problemas de forma dinâmica;
- Aumentar a confiabilidade, permitindo se recuperar de uma falha, sem alterar seu desempenho;
- Oferecer clareza e simplicidade conceitual do projeto.

Para que vários agentes autônomos pertencentes a um sistema multiagente possam cooperar, e dessa forma, atingirem seus objetivos é necessário que eles possam se comunicar, coordenar suas atividades e negociar soluções para conflitos que possam vir a acontecer. A coordenação é requerida para determinar a estrutura organizacional entre um grupo de agentes e para alocação de tarefas e recursos.

3.2.1. A fase de Projeto

Como já citado no capítulo 2, o projeto é uma fase de particular importância na construção de um sistema. No Projeto de sistemas multiagente (SMA) não é diferente, é nesta fase onde é definida a arquitetura do sistema multiagente e feito um estudo detalhado de cada agente.

A fase de projeto de um SMA abrange duas atividades principais:

- O projeto arquitetural do SMA para a construção da arquitetura de software, que estabelece os mecanismos de comunicação entre os agentes da sociedade;
- O projeto detalhado de cada agente da sociedade, onde cada agente é analisado de forma isolada e detalhada em relação ao seu conhecimento e comportamento.

As arquiteturas dos sistemas multiagente podem ser classificadas de acordo com as necessidades da aplicação, dos usuários e do grau de sofisticação ou nível de inteligência dos agentes. De acordo com a sua complexidade [KNAPIK, 1998], a arquitetura de um sistema multiagente pode ser classificada em três grupos:

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

Já de acordo com os mecanismos de cooperação e coordenação utilizados na sociedade, as arquiteturas dos sistemas multiagente podem ser classificadas em:

- *Arquiteturas cooperativas*: onde há uma centralização de esforços na cooperação entre os agentes da sociedade, como nas arquiteturas de quadro-negro, troca de mensagens e federativa. Na arquitetura de quadro-negro os agentes têm acesso a informações e conhecimento de maneira facilitada sem a necessidade de se comunicarem diretamente. Na arquitetura *troca de mensagens* os agentes cooperam se comunicando, através de mensagens assíncronas com o objetivo de facilitar e agilizar a comunicação. Na arquitetura Federativa, se tem um sistema de troca de mensagens onde o número de agentes é muito grande e os agentes são divididos em grupos ou federações segundo um critério de agrupamento escolhido
- *Arquiteturas coordenativas*: onde há uma centralização de esforços na coordenação entre os agentes da sociedade, como nas arquiteturas mestre - escravo e de mercado. Na arquitetura do tipo mestre-escravo existem duas classes de agentes: os gerentes (mestres) e os trabalhadores (escravos). Os agentes trabalhadores são coordenados por um gerente que distribui as tarefas entre estes e espera o resultado. Na arquitetura de mercado, todos os agentes estão em um mesmo nível e sabem as tarefas que cada agente é capaz de desempenhar. Os agentes colaboram para executar tarefas complexas que se estendem além de suas capacidades individuais.

Em relação ao projeto detalhado abordam-se também aspectos estruturais relativos aos agentes, onde é feita uma classificação baseada na forma de construção dos agentes:

- *Arquiteturas deliberativas*. As arquiteturas deliberativas seguem a abordagem clássica da Inteligência Artificial, onde os agentes contêm um modelo simbólico do mundo, explicitamente representado, e cujas decisões (ações) são tomadas via raciocínio lógico, baseado em casamento de padrões e manipulações simbólicas. Esta arquitetura é utilizada nos agentes baseados em metas e agentes baseados na utilidade da classificação de Russel [RUSSEL, 1995].
- *Arquiteturas reativas*. As arquiteturas reativas são aquelas que não incluem nenhum tipo de modelo central e simbólico do mundo e não utilizam raciocínio complexo e simbólico. Baseiam-se na proposta que um agente pode desenvolver inteligência a partir de interações com seu ambiente, não necessitando de um modelo pré-estabelecido. Esta arquitetura é utilizada nos agentes reflexivos e nos agentes reflexivos com estado da classificação de Russel.
- *Arquiteturas híbridas*: A arquitetura híbrida mistura componentes das arquiteturas deliberativas e reativas com o objetivo de torná-la mais adequada e funcional para a construção de agentes. Ela propõe um subsistema deliberativo que planeja e toma decisões da maneira proposta pela Inteligência Artificial Simbólica e um reativo capaz de reagir a eventos que ocorrem no ambiente sem se ocupar de raciocínios complexos.

3.3. AUML

A AUML (Linguagem de modelagem unificada para agentes) [ODELL, 2000] surgiu tendo em vista que a UML (Linguagem de modelagem unificada) possui limitações para modelagem de agentes e sistemas baseados em agentes. Dado que a UML tem tido uma larga aceitação na modelagem de software orientado a objetos, nada mais coerente que tentar expandir a UML para a utilização de agentes, considerando que um agente pode ser tratado como a extensão de um objeto. Logo, essa extensão levou em consideração as exigências e as distinções da tecnologia de agentes, surgindo então a AUML. Odell, Parunak e Bauder [ODELL, 2000] abordam um subconjunto da AUML, onde são especificados Protocolos de interações de agentes (AIP) e outras noções baseadas em agentes.

Protocolos de interação foram escolhidos porque eles são complexos o bastante para ilustrar o uso de não trivial da AUML e são comumente usados para fazer a AUML útil para outras pesquisas. Protocolos de interação de agentes são um bom exemplo de padrões de software que são idéias úteis em um contexto prático. Uma especificação de um AIP provê um exemplo ou analogia que nós poderíamos usar para resolver problemas em análise de sistema e projeto.

De forma geral, as técnicas para representação dos AIP's foram divididas em três níveis de apresentação [ODELL, 2000]:

Nível 1: representação dos protocolos de interações globais do sistema.

Nível 2: representação das interações entre os agentes.

Nível 3: representação do processamento interno do agente.

O nível 1 especifica uma solução que pode ser reutilizável para organizar os sistemas. Os protocolos de interação de agentes (AIP's) oferecem soluções reutilizáveis que podem ser aplicadas em vários tipos de seqüências de mensagens que são encontradas entre agentes. Na AUML há uma técnica que expressa soluções de protocolos para o reuso: os *pacotes* [ODELL, 2000].

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

A Figura 10 mostra dois pacotes [ODELL, 2000]. O pacote *comprador* expressa um protocolo simples entre um corretor e um varejista. O varejista recebe uma proposta de compra do corretor que responde com uma contraproposta. Para certos produtos, o varejista pode solicitar, previamente, a um atacadista as disponibilidades e os custos dos produtos solicitados pelo comprador. Baseado nas informações recebidas do atacadista, o varejista pode fornecer uma proposta mais precisa. Toda essa operação poderia ser colocada dentro do pacote *comprador*, entretanto, há situações em que não há necessidade de protocolos adicionais envolvendo o atacadista. Portanto, dois pacotes podem ser definidos: o pacote *comprador* e o pacote *fornecedor*. Quando um cenário particular requer um protocolo com o atacadista, o pacote *fornecedor* pode ficar aninhado ao pacote *comprador*, caso contrário ele pode ficar como um pacote distinto.

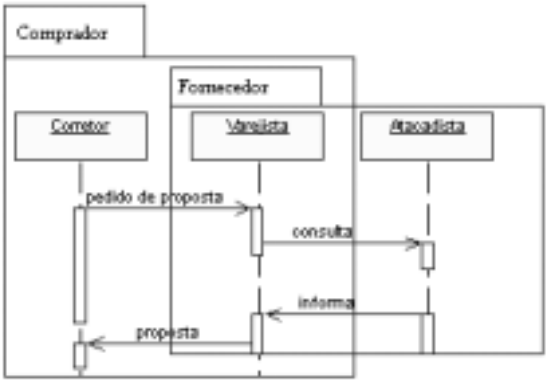

Figura 10. Utilização de pacotes para expressar protocolos aninhados

O nível 2 utiliza diagramas de seqüência, colaboração, atividade e estado que mostram como os agentes interagem entre si, através da troca de mensagens.

O layout gráfico dos diagramas de seqüência (Figura 11) e colaboração (Figura 12) mostra a seqüência cronológica das comunicações entre os agentes [SILVA, 2003].

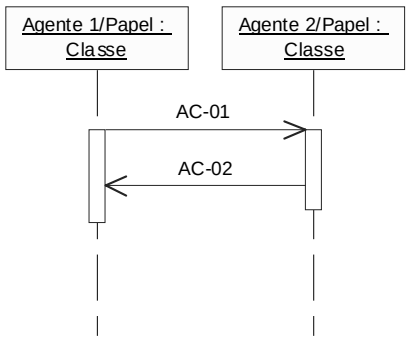

Figura 11. Exemplo de um diagrama de seqüência entre agentes da AUML

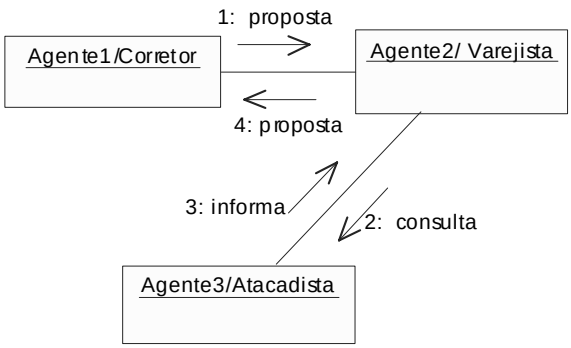

Figura 12. Exemplo de um diagrama de colaboração entre os agentes da AUML

O diagrama de atividades e o diagrama de estados mostram o fluxo do processamento da informação em um sistema multiagente. Os diagramas de atividades (Figura 13) fornecem uma visão baseada no processamento das informações, onde os agentes são representados pelas divisões verticais, as *operações* são representadas por retângulos de bordas arredondadas e os *eventos* por setas.

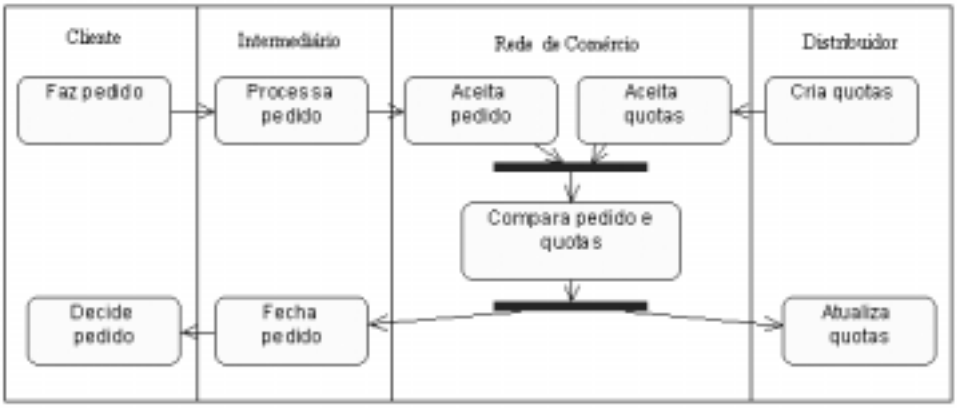

Figura 13. Exemplo de um diagrama de atividades da AUML

Um diagrama de estado (Figura 14) é um gráfico que representa uma máquina de estados, onde estados são simbolizados por retângulos arredondados e as transições por setas que interconectam os estados.

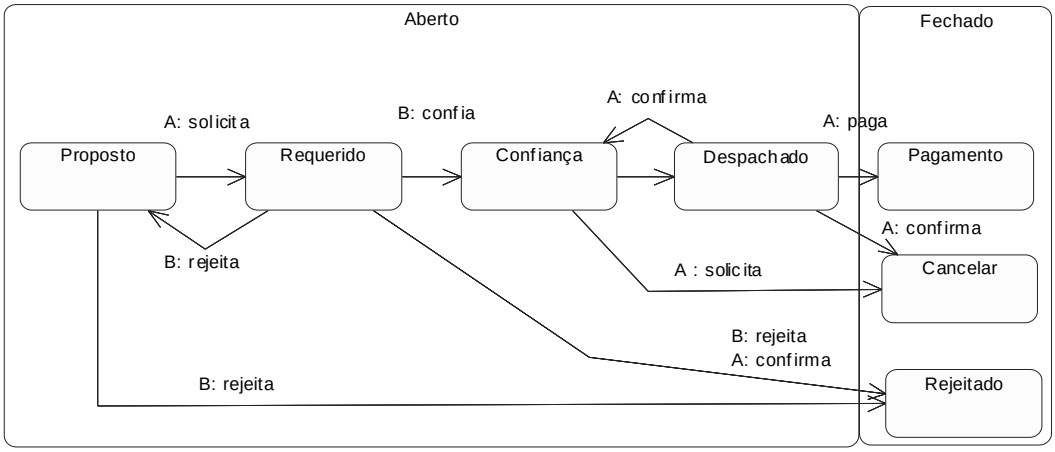

Figura 14. Exemplo de um diagrama de estados da AUML

No nível 3 é abordado o comportamento interno dos agentes. Para especificar esse comportamento utilizam-se os diagramas de estado e de atividades do nível 2. Nestes diagramas os estados e atividades de cada agente são mostrados de forma detalhada, com um

diagrama para cada agente. Na Figura 15 temos um exemplo de diagrama de atividades no nível 3.

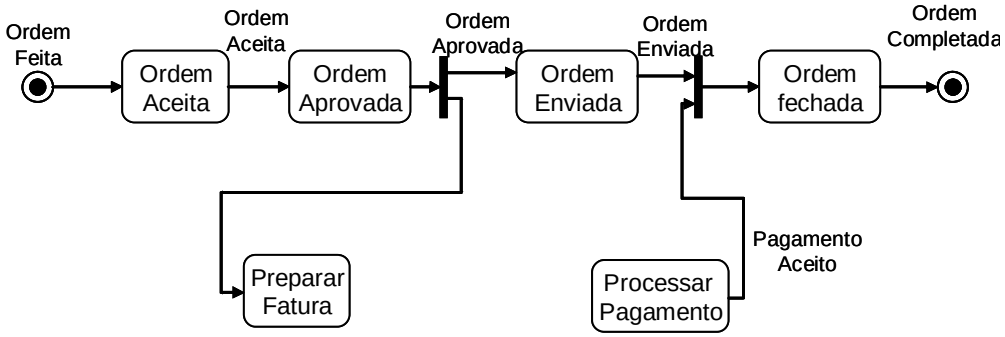

Figura 15. Exemplo de diagrama de atividade no nível 3

3.4. Análise das técnicas de projeto para o desenvolvimento de aplicações multiagente

Nesta seção é apresentada uma análise de algumas técnicas existentes para a fase de projeto de sistemas multiagente.

3.4.1. A técnica de projeto na metodologia GAIA

A metodologia GAIA [WOOLDRIDGE, 2000] busca a construção de sociedades de agentes que cooperam para atingir as metas do sistema, definindo o que é necessário que cada agente da sociedade faça para que se alcance este objetivo.

Na fase de análise são gerados dois modelos: o *modelo de papéis*, o qual identifica os papéis fundamentais no sistema, ou seja, descrição abstrata da função esperada de uma entidade, e o *modelo de interações*, o qual define as relações e protocolos existentes entre os papéis. Na fase de projeto são gerados três modelos: o *modelo de agentes*, o *modelo de serviços* e o *modelo social* (Figura 16).

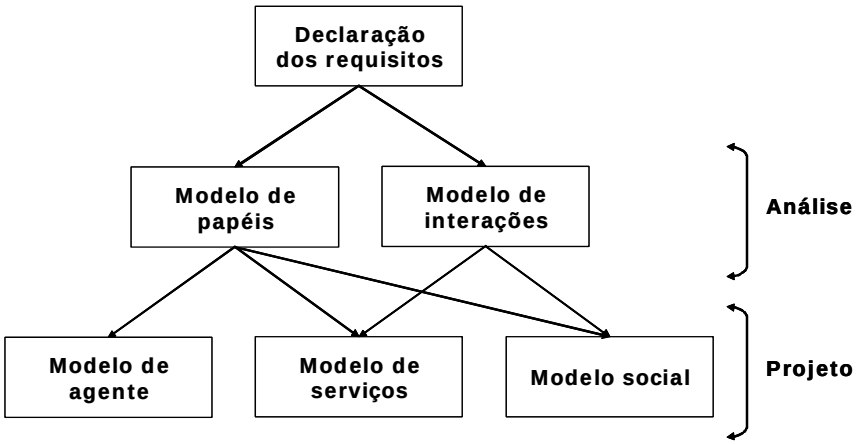

Figura 16. Análise e projeto na metodologia GAIA

A partir do *modelo de papéis* é construído o *modelo de agentes*, onde são identificados os tipos de agentes que comporão o sistema e suas instâncias. Este modelo é definido usando uma simples árvore de tipos de agente, na qual os nós folha correspondem aos papéis de cada agente e os outros nós correspondem aos tipos de agente. Se um tipo de agente t_1 tiver filhos t_2 e t_3, então isto significa que t_1 está composto dos papéis que compõem t_2 e t_3. Neste modelo é também apresenta o número de instâncias de um determinado agente, *n* representa que existiram exatamente *n* instâncias, *m..n* entre *m* e *n* instâncias, * zero ou mais instâncias e + uma ou mais instâncias. A Figura 17 mostra um *modelo de agentes* onde são apresentados os agentes e seus papéis em um sistema de *Administração do processo empresarial*. O agente *AtendimentoCliente* desempenha os papéis de *ControladorCliente* e *ReceptorCliente*; este agente possuirá apenas uma instância como apresentado no modelo.

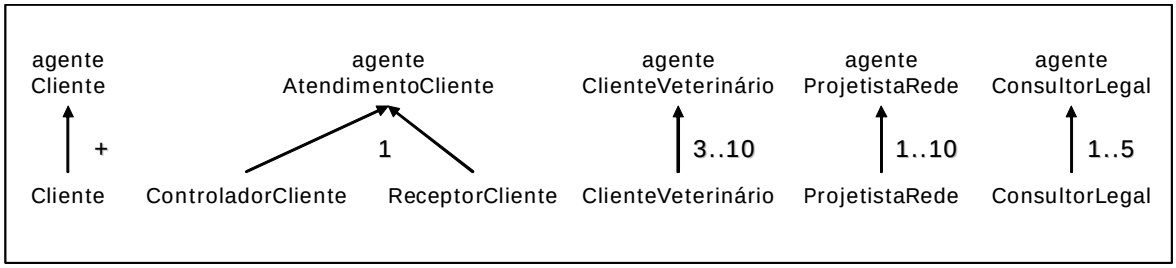

Figura 17. Exemplo de modelo de agentes da GAIA

A partir do *modelo de papéis* e do *modelo de interações* é construído o *modelo de serviços* onde são identificados os principais serviços necessários para a realização do papel desempenhado pelo agente. Este modelo busca identificar os serviços associados aos papéis de cada agente, e especificar as principais propriedades destes serviços. Um serviço se refere a uma função desempenhada por um agente. Toda atividade identificada na fase de análise corresponderá a um serviço, entretanto nem todo serviço corresponderá a uma atividade.

Para cada serviço que pode ser executado por um agente, é necessário documentar suas propriedades. Especificamente, devem ser identificadas as entradas, saídas, pré-condições, e pós-condições de cada serviço. As entradas e saídas são derivadas do modelo de interações, criado na fase de análise. Pré e pós-condições representam restrições dos serviços, derivadas das propriedades de segurança de um papel. Note que, por definição, cada papel será associado com pelo menos um serviço [WOOLDRIDGE, 2000]. A Figura 18 mostra os serviços apresentados juntamente com suas propriedades, como, por exemplo, o serviço

verificar satisfação do cliente, cuja entrada é *taxa de crédito*, saída *decisão de continuação*, pré-condição *decisão de continuação = 0* e pós-condição *decisão de continuação <> 0*.

Serviço	Entradas	Saídas	Pré-condição	Pós-condição
obter requisitos de cliente	detalhes do cliente	requisitos do cliente	verdadeiro	verdadeiro
serviço veterinário ao Cliente	detalhes do cliente	taxa de crédito	cliente veterinário disponível	taxa de crédito <> 0
verificar satisfação do cliente	taxa de crédito	decisão de continuação	decisão de continuação = 0	decisão de continuação <> 0

Figura 18. Exemplo de modelo de serviços da GAIA

A partir do *modelo de papéis* é construído o *modelo social* onde são representadas as comunicações entre os diferentes agentes [WOOLDRIDGE, 2000]. Este modelo não define que mensagens serão enviadas ou quando serão enviadas, apenas indica as comunicações existentes. Em particular, o propósito de um *modelo social* é identificar qualquer possível gargalo de comunicação que possa causar problemas.

Um *modelo social* é representado em um gráfico, onde os nós correspondem aos tipos de agente e os arcos correspondem à comunicações. Na Figura 19 é apresentado um exemplo de modelo social, onde o agente *Cliente* interage com o agente *AtendimentoCliente*, que por sua vez interage com os outros agentes *ClienteVeterinário*, *ProjetistaRede* e *ConsultorLegal*.

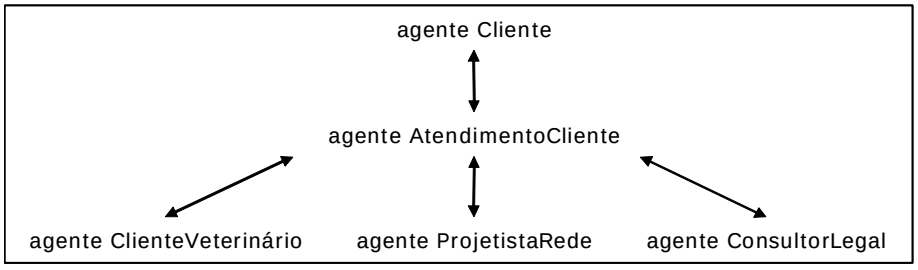

Figura 19. Exemplo de modelo social da GAIA

3.4.2. A técnica de projeto na metodologia MaSE

A metodologia MaSE ("Multiagent Systems Engineering") [DILEO, 2002] guia o desenvolvedor através de todo o ciclo de vida, desde

a especificação de requisitos até um sistema multiagente implementado. A MaSE define técnicas a serem aplicadas nas fases de análise e projeto e usa ontologias para descrever a informação de um domínio, permitindo o uso dos termos da ontologia durante a fase de análise de requisitos (Figura 20).

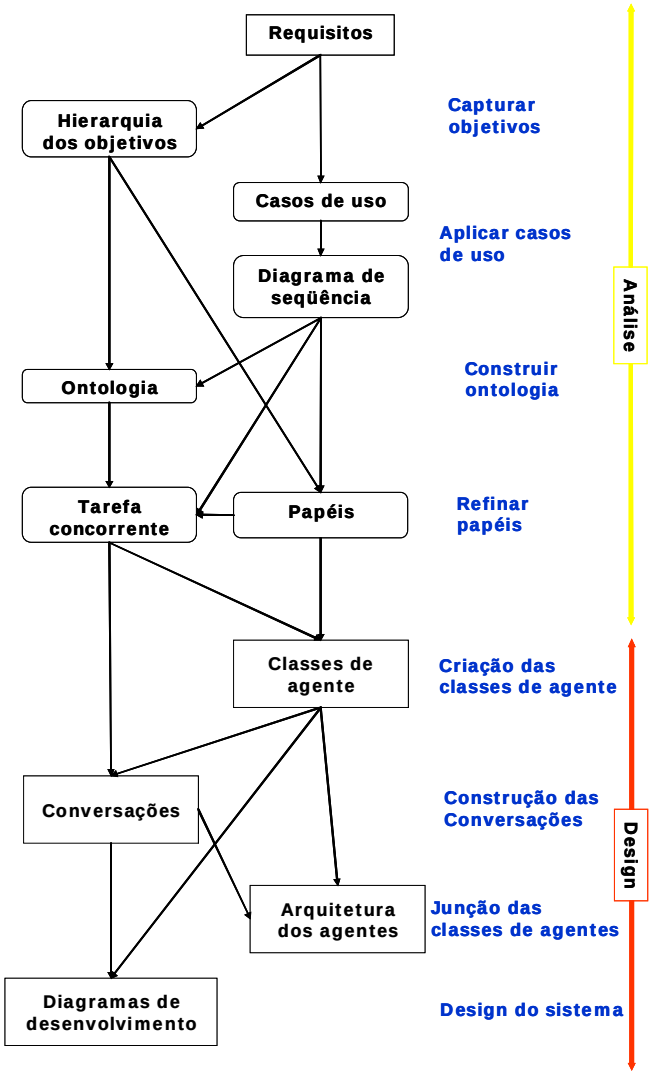

Figura 20. As fases da metodologia MaSE

A fase de projeto da MaSE tem como primeiro passo a criação das classes de agentes, onde os papéis são atribuídos a classes de agente específicas, originando o *diagrama de classes de agentes*. Este diagrama mostra as classes no sistema, os papéis desempenhados pelas classes de agentes e as conversações entre classes. Na Figura 21 temos um *diagrama de classes de agente* em um sistema de *gerenciamento de arquivos* onde é apresentado como exemplo o agente *MonitorArquivos* desempenhando os papéis *DetectorDelecaoArq* e *DetectorModificacaoArq* e tendo a conversação *Violação* com o agente *DetectorNotificação*.

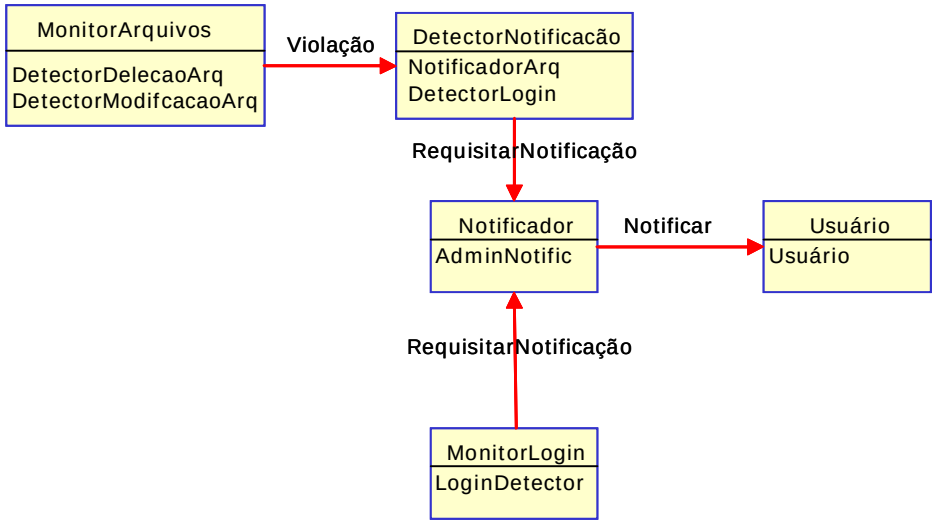

Figura 21. Exemplo de diagrama de classes de agentes da MaSE

Os detalhes das conversações são definidos na *construção das conversações*. Cada conversação, que representa as interações entre os agentes, tem dois diagramas: um para o iniciador e um para o respondedor da conversação. As conversações das quais uma classe de agente participa são derivadas das comunicações dos papéis que os agentes desempenham. Na Figura 22 temos a representação das conversações entre dois agentes de uma forma mais detalhada, iniciando com um *request()* que passa por um processo de decisão no estado *Wait1* e que pode seguir dois caminhos o *agree()* ou *failure()*.

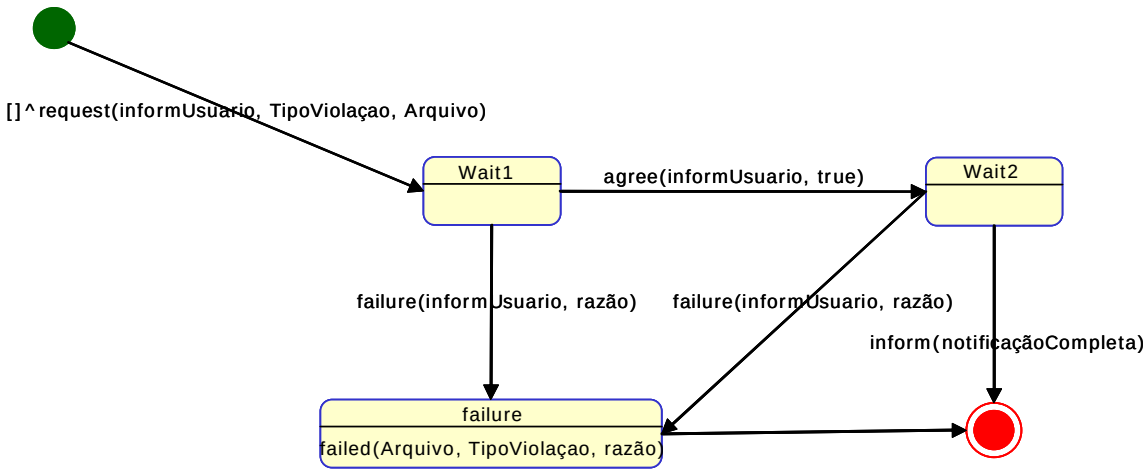

Figura 22. Exemplo de diagrama de conversação da MaSE

A atividade de *junção de classes de agente* define os componentes da arquitetura e permitindo a decomposição lógica dos agentes. O passo final cria um *diagrama de desenvolvimento* para mostrar a quantidade e local de cada tipo de agente no sistema. Na Figura 23 temos um representado um *diagrama de desenvolvimento* onde é definido que

existiria um módulo *Controlador* contendo instâncias do agente *Notificador*, sendo auxiliados por várias instâncias doa agentes *MonitorLogin* e *MonitorArquivo* localizadas fora deste módulo.

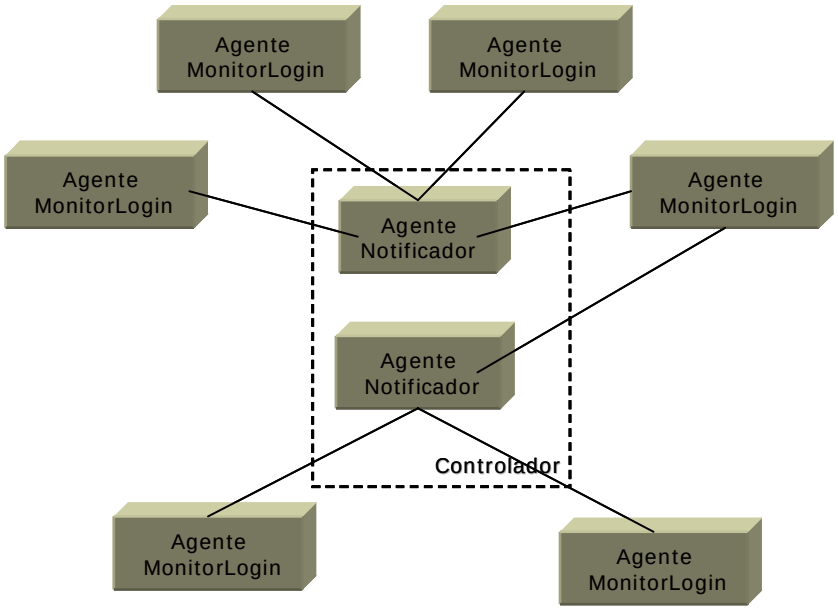

Figura 23. Exemplo de diagrama de desenvolvimento da MaSE

3.4.3. A técnica de projeto na metodologia MADS

MADS [SODRÉ, 2002] é uma metodologia que aborda as fases de análise e projeto global de aplicações de software baseada em agentes. A MADS consiste de um conjunto integrado de atividades e produtos gerados.

Na fase de projeto na metodologia MADS (Metodologia baseada em agentes para o desenvolvimento de software), os papéis identificados na fase de análise são reestruturados em abstrações agentes para a construção do *diagrama de agentes*. Em seguida, as atividades realizadas pelo agente são representadas no *diagrama de atividades* e por último, a arquitetura global da aplicação multiagente é representada no *diagrama de arquitetura* (Figura 24).

As atividades desta fase são os seguintes: *identificar os tipos de agentes, detalhar as atividades* e *montar a arquitetura global*.

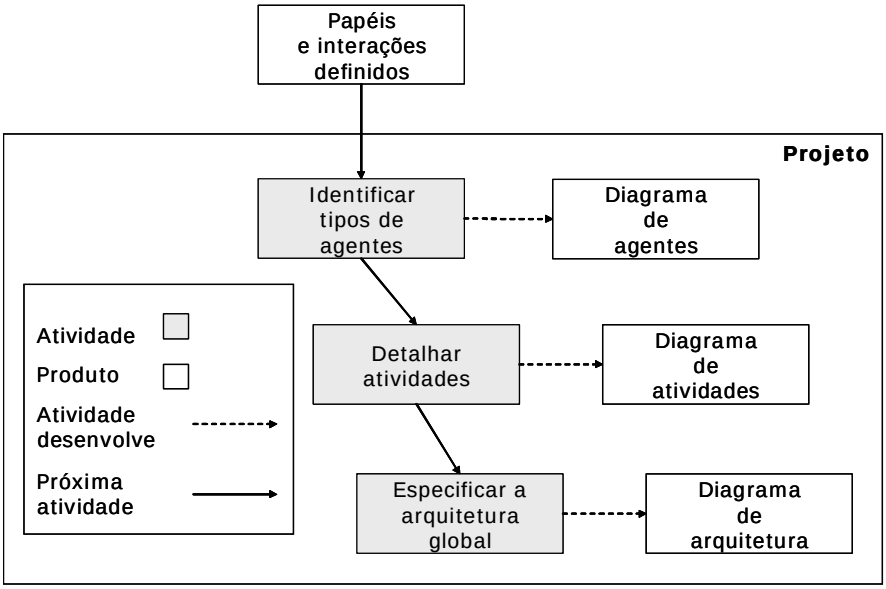

Figura 24. A fase de projeto na metodologia MADS

Na atividade de *identificar os tipos de agentes* são escolhidos os agentes que desempenharão cada papel, um agente pode executar um ou mais papéis num mesmo contexto, é construído nesta fase o *diagrama de agentes*. Na Figura 25 temos um exemplo de *diagrama de agentes* representando um agente de *monitoramento*. O agente de *monitoramento* desempenha o papel identificado na fase de análise *Monitor* utilizando-se do recurso da *Web*.

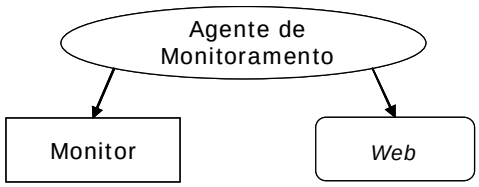

Figura 25. Exemplo de diagrama de agentes da MADS

Na atividade de *detalhar as atividades* é feita uma descrição mais minuciosa de cada atividade encontrada na fase de análise. Este detalhamento é representado graficamente pelo *diagrama de atividades*. Na Figura 26 temos um exemplo de *diagrama de atividades* para o *descobrimento de informação* em um processo de monitoramento da *Web*. Neste diagrama é apresentado as atividades dos agentes *Monitor* e *Descobridor*, os quais realizam as atvidades de *detectar alteração no estado dos sites, solicitar nova pesquisa na Web, extrair documentos* e *enviar documentos descobertos.*

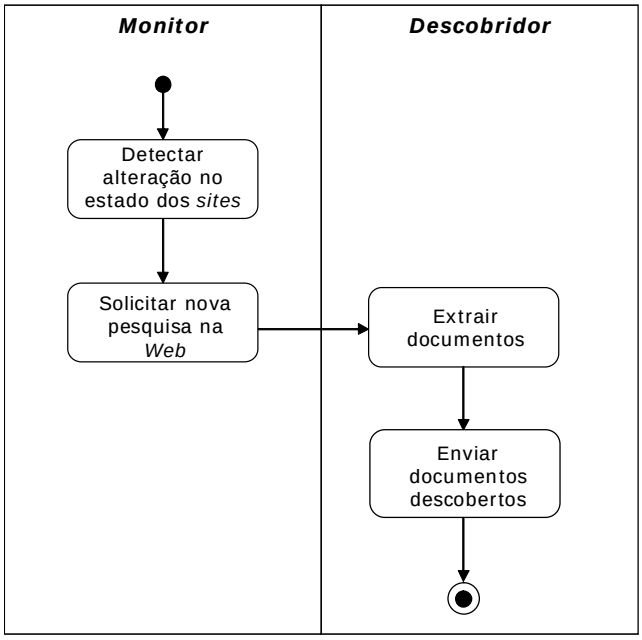

Figura 26. Exemplo de diagrama de atividades da MADS

A ultima atividade é *montar a arquitetura global*. Nesta atividade os agentes, seus os respectivos papéis e entidades externas são agrupados na construção do *diagrama de arquitetura*. Na Figura 27 temos um exemplo de diagrama apresentando a *arquitetura global* de um sistema de monitoramento de mudanças na *Web*.

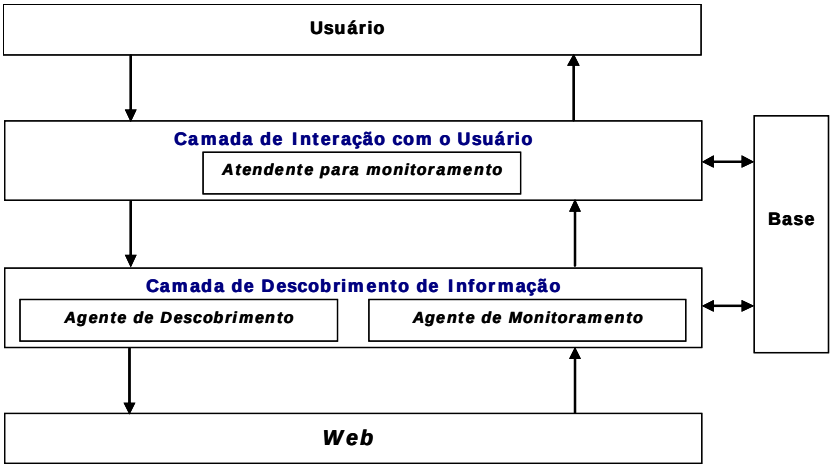

Figura 27. Exemplo de diagrama de arquitetura

3.4.4. A técnica de projeto na metodologia Tropos

A Tropos [CASTRO, 2001] [MYLOPOULOS, 2000] utiliza conceitos da estrutura i* [YU, 1995], tais como ator, agente, posição e papel, assim como dependências sociais entre atores, incluindo objetivos, objetivos auxiliares, tarefas e recursos.

A estrutura i* foi desenvolvida para modelagem e compreensão sobre os ambientes organizacionais e suas organizações. A estrutura i* consiste de dois modelos principais, o *modelo de dependência estratégica*, que é usado para descrever as relações de dependências entre vários atores em um contexto organizacional e o *modelo de razão estratégica*, que é usado para descrever os interesses dos clientes e como eles poderiam se relacionar.

A estrutura i* constrói uma aproximação da representação do conhecimento das informações de desenvolvimento do sistema. O conceito central de i* é da intencionalidade do ator, onde os atores têm propriedades intencionais como objetivos, crenças, habilidades e compromissos. Os atores têm objetivos a atingir através da execução de tarefas e com o auxílio de recursos.

O projeto na metodologia Tropos é composto por duas fases: *o projeto arquitetural* e o *projeto detalhado*.

O projeto arquitetural consiste de três atividades:

- Refinamento do diagrama de atores;
- Identificação das capacidades;
- Atribuição das capacidades aos agentes.

Na atividade 1 do projeto arquitetural é feito um *refinamento do diagrama de atores* que foi construído na fase de análise de requisitos. Este diagrama é estendido segundo padrões de projeto, os quais provêem soluções para comunicação entre agentes heterogêneos a requisitos não funcionais. A Figura 28 mostra o diagrama de atores estendido com a utilização do padrão *Info broker*. O *Administrador Interface de Usuário* e o *Administrador Interface de Fonte* são responsáveis por interfacear o sistema com os atores externos *Cidadão* e *Museu* respectivamente.

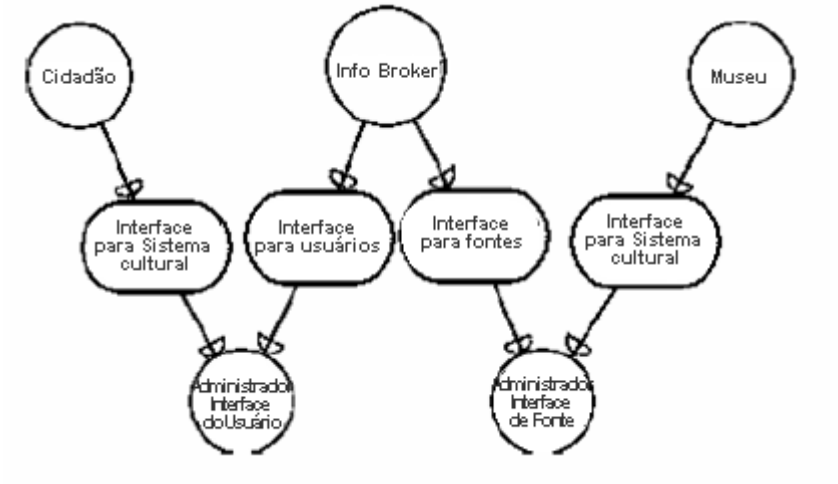

Figura 28. Diagrama de atores estendido, *Info broker*, na Tropos

A atividade 2 realiza a *identificação das capacidades*, consiste em capturar as capacidades dos atores analisando as tarefas que os atores e sub-atores irão executar em busca dos requisitos funcionais. Uma capacidade é um conjunto de eventos, planos e crenças necessários para o alcance dos objetivos e tarefas dos atores. A Figura 29 mostra um exemplo para analise do ator *Info broker*, direcionando-se ao objetivo de busca de informações por área. O *Info broker* é decomposto em três sub-atores: o *Classificador Área*, responsável pela classificação da informação provida pelo usuário, isto depende do *Administrador Interface de Usuário* para a interação com o usuário; o *Info searcher* depende do *Classificador Área* para obter informações sobre a área temática que o usuário está interessado e depende do *Administrador Interface de Fonte* para interagir com as fontes (Museus); e o *Sintetizador de Resultados* que depende do *Info searcher* para a tratar consultas à informação requisitadas e depende do *Museu* para ter os resultados da consulta. Cada relação de dependência na Figura 29 pode originar uma ou mais capacidades iniciadas por eventos externos. A Tabela 2 lista as capacidades dadas aos atores e identificadas pelo diagrama da Figura 29.

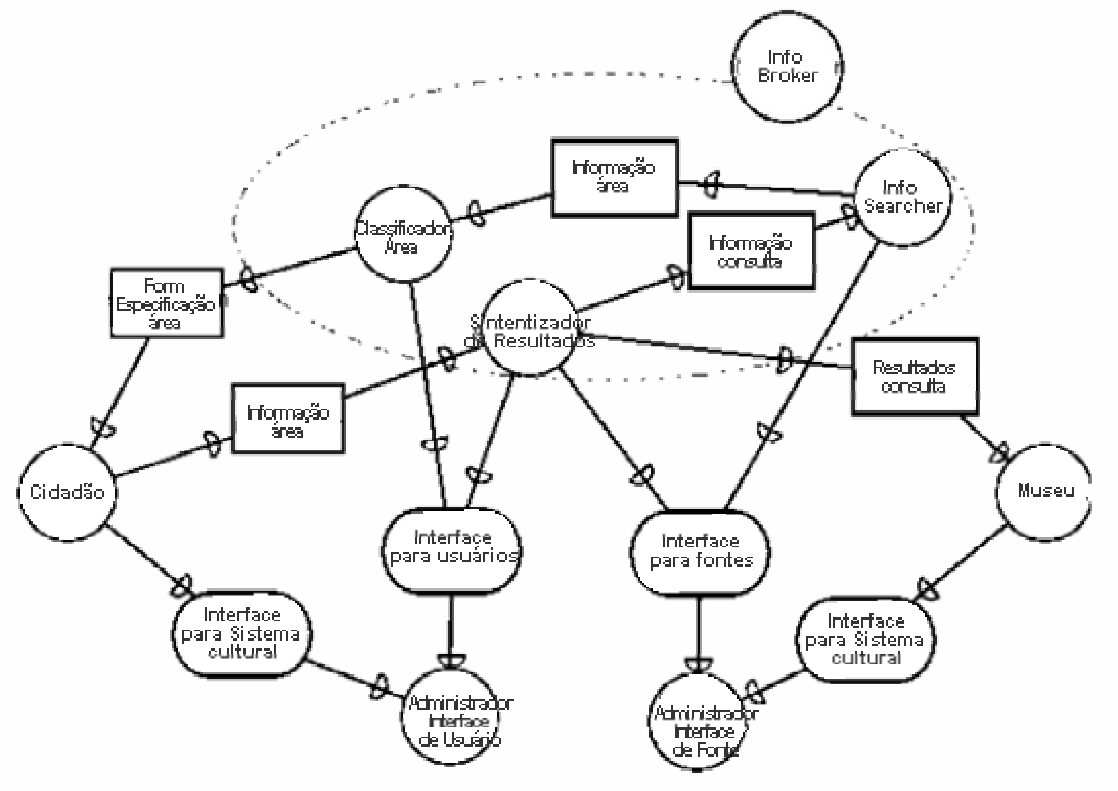

Figura 29. Diagrama de atores para análise de capacidades na Tropos

Nome do ator	N	Capacidade
Classificador Área	1 2 3 ...	Pegar a especificação da área Classificar a área Prover a informação da área
Info searcher	5 6 ...	Pegar informações da área Achar Fontes de Informação
Sintetizador de Resultados	10 11 12	Pegar informações da consulta Pegar resultados da consulta Prover resultados da consulta

Tabela 2. Capacidade dos atores

A última atividade consiste em fazer a *atribuição das capacidades aos agentes*, ou seja, definir um grupo de tipos de agentes e atribuir para cada agente uma ou mais capacidades. A Tabela 3 reporta aos agentes as respectivas capacidades listadas na Tabela 2.

Agentes	Capacidades
Tratador Consulta	1, 3, 4, 5, 7, 8, 9
Classificador	2, 4
Searcher	6, 4
Wrapper	14, 4

Tabela 3. Tipos de agentes e suas capacidades

No projeto detalhado, especificam-se as capacidades e interações dos agentes. Isto equivale a modelar os eventos internos e externos que disparam planos e crenças envolvidos no raciocínio do agente. São definidos então os *diagramas de capacidade, de planos* e *de interação dos agentes* [MYLOPOULOS, 2000]. Estes diagramas são um subgrupo dos diagramas da AUML definidos em [ODELL, 2000]. O *diagrama de capacidades* é feito utilizando o diagrama de atividades da AUML (Figura 13), através deles são modeladas as capacidades, do ponto de vista de um ator específico. Eventos externos definem o estado inicial no *diagrama de capacidade*; no diagrama de atividades, são modelados os planos, arcos de transição, eventos e crenças como objetos.

A Figura 30 mostra o *diagrama de capacidades* referente a busca dos resultados da consulta efetuada pelo agente *Interfaceador Usuário*. As formas ovais representam planos, os arcos representam os eventos internos e externos.

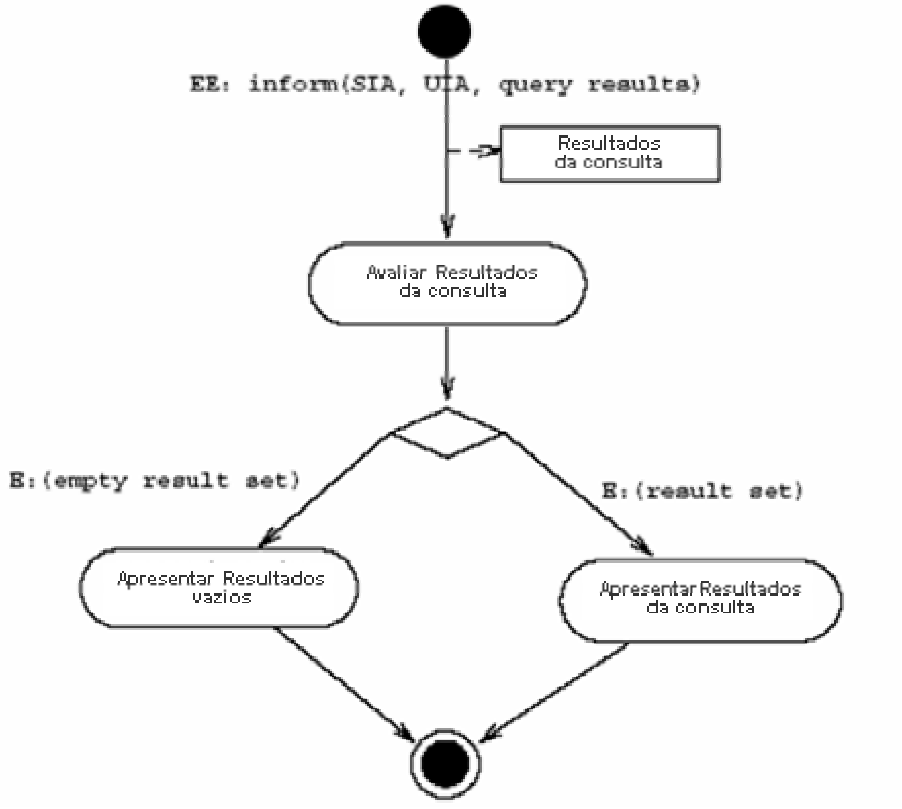

Figura 30. Diagrama de capacidade usando o diagrama de atividades da AUML.

No *diagrama de planos*, cada nó de plano do *diagrama de capacidade* é especificado pelo diagrama de ação da AUML. Na Figura 31 temos um exemplo de *diagrama de planos* referente a capacidade de *Troca de Informação*.

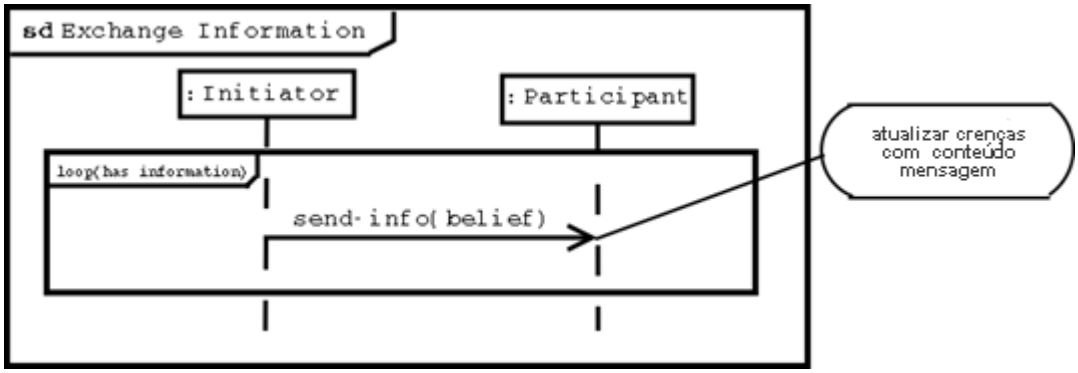

Figura 31. Exemplo de diagrama de planos

No diagrama de interações de agentes, o diagrama de seqüência da AUML é utilizado (Figura 11). Os agentes correspondem aos objetos que possuem comunicações representadas por arcos de mensagens assíncronas.

3.4.5. A técnica de projeto na metodologia PASSI

PASSI [COSSENTINO, 2002] (Figura 32) é uma metodologia que aborda as fases de análise e projeto de aplicações baseadas em agentes, que integra os modelos de projeto da orientação a objetos e a notação UML. A PASSI aborda os conceitos de agentes, papéis e tarefas. Um agente pode desempenhar papéis durante as interações com outros agentes para alcançar seus objetivos, onde um papel tem uma coleção de tarefas a desempenhar para o alcance dos objetivos específicos. Uma tarefa, por sua vez, é definida como uma atividade que o papel desempenha.

Essa metodologia integra as filosofias e conceitos usados na abordagem orientada a objetos e orientada em agentes, para isso usa as notações da AUML [ODELL, 2000]. A opção pelo uso da AUML, que é uma extensão da UML, e por ser uma linguagem muito difundida no meio industrial e acadêmico. Além disso, seus mecanismos de extensão (estereótipos, restrições) facilitam a representação de projetos orientados a agentes.

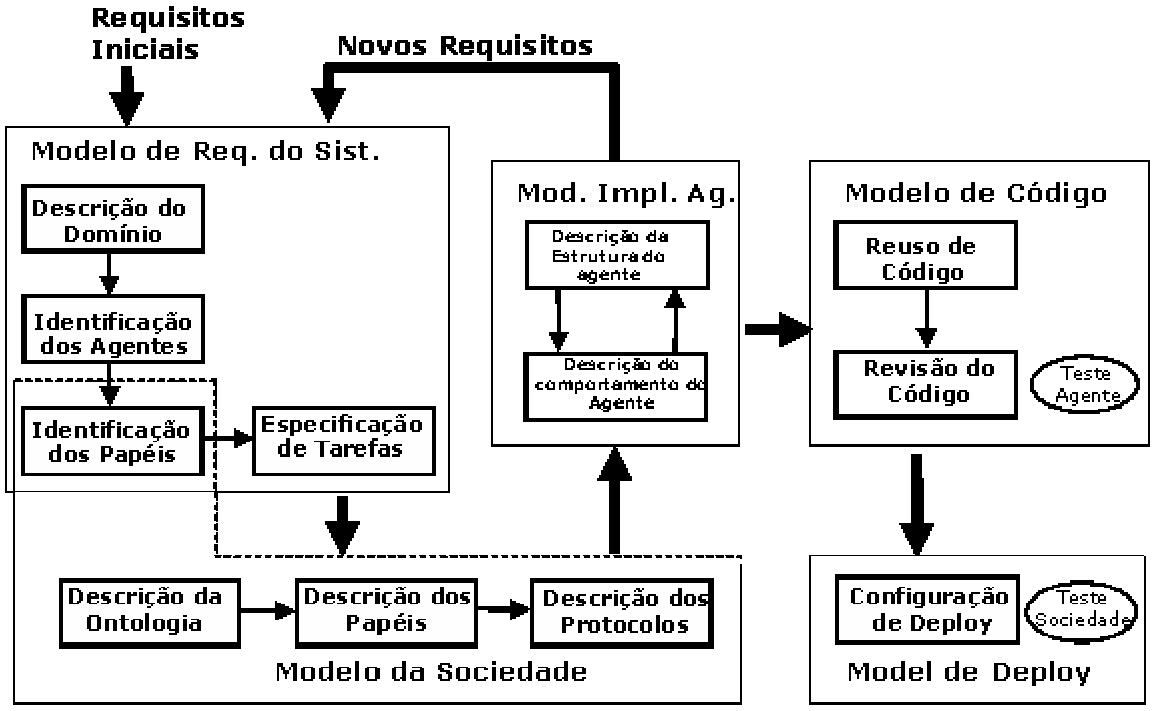

Figura 32. A metodologia PASSI

Essencialmente, as atividades relativas ao projeto segundo essa metodologia são: estruturação do *modelo de implementação de agentes*, *de código* e *de desenvolvimento*.

O *modelo de implementação de agentes* é uma solução em termos de classes e métodos. Para construí-lo, é necessário que seja definida a estrutura através de diagramas de classes (Figura 33) e o comportamento através de diagramas de atividades (Figura 13) dos agentes.

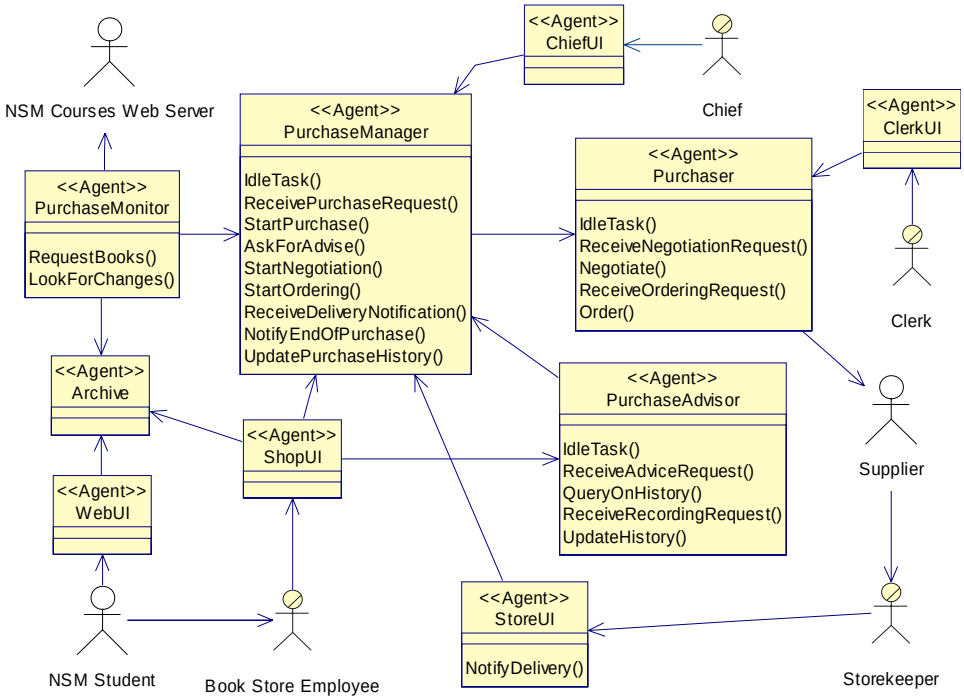

Figura 33. Exemplo de diagrama de classes na PASSI

Já o *modelo de código* provê a solução em nível de código. Para tanto, faz-se necessário codificar a biblioteca de Reuso (biblioteca de classes e diagramas de atividades com o código de reuso associado) e construir o código fonte final do sistema.

Finalmente, na estruturação do *modelo de desenvolvimento*, tem-se a distribuição das partes do sistema pelas unidades de hardware. Para isso, utiliza-se o *diagrama de distribuição*, o qual descreve a alocação dos agentes às unidades de processamentos disponíveis (Figura 34).

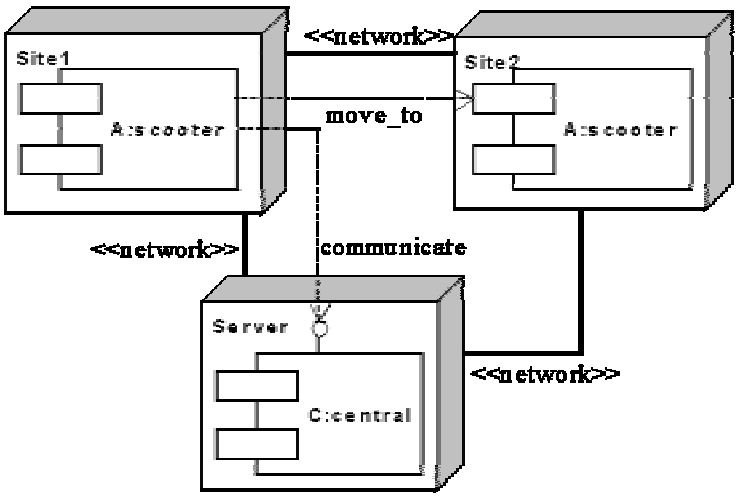

Figura 34. Exemplo de diagrama de distribuiçao na PASSI

3.4.6. Análise Comparativa

Metod	Atividade	Produto do Projeto
GAIA	Identificar os agentes através dos papéis	Modelo de Agentes
	Definir as comunicações existentes Definir a organização do sistema	Modelo Social
	Definir as atividades e papéis desempenhados pelos agentes.	Modelo de Serviço
MaSE	Criação das classes de agentes	Diagrama de Classes de Agente
	Identificação Conversações Organização das conversações	Diagrama de Conversações
	Junção das classes de agentes	Arquitetura dos Agentes
MADS	Identificar tipos de Agentes	Diagrama de Agentes
	Especificar a arquitetura Global	Diagrama de Arquitetura
	Detalhar atividades	Diagrama de Atividades
TROPOS	Refinamento do diagrama de atores	Diagrama de atores estendido
	Identificação das capacidades Atribuição das capacidades aos agentes Especificação das interações dos agentes	Diagrama de atores para análise de capacidades
	Projeto detalhado	Diagrama de capacidade Diagrama de planos Diagrama de interação dos agentes
PASSI	Estruturação do modelo de Requisitos do Sistema	Modelo de Requisitos do Sistema
	Estruturação do modelo da Sociedade de Agentes	Modelo da Sociedade de Agentes
	Estruturação do modelo de Implementação do agente	Modelo de Implementação do agente

Tabela 4. Análise comparativa das técnicas para a fase de projeto

Os produtos e as atividades da fase de projeto das metodologias analisadas estão mostrados na Tabela 4.

A maioria destas técnicas busca a identificação e análise das sociedades dos agentes, das interações entre agentes de forma individualizada.

A GAIA [WOOLDRIDGE, 2000] busca a construção de sociedades de agentes que cooperam para atingir as metas do sistema, definindo o que é necessário que cada agente da sociedade faça para que se alcance este objetivo. Os agentes são identificados (*modelo de agentes*), são atribuídos a eles serviços (*modelo de serviços*) e, então, são analisados em termos de comunicação (*modelo social*).

A MaSE [DILEO, 2002] busca a identificação das classes de agentes, onde os papéis são atribuídos a classes de agentes específicas e então as comunicações ou conversações são definidas, analisando-se as interações entre estas classes de agentes.

A MADS [SODRÉ, 2002] identifica os agentes baseados nos papéis advindos da fase de análise, atribuindo a estes agentes, em seguida, as atividades dos papéis e, por fim, o sistema é analisado de uma forma global com a construção de uma arquitetura de software.

A Tropos [CASTRO, 2001] [MYLOPOULOS, 2000], no projeto, trabalha com duas fases bem definidas: o projeto arquitetural, onde os agentes e suas capacidades são identificados, e o projeto detalhado onde os agentes são analisados individualmente e representados através de diagramas da AUML [ODELL, 2000].

A PASSI [COSSENTINO, 2002] trabalha na identificação dos agentes e suas estruturas, incluindo uma aproximação direta da implementação, com a construção de modelos de código e modelos de desenvolvimento.

Outro aspecto importante é a informação do domínio, que é benéfica para a fase de análise e conseqüentemente para o projeto. As metodologias que levam em consideração a informação do domínio são: a MASE através da construção da ontologia de domínio e a PASSI através da construção da ontologia de domínio.

Foram identificados dois tipos de modelo de desenvolvimento nas metodologias analisadas, o seqüencial e o interativo. A metodologia que utiliza o modelo de desenvolvimento seqüencial é a PASSI. As metodologias que utilizam o modelo de desenvolvimento interativo são: GAIA, MADS, MaSE e Tropos.

As metodologias analisadas utilizam notação gráfica, que traz vantagens, pois leva o desenvolvedor a agir de uma maneira mais intuitiva e leva a uma visualização mais clara dos conceitos.

3.5. Considerações finais

Neste capítulo foi apresentado o projeto de sistemas multiagentes. Inicialmente, foram destacados os conceitos e utilidades de agentes e sistemas multiagentes, seguido de um detalhamento da fase de projeto em termos de atividades e classificação das arquiteturas dos sistemas multiagente e dos agentes.

Foi também feita uma análise comparativa das principais metodologias da Engenharia de Software baseada em Agentes, enfatizando as técnicas para a fase de projeto dos sistemas multiagente.

Tais metodologias vêm contribuindo para o desenvolvimento de software baseado em agentes, por utilizarem o conceito de agente, adequado para a concepção e construção de sistemas complexos, distribuídos e heterogêneos.

A linguagem de modelagem para agentes AUML foi apresentada mostrando seu uso nos vários níveis do projeto através de exemplos. Esta linguagem é utilizada na técnica proposta neste trabalho, detalhada no capítulo a seguir.

4. DDEMAS – Uma técnica para o Projeto de domínio de aplicações multiagente

A maioria das técnicas analisadas na seção 3.4 buscam a construção de aplicações multiagente específicas. Este trabalho visa a construção de artefatos reutilizáveis, como frameworks e componentes agentes. Por esta razão, fez-se necessária a definição da técnica DDEMAS ("Domain design for multi-agent systems") [FERREIRAa, 2003] que, diferentemente das outras, busca a construção de soluções computacionais para uma família de sistemas multiagente. Através da utilização de informações contidas em um modelo de domínio em conjunto com uma coletânea de padrões arquiteturais e de projeto detalhado é feita a construção de frameworks multiagente (Figura 35).

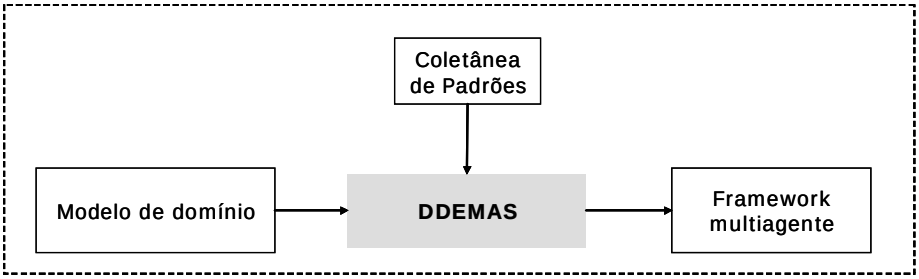

Figura 35. Técnica DDEMAS

O modelo de domínio utilizado na DDEMAS é criado na fase de Análise de domínio utilizando a técnica GRAMO [FARIAb, 2003]. A técnica GRAMO define as atividades a serem realizadas na construção de modelos de domínio e usuários. O modelo de domínio é constituído dos modelos de conceitos, de objetivos, de papéis e de interações. No modelo de usuários estão especificadas as informações dos usuários. A seguir temos uma breve descrição destes modelos.

No *modelo de conceitos* estão especificados os conceitos e relacionamentos do domínio. No *modelo de objetivos* estão especificados o objetivo geral, os objetivos específicos e as responsabilidades. Os objetivos são metas a serem atingidas pelo sistema. O objetivo geral refere-se ao problema que o sistema se propõe a resolver. Os objetivos específicos referem-se ao refinamento ou especialização do objetivo geral. As responsabilidades são derivadas a partir dos objetivos específicos, ou seja, as responsabilidades levam ao cumprimento dos objetivos específicos.

No *modelo de papéis* estão especificados os papéis e seus respectivos atributos. Um papel representa uma função que uma entidade desempenha em uma determinada organização. Cada papel é derivado a partir das responsabilidades especificadas no *modelo de objetivos*. As atividades de cada papel também surgem da sua responsabilidade, ou seja, do conjunto de atividades que o papel deverá realizar para cumprir sua responsabilidade. Os recursos são conceitos de domínio derivados a partir das atividades, isto é, são requeridos pelas atividades, para que elas possam ser executadas. Existem dois tipos de *modelos de papéis*: um *modelo de papel geral*, que contém todos os papéis do sistema com seus respectivos atributos e os *modelos de papéis específicos*, que contêm a especificação de cada papel com seus respectivos atributos.

No *modelo de interações* entre papéis estão especificadas as interações entre papéis ou entre papéis e entidades externas envolvidas no alcance de um objetivo específico. As interações e as entidades externas são derivadas a partir das atividades que os papéis realizam no cumprimento de suas responsabilidades. Existe um *modelo de interações* para cada objetivo específico no *modelo de objetivos*.

São também utilizados, na construção do framework, padrões contidos em uma coletânea de padrões arquiteturais, composta do APSMAD ("*Architectural pattern system for multi-agent application development*") [SILVA, 2003] e no APSUMAS ("*Agent-based pattern system for user modeling and adaptation of systems*") [RIBEIRO, 2004]. APSMAD é um sistema de padrões para o projeto arquitetural para o desenvolvimento de sistemas multiagentes. APSUMAS é um sistema de padrões baseados em agentes para o projeto de sistemas multiagentes que utilizam a modelagem de usuários para oferecer serviços personalizados a usuários e/ou grupos de usuários.

A técnica DDEMAS é composta de três fases, através das quais é especificado o projeto de uma família de sistemas multiagente em ordem crescente de detalhamento: *modelagem de agentes, interações e atividades*, *projeto global* e *projeto detalhado*. Nestas fases é utilizada a notação dos diagramas da AUML [ODELL, 2000] na representação de alguns dos produtos resultantes da execução destas fases.

Busca-se durante as fases, a realização das tarefas para a construção do *modelo arquitetural* e o *modelo de projeto detalhado*, produtos de software que representam uma

completa especificação da solução de projeto de uma família de sistemas multiagente. Na Tabela 5 temos as fases, tarefas e produtos da DDEMAS.

<table>
<tr><th>Fases</th><th colspan="2">Tarefas</th><th>Produtos</th></tr>
<tr><td rowspan="3">Modelagem de agentes, interações e atividades</td><td rowspan="3">Modelagem de agentes</td><td rowspan="3">Modelagem de interações e atividades</td><td>Modelo de agentes</td></tr>
<tr><td>Modelo de interações</td></tr>
<tr><td>Modelo de atividades</td></tr>
<tr><td rowspan="3">Projeto global</td><td colspan="2">Construção do esboço do framework</td><td>Esboço do modelo arquitetural</td></tr>
<tr><td colspan="2">Seleção de padrão arquitetural</td><td rowspan="2">Modelo arquitetural</td></tr>
<tr><td colspan="2">Refinamento do framework</td></tr>
<tr><td rowspan="3">Projeto detalhado</td><td colspan="2">Detalhamento dos agentes do framework</td><td>Modelo de atividades detalhado</td></tr>
<tr><td colspan="2">Seleção de padrão de projeto detalhado</td><td rowspan="2">Modelo de projeto detalhado</td></tr>
<tr><td colspan="2">Refinamento dos agentes</td></tr>
</table>

Tabela 5. As fases, tarefas e produtos da técnica DDEMAS

A primeira fase visa a construção dos *modelos de agentes*, *interações* e *atividades*, através dos quais são especificados os agentes e suas interações. Na segunda fase é feito o projeto global da família de sistemas, onde é criado o *modelo arquitetural* representado o framework multiagente, com os agentes organizados segundo mecanismos de coordenação e cooperação apropriados ao contexto do problema. Na ultima fase, são construídos o *modelo de atividade detalhado* e o *modelo de projeto detalhado*. Estes modelos são referentes ao detalhamento do processamento e dados internos dos agentes que constituem o framework.

As fases, tarefas e produtos da DDEMAS são detalhados a seguir. É também apresentado um conjunto de exemplos da execução das tarefas e dos produtos resultantes da aplicação da técnica no domínio do acesso a informação. Na seção 6.1 são descritos os principais conceitos referentes ao acesso a informação utilizados nestes exemplos.

4.1. Modelagem de agentes, interações e atividades

Nesta fase são identificados os agentes que irão compor o framework e suas interações com base em informações advindas da fase de Análise de domínio [FARIAa, 2003] [FARIAb, 2003] [FARIAc, 2003] [GIRARDIb, 2003]. São construídos os *modelos de agentes*, *interações* e *atividades*, através da execução das tarefas de *modelagem de agentes* e *modelagem de interações e atividades*, detalhadas a seguir.

4.1.1. Modelagem de agentes

O objetivo desta tarefa é identificar e especificar os agentes que irão compor o framework em um *modelo de agentes*.

Tem-se como proposta inicial o mapeamento um "papel" para um "agente", sendo cada papel associado a um agente. Porém, podem existir situações onde um agente pode englobar mais de um papel, ou ainda, da necessidade de atribuir um papel a mais de um agente, de forma a atender requisitos de coesão, desempenho e reusabilidade. Cada agente irá realizar um conjunto de atividades necessárias para o cumprimento de suas responsabilidades, Estas atividades são derivadas diretamente das atividades dos papéis desempenhados pelos agentes.

A heurística é que se defina um agente para cada papel, mostrando graficamente, seu nome, papel(is) correspondente(s), responsabilidades e atividades (Figura 36). A Figura 37 mostra a representação de um agente que desempenha o papel de *Interfaceador*, em um sistema de recuperação de informação.

A identificação das interações entre agentes deve ser executada em paralelo com a identificação dos agentes, pois aspectos como níveis de troca de informações ou utilização de recursos podem influir na definição dos agentes.

Deve-se fazer uma análise da necessidade de se fundir agentes que possuam responsabilidades afins, atividades próximas ou comunicação em excesso, ou da necessidade de se atribuir atividades de um papel a mais de um agente.

É importante ressaltar que, nesta tarefa, é realizada apenas a identificação dos agentes que irão constituir o framework, sem a preocupação com sua arquitetura interna. Estes aspectos são tratados posteriormente na fase de projeto detalhado.

4.1.2. Modelagem de interações e atividades

Esta tarefa esta intimamente ligada à tarefa de *modelagem de agentes*. Junto com a identificação dos agentes é preciso identificar e especificar suas interações no *modelo de interações*.

As interações nas quais um agente participa são listadas no *modelo de agentes* (Figura 36). O detalhamento das interações (origem, destino e seqüência) é apresentado no *modelo de interações.*

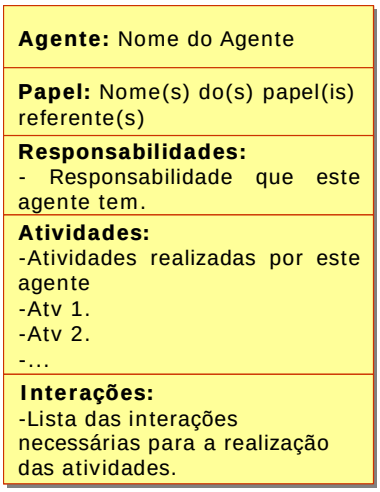

Figura 36. Representação de um agente e suas interações no *modelo de agentes*

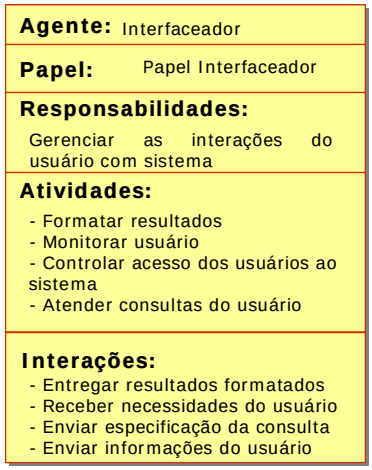

Figura 37. Representação do agente *interfaceador* e suas interações.

Fazer um mapeamento do *modelo de interações entre papéis* ao *modelo de interações entre agentes* considerando os papéis que cada agente desempenha. A representação gráfica deste modelo segue a representação do diagrama de pacotes da AUML [ODELL, 2000]. A Figura 38 mostra um exemplo de *modelo interações* entre agentes, onde são ilustradas as interações existentes associadas ao objetivo específico *satisfazer as necessidades de informação pontuais dos usuários.* A figura mostra inicialmente o *usuário* fazendo uma requisição baseada na sua necessidade pontual ao agente *Interfaceador* que a repassa ao agente *Recuperador* que requisita índices construídos pelo agente *Indexador* utilizando-se das páginas recuperadas pelo agente *Descobridor*. Então, os itens de informação recuperados são enviados ao *Interfaceador*, a fim de que este os envie ao usuário.

As atividades dos agentes em conjunto com as interações devem ainda ser mostradas em *modelos de atividades.* As atividades deste modelo são recuperadas das listas de atividades dos agentes contidos no *modelo de agentes.* É construído um *modelo de atividades* para cada *modelo de interações* entre agentes.

Um *modelo de atividades* mostra as operações que os agentes realizarão e como estas operações serão ativadas. A representação gráfica deste modelo segue o padrão do diagrama de atividades da AUML [ODELL, 2000], tem-se os agentes mostrados no topo e

abaixo destes suas atividades. A representação das atividades deve estar disposta de uma forma que possa mostrar seu fluxo de execução.

A Figura 39 mostra o exemplo de um *modelo de atividades*, que descreve as atividades envolvidas na busca do objetivo de *satisfazer as necessidades de informação pontuais dos usuários*. Neste modelo, o processo inicia-se com a atividade do agente *Descobridor* de *descobrir itens de informações*, seguindo com as atividades referentes a representação e indexação dos itens, *construir surrogate de itens de informação* e *indexar surrogate de itens de informação*. Outro ponto de início do processo na busca do objetivo é a atividade de *atender necessidades do usuário* pelo *Interfaceador*, passando a representação desta consulta, *construir surrogate da consulta* pelo agente *Construtor de Surrogate*; busca de itens relativos a consulta, *requisitar índice* e *recuperar índices*; comparação de surrogates de consulta e elementos de informação pelo agente *Recuperador*, *comparar surrogate*, *fazer análise de similaridade* e *enviar resultados*; finalizando com *entregar resultados formatados* pelo *Interfaceador*.

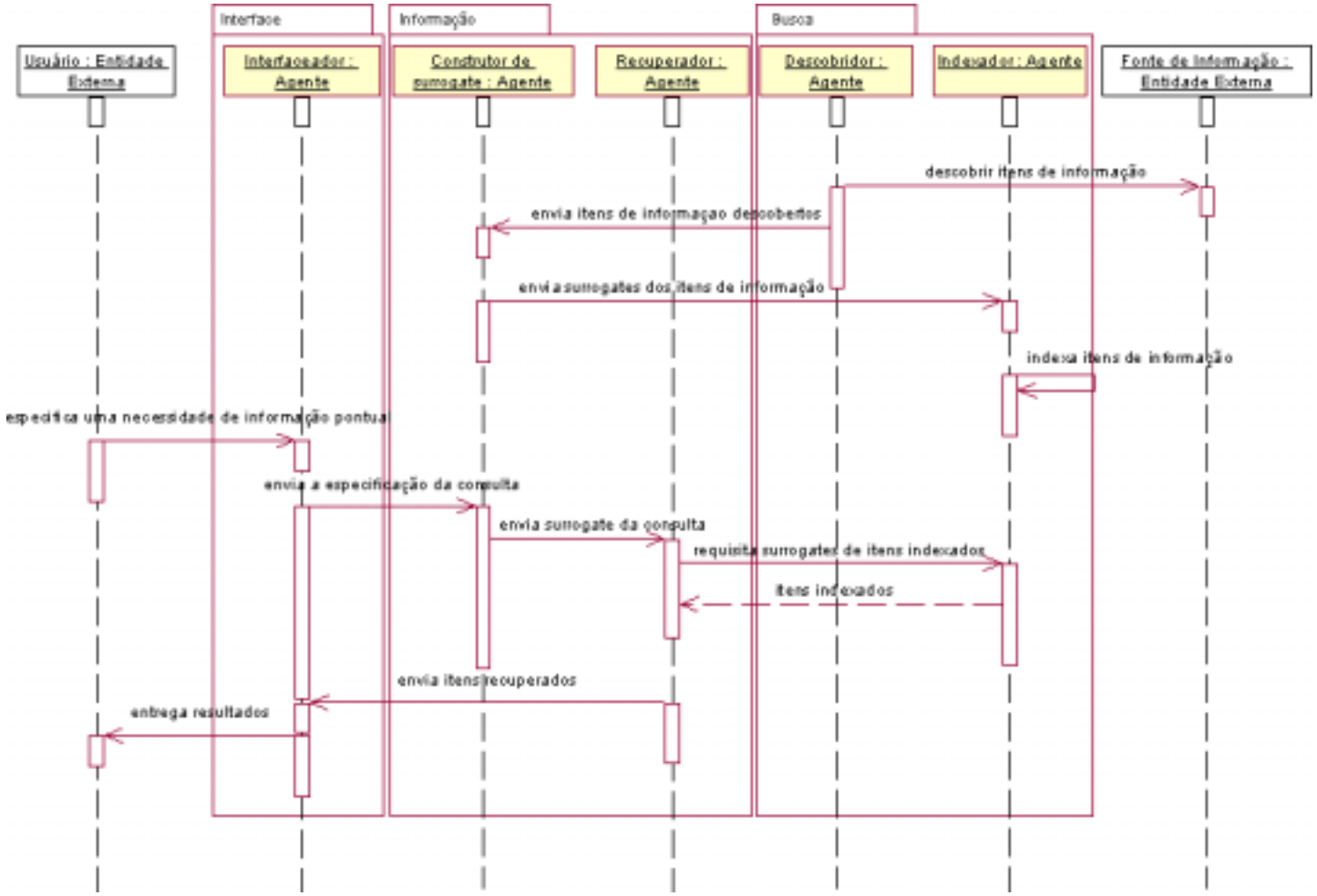

Figura 38. Exemplo de modelo de interações entre agentes para o objetivo específico *satisfazer as necessidades de informação pontuais dos usuários*

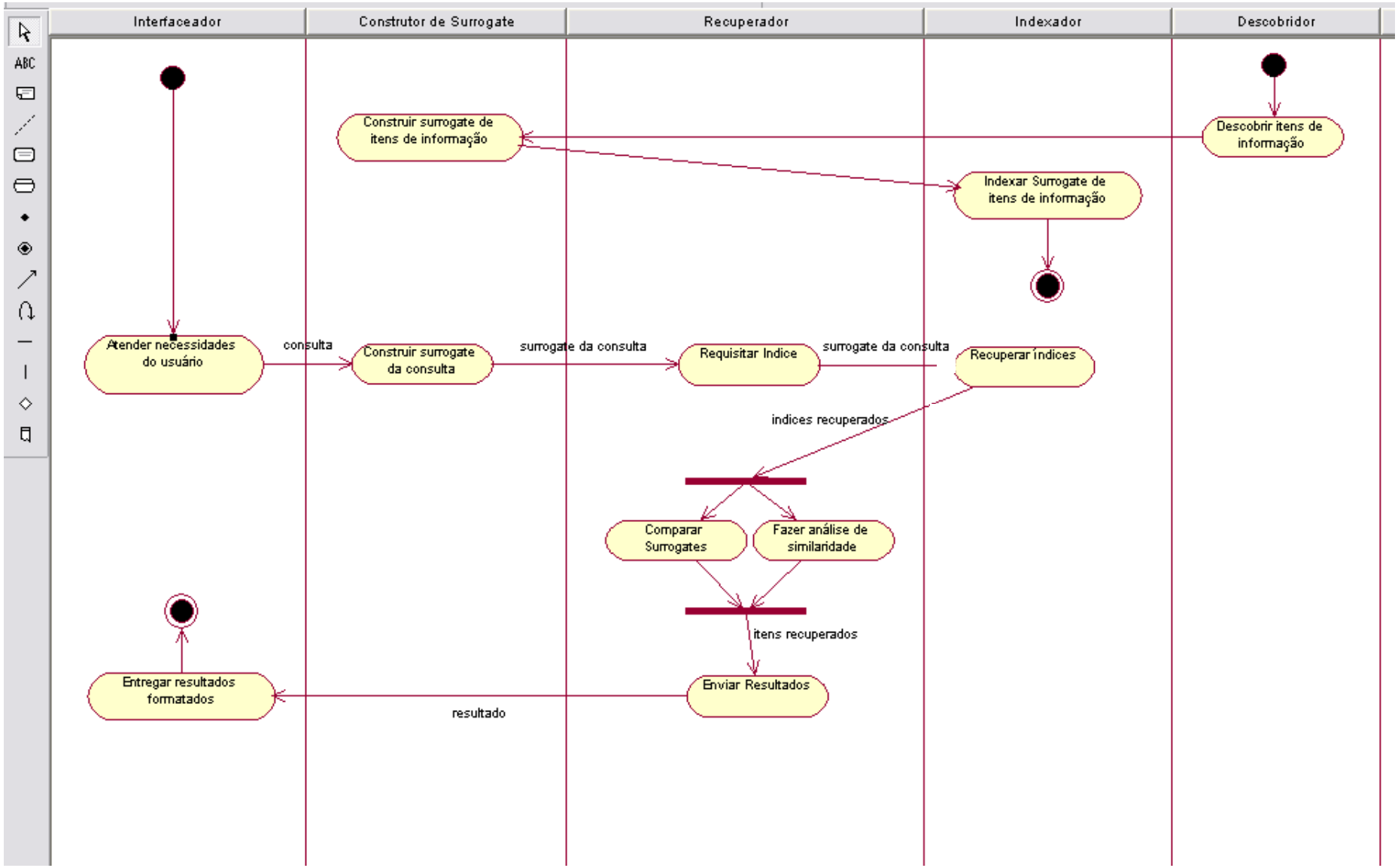

Figura 39. Exemplo de modelo de atividades para o objetivo específico *satisfazer as necessidades de informação pontuais dos usuários*

4.2. Projeto global

Nesta fase é construída a arquitetura genérica do sistema. Os agentes modelados na fase anterior são integrados para formar uma solução computacional ao problema tratado. São mostrados os agentes, entidades externas, suas interações e dependências. Estes agentes são organizados de forma adequada, a fim de que o sistema resolva eficientemente o problema.

O objetivo desta fase é a construção de um *modelo arquitetural* que representa o framework multiagente. Esta fase consiste das seguintes tarefas: *construção do esboço do framework, seleção de padrão arquitetural* e *definição do framework.*

4.2.1. Construção do esboço do framework

Nesta tarefa é especificado um primeiro modelo do framework mostrando os agentes que o compõem, as interações entre eles e as entidades externas à sociedade. O produto desta tarefa é um *esboço do modelo arquitetural*.

Na construção deste esboço devem-se observar os *modelos de agentes, de interações* e *atividades,* que ilustram as interações entre os agentes e entidades externas e as atividades destes agentes na busca dos objetivos do sistema. Deve-se buscar identificar as comunicações entre os agentes e entidades externas, mostrando-as no esboço. Temos um exemplo de um esboço na Figura 40, onde são apresentados os agentes, entidades externas e suas interações em um sistema de recuperação de informação. Dos *modelos de agentes* são extraídos os agentes do framework: *Interfaceador, Recuperador, Construtor de surrogate, Indexador* e *Descobridor*. Dos modelos de interações são extraídas as comunicações entre agentes e entidades externas. Por exemplo, do *modelo de interações* da Figura 38 pode-se observar a comunicação entre o agente *Interfaceador* e o agente *Construtor de surrogate*, interação que é então representada no esboço do framework da Figura 40.

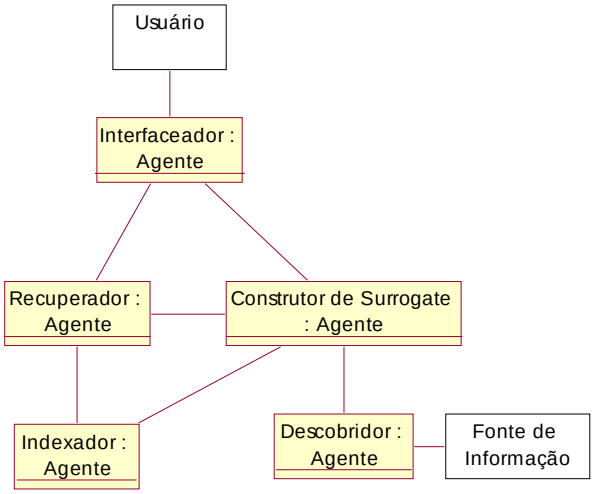

Figura 40. Esboço do framework

4.2.2. Seleção de padrão arquitetural

Tendo o framework esboçado, deve-se buscar refinar o esboço inicial utilizando boas soluções arquiteturais conhecidas para o contexto particular de aplicação do framework, considerando aspectos de coordenação e cooperação entre os agentes. Para estes fins é utilizada uma coletânea de padrões arquiteturais para o projeto de sistemas multiagente [SILVA, 2003] [GIRARDIa,2003] [RIBEIRO, 2003] que fornece informações para a estruturação do sistema multiagente, enfatizando os aspectos de coordenação e cooperação entre os agentes da sociedade.

Basicamente os padrões arquiteturais contidos na coletânea são os seguintes: *Quadro-negro, Difusor, Federativo, Mestre-escravo, Agente Negociador, Reunião, Mercado* e *Camada* [SILVA, 2003] [GIRARDIa,2003] [RIBEIRO, 2003].

O padrão *Quadro-negro* tenta resolver o problema de como os agentes de uma sociedade podem ter acesso a informações e conhecimento de maneira facilitada sem se comunicarem diretamente, através da criação de um ambiente de coordenação passivo chamado de Quadro-negro.

O padrão *Difusor* trata o problema de como os agentes podem se comunicar com outros agentes, através de mensagens assíncronas buscando facilitar e agilizar a comunicação. A solução é a seguinte. Para que as trocas de mensagens ocorram de maneira adequada entre os agentes, é necessário estabelecer um protocolo de comunicação. O protocolo é quem dita

as regras e impõe o formalismo necessário para que as mensagens sejam encaminhadas e compreendidas pelos agentes.

O padrão *Federativo* aborda o problema de troca de mensagens em *broadcasting* que, podem eventualmente inviabilizar todo o processo de comunicação de um sistema, solucionado-o através da divisão da sociedade em grupos (federações) menores seguindo um critério de semelhança. Em cada um desses grupos existe um agente facilitador que é responsável por receber e encaminhar a mensagem para o agente destinatário.

O padrão *Mestre-escravo* é aplicado no problema de delegar sub-tarefas e coordenar sua execução. A solução envolve um Mestre, que divide a tarefa em sub-tarefas, em seguida delega essas sub-tarefas a Escravos e depois os resultados parciais são enviados dos Escravos para o Mestre que tem a responsabilidade de computar o resultado final.

O padrão *Agente Negociador* tenta resolver o problema de como descobrir e tratar interações que são conflitantes entre os agentes, através da solução que envolve um Iniciador que começa um círculo de negociação declarando sua intenção para seus pares, que são todos os agentes que devem ser consultados antes do Iniciador prosseguir com suas ações planejadas.

O padrão *Reunião* é aplicado ao problema de como os agentes aceitam coordenar suas tarefas e mediar suas atividades. A solução é a seguinte: cria-se um lugar em um ambiente para que os agentes possam se reunir. Em seguida, deixa-se um agente chamar para uma reunião os diversos agentes da sociedade, permitindo que as interações possam ocorrer no contexto da reunião.

O padrão *Mercado* mostra como se pode relacionar os usuários de um serviço (compradores) com os provedores (vendedores). Isto é feito definindo-se um Corretor que aceita pedidos de Compradores e ofertas de bens (recursos ou serviços) de Vendedores. O Corretor controla a lógica da coordenação, anunciando pedidos a Vendedores.

O padrão *Camada* é aplicado quando se quer estruturar e organizar as dependências entre subsistemas que estão em diferentes níveis de abstração. A solução usada neste padrão envolve a definição de um critério de abstração para a divisão das responsabilidades entre as camadas. Em seguida, determina-se o número de níveis de abstração, dá-se um nome as camadas e associa-se o serviço de cada uma delas.

Esses padrões são reutilizados nesta tarefa para abordar os problemas referentes à organização do sistema, da seguinte forma:

1. Analisa-se o problema ou subproblema que o framework pretende resolver buscando identificar características ou pontos determinantes ;
2. Seleciona-se um ou mais padrões adequados ao problema e contexto abordados;
3. Depois de selecionados os padrões volta-se a analisar o esboço do framework, buscando organizá-lo seguindo a solução proposta pelo(s) padrão(ões). É estabelecido então, uma estrutura organizacional para sociedade multiagente, aplicando-se a solução descrita no padrão;
4. Estabelecer os mecanismos de cooperação e coordenação a serem utilizados pelos agentes da sociedade. Estas informações são também encontradas no padrão selecionado.

4.2.3. Refinamento do framework

Esta tarefa destina-se a construção do *modelo arquitetural* definitivo, que representa o framework final. São levados em conta o *esboço do modelo arquitetural* especificado e as considerações referentes ao padrão selecionado. As soluções dos padrões escolhidos devem ser aplicadas ao problema, seguindo a forma de aplicação descrita no padrão.

O *modelo arquitetural* após aplicada a solução proposta pelo padrão deve representar a sociedade de agentes organizada segundo mecanismos de cooperação e coordenação, na busca da resolução do problema tratado (Figura 41).

Realizando as tarefas de *seleção de padrões arquiteturais* e *refinamento do framework* sobre o exemplo utilizado nas seções anteriores, chega-se a seguinte solução. Primeiramente, verificou-se que existem conjuntos de agentes com responsabilidades afins. Por exemplo, tanto os agentes *Recuperador* quanto *Construtor de surrogate* têm responsabilidades ligadas ao tratamento das informações do processo de recuperação. Também podem ser percebidos agentes delegando e coordenando tarefas, como, por exemplo, o agente *Interfaceador* ao receber uma requisição do usuário delega tarefas ao agente *Construtor de surrogate*. Devido a essas características, o padrão *Camadas* foi então escolhido por prover uma solução onde as camadas dividem os agentes de acordo com suas

responsabilidades e o padrão *mestre-escravo* por prover uma solução de delegação tarefas e coordenação de execução [SILVA, 2003].

A divisão em camadas ficou definida em três (Figura 41): *Camada de gerenciamento do usuário*, a qual interage com o usuário, recebendo requisições e entregando resultados; *Camada de tratamento da informação*, responsável por todo o processamento referente a informações; e a *Camada de busca de informações*, a qual interage com as fontes, armazenando os dados recuperados em forma de índices.

A comunicação entre os agentes (Figura 41) é iniciada com a construção dos índices a partir de páginas recuperadas pelo *Descobridor* (fluxos 1 e 2), representadas internamente pelo *Construtor de surrogate* (fluxo 3), indexadas e armazenadas pelo *Indexador* (fluxo 4). Esses índices são utilizados no processo de recuperação, que através de uma requisição do usuário ao agente *Interfaceador* é iniciado (fluxo 5). O agente *Interfaceador* repassa as requisições do usuário ao *Construtor de surrogate* (fluxo 6), o qual interage com o *Recuperador* (fluxo 7) e este com *Indexador* (fluxos 8 e 9), para a realização da busca, finalizando com a devolução dos resultados ao *Interfaceador* (fluxo 10) e conseqüentemente ao usuário (fluxo 11).

O padrão *mestre-escravo* foi aplicado, dando ao agente *Interfaceador* (mestre) a responsabilidade de fazer requisições ao *Construtor de surrogate* e ao *Recuperador* (escravos) e coordenar o andamento destas requisições.

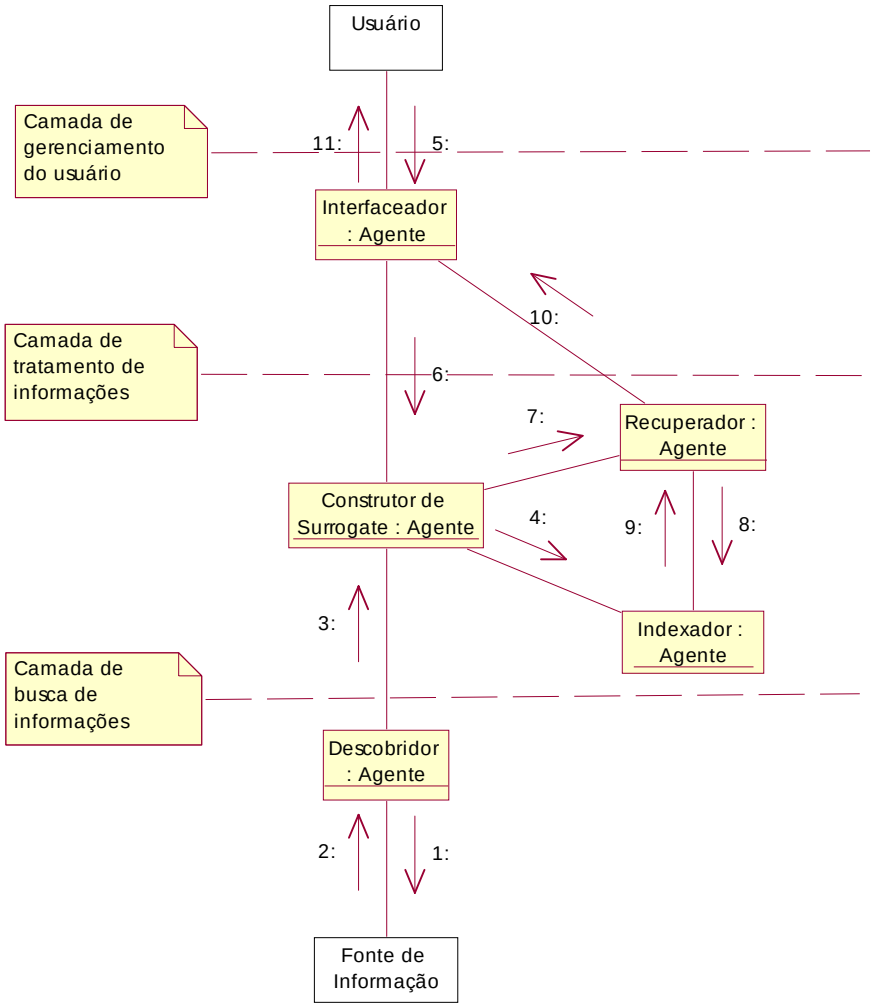

Figura 41. Framework final

4.3. Projeto detalhado

Finalizada a construção do framework representado no *modelo arquitetural*, teremos a estrutura global do sistema definida, iniciando então o detalhamento de cada um dos agentes que compõem o framework, ou seja, o projeto detalhado dos agentes da sociedade.

Como produto desta tarefa, são construídos os *modelos de atividades detalhados*, onde são detalhadas as operações realizadas por cada agente. Também é criado o *modelo de projeto detalhado* que especifica o comportamento e conhecimento de cada agente, estruturados de acordo com padrões de projeto detalhado disponíveis.

A fase de projeto detalhado na DDEMAS consiste das seguintes tarefas: *detalhamento dos agentes do framework, seleção de padrões de projeto detalhado* e *refinamento dos agentes.*

4.3.1. Detalhamento dos agentes do framework

Nesta tarefa é abordado outro nível de abstração do projeto de software: o projeto detalhado dos agentes da sociedade, onde é efetuada uma análise do conhecimento e comportamento de cada um dos agentes do framework. O produto desta fase é um *modelo de atividades detalhado.*

É construído um *modelo de atividades detalhado* para cada agente, através do refinamento da lista de atividades representada no *modelo de agentes* e no *modelo de atividades* obtidos na fase de *Modelagem de agentes, interações e atividades*. Caso surja a necessidade de criar de novas atividades, estas devem ser retratadas nas instâncias dos agentes e no *modelo de atividades.*

O *modelo de atividades detalhado* deve representar todas as atividades de um agente, independentemente dos objetivos a serem alcançados pelo sistema, este propósito já foi tratado pelos *modelos de atividades* e *de interações.* A notação gráfica do modelo de atividades detalhado é a do diagrama de atividades da AUML [ODELL, 2000], ilustrando o conjunto de operações e o fluxo de realização destas. Por exemplo, o agente *Interfaceador* possui as atividades de: *Controlar acesso dos usuários ao sistema*, *Monitorar usuário*, *Enviar informações monitoradas* e *Receber necessidades do usuário* e *Entregar resultados* (Figura 42).

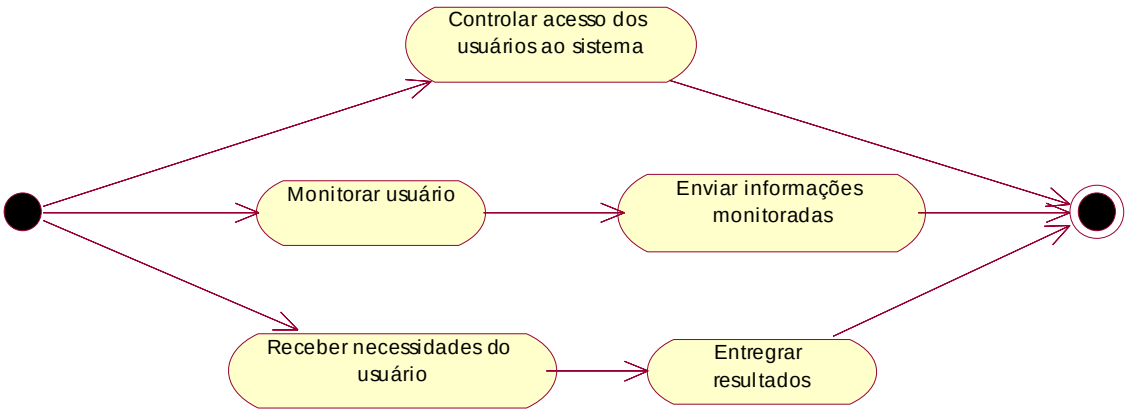

Figura 42. Modelo de atividades detalhado do agente *Interfaceador*

Após criados os *modelos de atividades detalhados*, deve-se fazer uma breve descrição do funcionamento de cada agente identificado, baseando-se nos comportamentos, atividades e operações, para que posteriormente na construção dos agentes estas informações sejam utilizadas.

O comportamento requerido dos agentes é analisado segundo os dois tipos básicos de agentes [RUSSEL, 1995]: reativo e deliberativo. Um agente reativo é uma entidade que age usando um tipo de comportamento baseado em estímulo/resposta, seu comportamento é reduzido à execução de um conjunto de regras de condição-ação (do tipo se...então). Um agente deliberativo é uma entidade que possui um modelo simbólico interno. Ele elabora e seleciona planos ou ações, através de um processo baseado em raciocínio lógico, para que possa atingir sua meta no contexto da situação em questão.

Por exemplo o agente *Interfaceador* possui um comportamento que se resume a receber uma requisição do usuário, encaminhar essa requisição e verificar se a tarefa já foi realizada, para no final entregar o resultado ao usuário. Este conjunto de atividades não requer raciocínio, sendo resolvido apenas com um conjunto de regras do tipo: *verificar se já terminou a recuperação, se sim então recebe e entrega resultados ao usuário*. Portanto, este agente possui as características de um agente reativo.

Outro exemplo é o agente *Modelador*, que possui um conjunto complexo de atividades necessitando possivelmente da elaboração de planos e de raciocínio para a execução destas. Estas características citadas levam a conclusão de que um agente do tipo deliberativo seria a solução mais adequada para estruturar este agente.

De posse das atividades, operações e conhecimento dos agentes, passa-se à construção da arquitetura interna de cada agente.

A arquitetura de um agente mostra como ele está organizado em relação às suas propriedades, à sua estrutura e como os módulos que a compõem podem interagir, garantindo sua funcionalidade. Em suma, a arquitetura de um agente especifica sua estrutura e funcionamento [FERREIRA, 2002].

Na próxima tarefa estes agentes são refinados através da aplicação de padrões de projeto detalhado.

4.3.2. Seleção de padrões de projeto detalhado

Nesta tarefa é realizada uma análise de possíveis padrões de projeto detalhado que se aplicam aos comportamentos dos agentes da sociedade. Para isto verificam-se na coletânea, padrões de projeto detalhado adequados às tarefas destes agentes [RIBEIRO, 2003] [GIRARDIa, 2003].

Os padrões de projeto detalhado disponíveis, basicamente são os seguintes: *Interface*, *Modelagem*, *Adaptação*, *Reativo* e *Deliberativo*. O problema tratado e uma breve descrição da solução destes padrões são mostrados na Tabela 6.

Padrão	Descrição do padrão
Interface	Problema: Como gerenciar a interação entre um usuário ou grupo de usuários e uma aplicação de software de forma personalizada?
	Solução: A solução envolve a criação de um agente que servirá como interface entre o usuário e a aplicação. O agente de interface está associado a um usuário ou grupo de usuários com necessidades similares.
Modelagem	Problema:Como criar e manter modelos de usuários que serão utilizados como referência para que a aplicação ofereça serviços personalizados aos seus usuários?
	Solução: A solução envolve a criação de um agente de modelagem, responsável por de construir e manter modelos de usuários com base nas informações fornecidas por um agente de interface através da interação com o usuário.
Adaptação	Problema: Como construir modelos de adaptação para que a aplicação possa adaptar-se às necessidades de seus diferentes usuários ou grupo de usuários?
	Solução: A solução envolve a criação de um agente de adaptação. O principal papel do agente de adaptação é construir, manter e representar modelos de adaptação de acordo com os modelos de usuários.
Reativo	Problema: Como projetar um agente para apenas reagir a estímulos do ambiente no qual está inserido ou a mensagens de outros agentes quando ele não tem conhecimento sobre esse ambiente e nem pode aprender a partir dele?
	Solução: O agente deve ser do tipo reativo, ou seja, ele não deve ter um modelo interno do ambiente no qual está inserido, o agente deve agir usando um comportamento estímulo/resposta de acordo com o estado corrente do ambiente ao qual está integrado. A solução do padrão reativo estrutura um agente em quatro módulos: comunicação, ação, regras e sensores.
Deliberativo	Problema:Como projetar um agente para que possa raciocinar sobre um problema de forma que o possibilite atingir metas pró-ativamente dentro do contexto no qual está inserido?
	Solução: O agente deve ser do tipo deliberativo, ou seja, ele deve possuir modelos simbólicos internos de raciocínio do ambiente no qual ele está inserido e de si mesmo. Ele raciocina sobre esses modelos para criar um plano que lhe permite atingir suas metas. O padrão deliberativo é estruturado por cinco módulos organizados verticalmente e horizontalmente: módulo de comunicação, módulo de ação, módulo de raciocínio, módulo de sensores e módulo de conhecimento.

Tabela 6. Padrões de projeto detalhado contidos na coletânea.

Através destes padrões são abordados os problemas referentes à arquitetura interna de cada agente.

O processo funciona da seguinte forma:

1. Analisa-se o problema ou subproblema que o agente pretende resolver;
2. Seleciona-se um ou mais padrões que combinem com o problema e o contexto tratados, além de analisar as forças destes padrões;
3. Caso não exista nenhum padrão que aborde o comportamento geral do agente, deve-se buscar utilizar os padrões básicos: deliberativos e reativos, especializando-os segundo os comportamentos específicos.
4. Depois de selecionados os padrões, os agentes são estruturados segundo a solução proposta pelo(s) padrão(ões).

4.3.3. Refinamento dos agentes

Esta tarefa destina-se a criação do *modelo de projeto detalhado* que representa os agentes e suas estruturas internas, definidas através da aplicação dos padrões de projeto detalhado selecionados na tarefa anterior. A estrutura deve estar baseada em módulos que englobam o comportamento e o conhecimento do agente. É construído um *modelo de projeto detalhado* para cada agente.

Na Tabela 7 temos um exemplo de *modelo de projeto detalhado*, ilustrando o detalhamento do agente *Recuperador*, um dos agentes que compõem o framework de recuperação da informação da Figura 41, de acordo à aplicação do padrão deliberativo [OLIVEIRA, 2004].

Este agente não necessita de raciocínio, irá possuir um conjunto de regras, estas regras guiarão as ações executadas pelos agentes. Segundo o *padrão reativo* o agente é estruturado em quatro módulos: *comunicação, ação, regras* e *sensores*.

Através do *módulo sensores*, o agente irá receber as requisições e informações advindas dos agentes *Construtor de surrogate* e *Indexador*. Neste módulo ficarão localizadas as operações *receber surrogate da consulta* e *receber índices indexados*. O recebimento de um surrogate da consulta indica o início do processo de recuperação.

No *módulo regras* são armazenadas o conjunto de regras a serem executadas de acordo com um estimulo recebido pelo *módulo de sensores* e que serão executados pelo *módulo de ação*, algumas das regras contidas é: *se receber requisição surrogate de consulta então requisitar e receber índice; se recebeu índice então realizar comparar surrogates e fazer análise de similaridade*.

Através do *módulo ação* o agente atua no ambiente executando as regras definidas no módulo de regras. As ações contidas neste módulo referem-se às operações de *comparar surrogates* e *fazer análise de similaridade.*

O *módulo ação* serve ao *módulo comunicação* que é responsável por enviar e processar mensagens, realizando as operações *requisitar índice* e *enviar resultados.*

<table>
<tr><th>Agentes</th><th colspan="2">Descrição detalhada do agente</th></tr>
<tr><td rowspan="7">Recuperador</td><td colspan="2">Papel: Recuperador</td></tr>
<tr><td colspan="2">Padrão de projeto detalhado : Padrão reativo</td></tr>
<tr><td colspan="2">Documentação: Este agente busca recuperar um conjunto de informações relevantes a uma consulta pontual do usuário do usuário, ele trabalha tendo como entrada o surrogate da consulta repassada pelo Construtor de Surrogate, os compara com valores dos índices e retorna os elementos recuperados ao Interfaceador.</td></tr>
<tr><td colspan="2">Atividades: receber surrogate da consulta, requisitar índice, receber índices indexados, fazer análise de similaridade, comparar surrogates, elaborar plano de execução da recuperação e enviar resultados.</td></tr>
<tr><td>Módulos</td><td>Conteúdos</td></tr>
<tr><td>Sensores</td><td>Receber surrogate da consulta, receber índices indexados</td></tr>
<tr><td>Regras</td><td>se receber requisição surrogate de consulta então requisitar e receber índice; se recebeu índice então realizar comparar surrogates e fazer análise de similaridade ...</td></tr>
<tr><td>Ação</td><td>Fazer análise de similaridade, comparar surrogates</td></tr>
<tr><td>Comunicação</td><td>Requisitar índice, enviar resultados</td></tr>
</table>

Tabela 7. Tabela representando uma parte do *modelo projeto detalhado*

4.4. Considerações Finais

Neste capítulo foi apresentada a técnica DDEMAS para o Projeto do domínio de sistemas multiagente, suas fases, tarefas e produtos. As tarefas foram detalhadas e exemplificadas na construção dos produtos que representam uma especificação completa de um framework multiagente.

5. Especificação de uma ferramenta para o Projeto de Domínio de sistemas multiagente – ONTODD

Neste capítulo é feita a especificação da ferramenta ONTODD, uma ontologia genérica para o projeto de domínio de sistemas multiagente. Esta ontologia modela o conhecimento da técnica DDEMAS, disponibilizando o conhecimento sobre as fases desta técnica, de forma concisa e clara, facilitando sua aplicação.

A ONTODD é uma ferramenta pois ela serve como um instrumento que auxilia a execução das fases e tarefas da DDEMAS, guiando e facilitando o processo de construção de soluções de projeto reutilizáveis.

O conhecimento modelado pela ONTODD está baseado nos conceitos, tarefas e produtos gerados na fase de Projeto de domínio de sistemas multiagente, para a construção de uma solução computacional reutilizável em relação aos requisitos genéricos de uma família de aplicações especificados na fase de análise de domínio e usuários.

A Figura 43 ilustra o processo de construção da ONTODD, que foi inspirado no método proposto por Fridman para a construção de ontologias [FRIDMAN, 2001]. O processo consiste de duas fases: a definição e o projeto da ontologia. Na fase de definição é utilizado o conhecimento da técnica DDEMAS e conceitos da ONTODUM [FARIAa, 2003] [FARIAc, 2003] [FARIAd, 2003] para gerar uma rede semântica com a representação desses conceitos. Na fase de projeto, a ONTODD é criada através do mapeamento da rede semântica a uma ontologia baseada em frames representada em uma hierarquia de meta-classes.

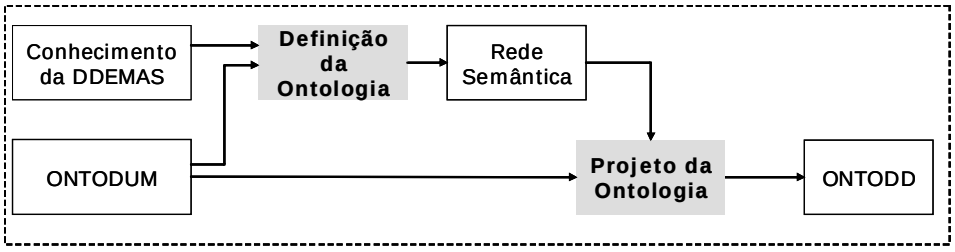

Figura 43. Construção da ONTODD

5.1. ONTODUM - Uma ontologia genérica para a construção de modelos de domínio e usuários

A ONTODUM é uma ontologia genérica para a construção de modelos de domínio e usuários na Engenharia de domínio multiagente [FARIAa, 2003] [FARIAc, 2003] [FARIAd, 2003] [GIRARDI, 2001] [GIRARDI, 2002]. A ontologia representa os conceitos referentes as técnicas para a captura e especificação de requisitos genéricos de sistemas multiagente, técnicas para a captura e especificação dos conceitos de domínio e técnicas para a modelagem de usuários. Ela guia a captura e especificação do domínio do problema, das tarefas a serem realizadas, do modelo de usuários, modelando requisitos genéricos em um domínio e tarefas particulares.

A descrição detalhada da ONTODUM esta fora do escopo deste trabalho. Porém são descritos a seguir, brevemente, alguns conceitos básicos nos quais ela está baseada e que são utilizados na construção da ontologia de projeto ONTODD.

Dentre outros conceitos, ONTODUM representa os conceitos de papéis, objetivos e atividades. Um papel representa uma função que uma entidade desempenha em uma determinada organização. Os objetivos são metas a serem atingidas pelo sistema. As atividades representam conjuntos de tarefas em busca da realização dos objetivos.

O produto da aplicação da ONTODUM em uma determinada área de problema é um modelo do domínio do problema baseado em ontologias que representa os requisitos genéricos dos sistemas nesse domínio. Neste modelo estão contidos os conceitos referentes a papéis, objetivos e atividades de um determinado domínio.

5.2. Construção da ONTODD

A construção da ontologia foi feita utilizando o Protégé [PROTÉGÉ, 2003], uma ferramenta construída pelo departamento de Informática Médica da Universidade de Standford. A ferramenta Protégé é um ambiente de edição de bases de conhecimento que busca a interoperabilidade com outros sistemas de representação do conhecimento, é extensível e é uma ferramenta de aquisição de conhecimento fácil de configurar e usar.

Uma ontologia no Protégé consiste de:

- classes: conceitos no domínio abordado que constituem uma hierarquia taxonômica;

- slots: que descrevem propriedades de classes e instâncias;
- facetas: que descrevem propriedades de slots e permitem a especificação de restrições (constraints) nos valores dos slots;

Esta ferramenta foi selecionada e utilizada, devido a ele ser de domínio público e devido a experiência adquirida em trabalhos anteriores [FARIAa, 2003] [GIRARDIb, 2003].

5.2.1. Definição da ontologia

Na fase de *Definição da ontologia*, o conhecimento da técnica DDEMAS é representado em uma rede semântica. Na Figura 44 e Figura 45 é mostrada a rede semântica que representa o conhecimento da técnica, contendo os conceitos da modelagem, seus relacionamentos, atributos, as tarefas da modelagem e os produtos gerados.

A Figura 44 mostra a parte da rede semântica representando o conhecimento da técnica de projeto relativa aos conceitos da modelagem, seus relacionamentos e atributos. Cada *agente* desempenha um ou mais dos *papéis* definidos na fase de análise. Para isso, executa um conjunto de *atividades* e interage com outros *agentes* e/ou *entidades externas* buscando alcançar os *objetivos* do sistema.

Cada *agente* é estruturado em módulos de acordo com um *padrão de projeto detalhado* segundo o problema a ser resolvido e o contexto de aplicação, os módulos englobam o conhecimento e o comportamento que o agente possui. Os *agentes* são parte de um *framework*, que é estruturado segundo um *padrão arquitetural* que estabelece os mecanismos de coordenação e cooperação entre os *agentes* da sociedade. As entidades representadas por círculos são conceitos associados à técnica DDEMAS, as entidades representadas por octágonos são referentes a ONTODUM, conceitos estes identificados na fase de análise e as entidades representadas por losangos representam instâncias de meta-classes.

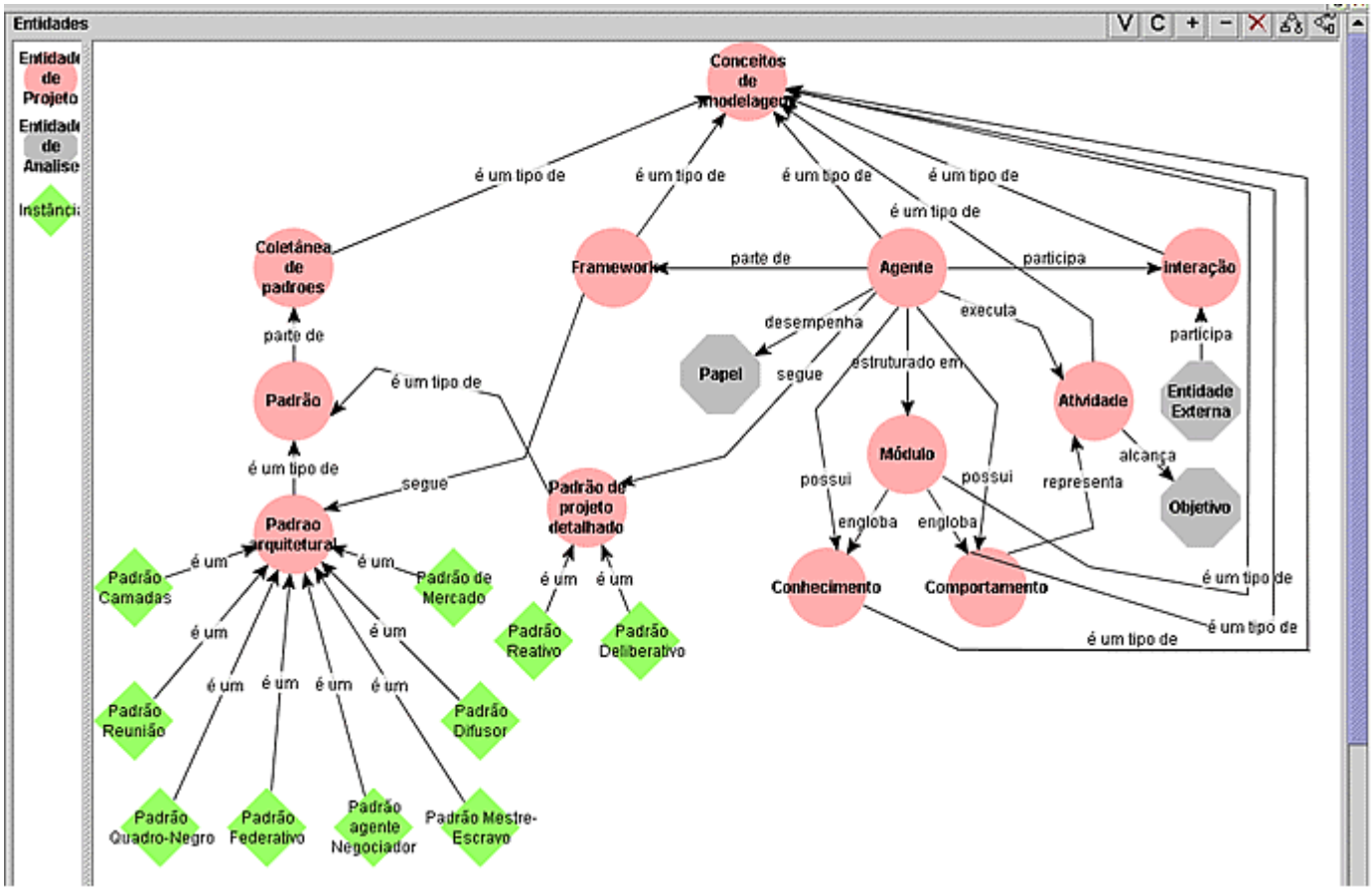

Figura 44. Rede semântica dos conceitos de modelagem da técnica DDEMAS

A Figura 45 mostra a parte da rede semântica representando o conhecimento da técnica de projeto relativa às tarefas de modelagem e aos produtos gerados. São realizadas as tarefas de *modelagem de agentes* e *modelagem de interações e atividades* que irão gerar os *modelos de agentes*, *interações* e *atividades*, produtos que representam os *agentes* pertencentes ao *framework*, *entidades externas*, suas *interações* e *atividades* na tentativa de atingir os *objetivos* do sistema.

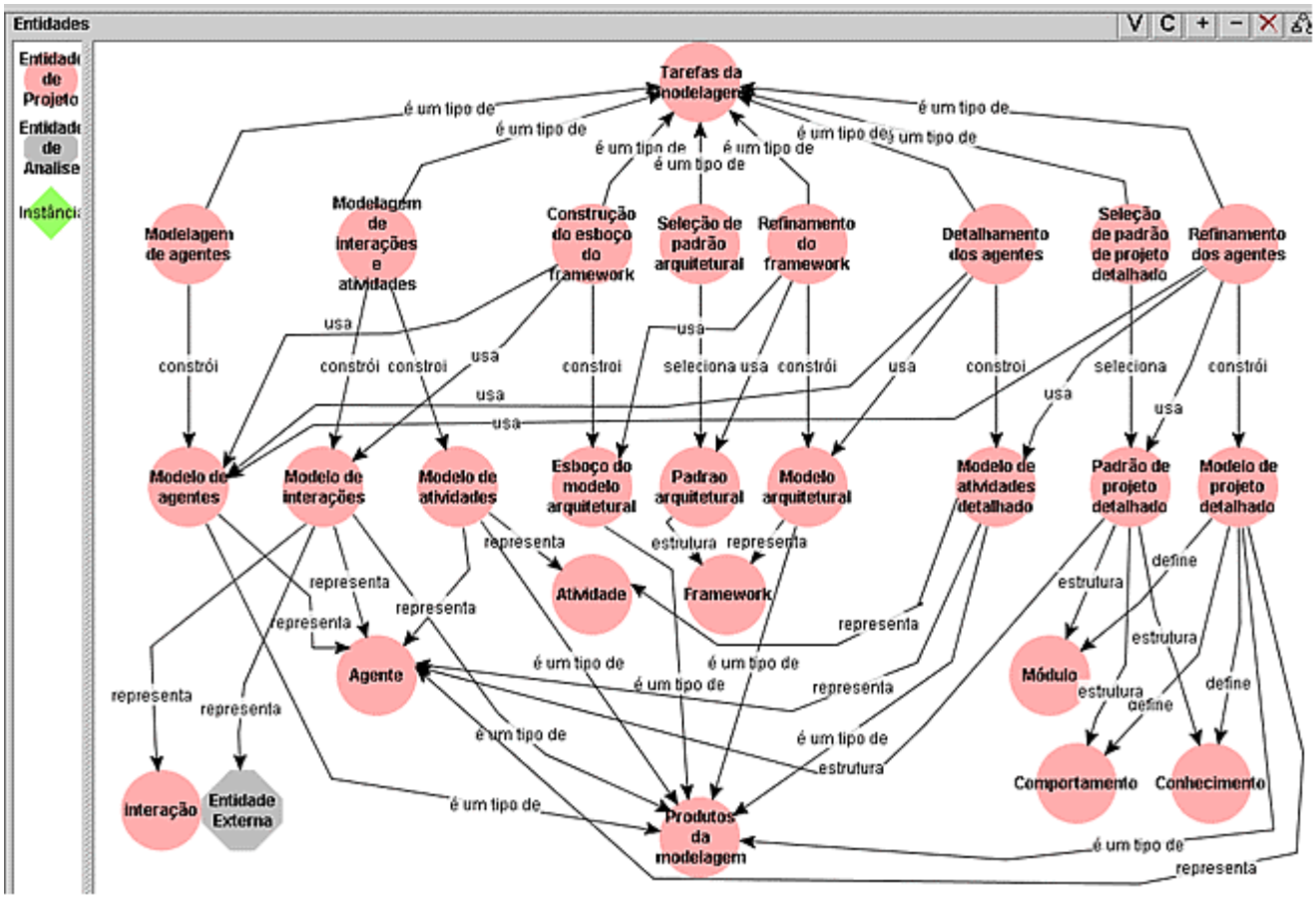

Figura 45. Rede semântica das tarefas e dos produtos da modelagem

As tarefas de *construção do esboço do framework*, *seleção de padrão arquitetural* e *refinamento do framework* tem como produtos um *esboço do modelo arquitetural* e o *modelo arquitetural* definitivo construído segundo um *padrão arquitetural* e que representa o *framework.*

A tarefas de *detalhamento dos agentes*, *seleção de padrão de projeto detalhado* e *refinamento dos agentes* tem como produtos os *modelos de atividade detalhados* e o *modelo de projeto detalhado*, que representa os *agentes* construídos segundo um *padrão de projeto detalhado* .

5.2.2. Projeto da ontologia

Nesta fase os conceitos e relacionamentos da rede semântica são mapeados à ONTODD. Os nós são mapeados para meta-classes. Os nós relacionados por um link do tipo "*é um tipo de*" são mapeados em uma hierarquia de subclasses e superclasses. Outros links são mapeados para slots das meta-classes correspondentes. Cada slot é associado com facetas apropriadas como *tipo* e *cardinalidade*. Nas Figura 46 são mostrados as meta-classes associadas ao nó *Conceitos da modelagem* na rede semântica da Figura 44. A Figura 47 mostra os slots da meta-classe *Padrão arquitetural* obtidos a partir dos links na Figura 44.

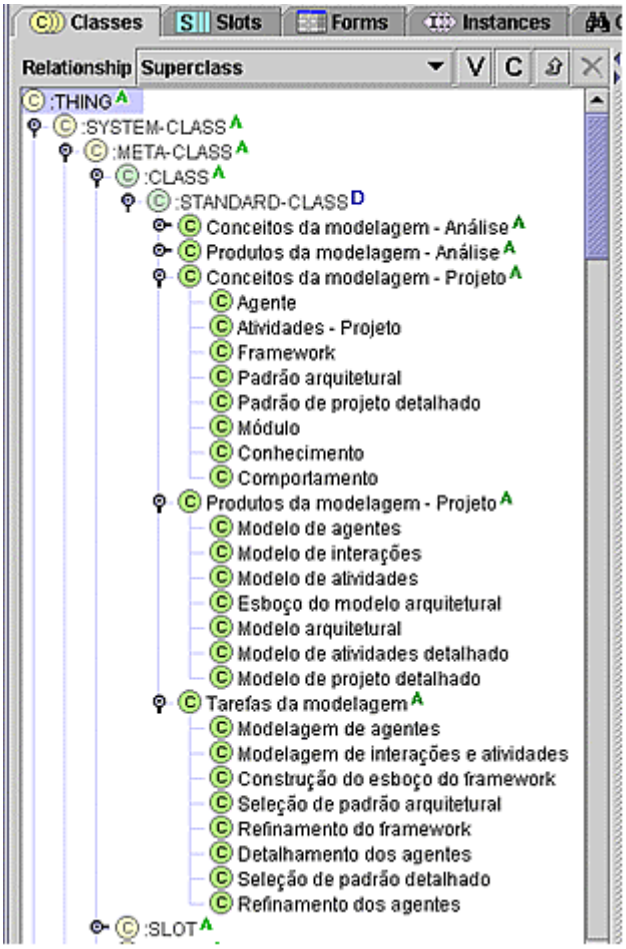

Figura 46. Hierarquia de meta-classes da ONTODD

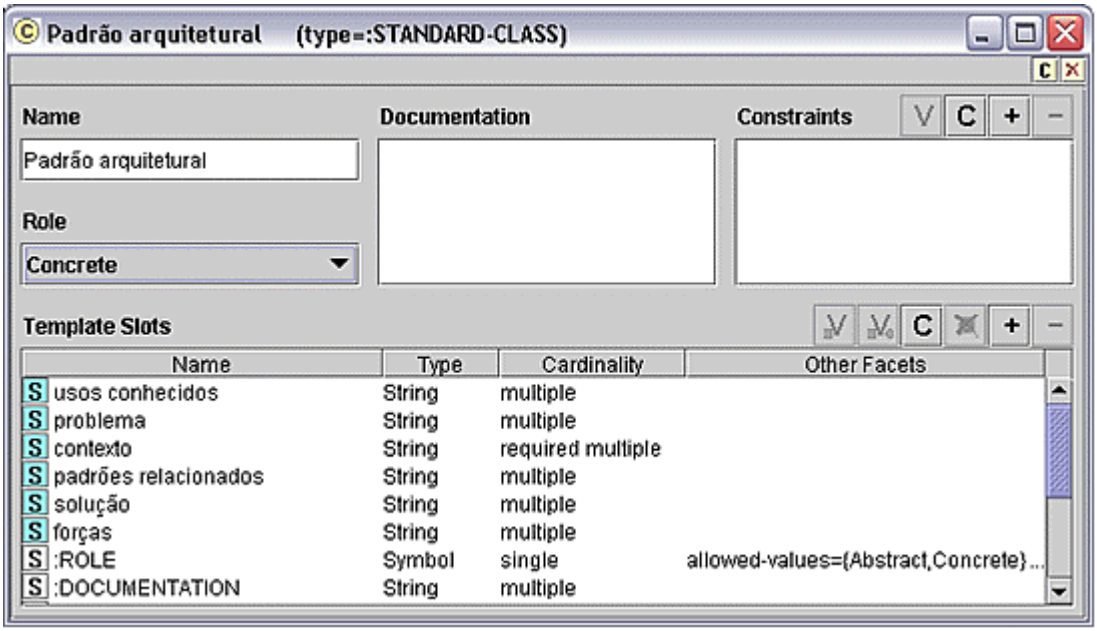

Figura 47. Slots da meta-classe *Padrão arquitetural*

5.3. Construção de frameworks multiagente guiada pela ONTODD

O processo de construção de frameworks multiagente guiado pela ONTODD é ilustrado na Figura 48. Através da instanciação de meta-classes da ONTODD, em conjunto com a utilização de coletâneas de padrões para a fase de projeto são construídos os frameworks multiagente.

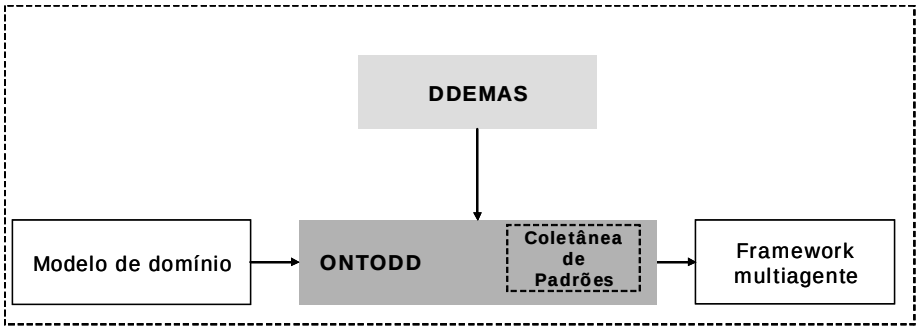

Figura 48. Os insumos e os produtos do uso da ONTODD

Através da instanciação de meta-classes pertencentes à hierarquia de Produtos da modelagem, são construídos os modelos contendo as especificações necessárias para a construção do framework. A seguir é descrito o processo de instanciação da ONTODD segundo as tarefas da DDEMAS.

5.3.1. Modelagem de agentes

A tarefa de *Modelagem de agentes* busca a construção do *modelo de agentes*, uma instância da meta-classe *modelo de agentes*. Para isto, primeiramente são definidos os *agentes*, os quais desempenharão os *papéis* descritos nos *modelos de papéis* (Figura 49) do modelo de domínio ONTOINFO [SERRA, 2003].

É criada uma instancia da meta-classe *modelo de agentes* onde cada papel do *modelo de papéis* é associado a uma instância da meta-classe *Agente*. Temos na Figura 50 um exemplo de *modelo de agentes*, obtido através da execução desta tarefa na construção de um framework para o acesso à informação. Este e os próximos modelos exemplificados são detalhados no estudo de caso apresentado no capítulo 6.

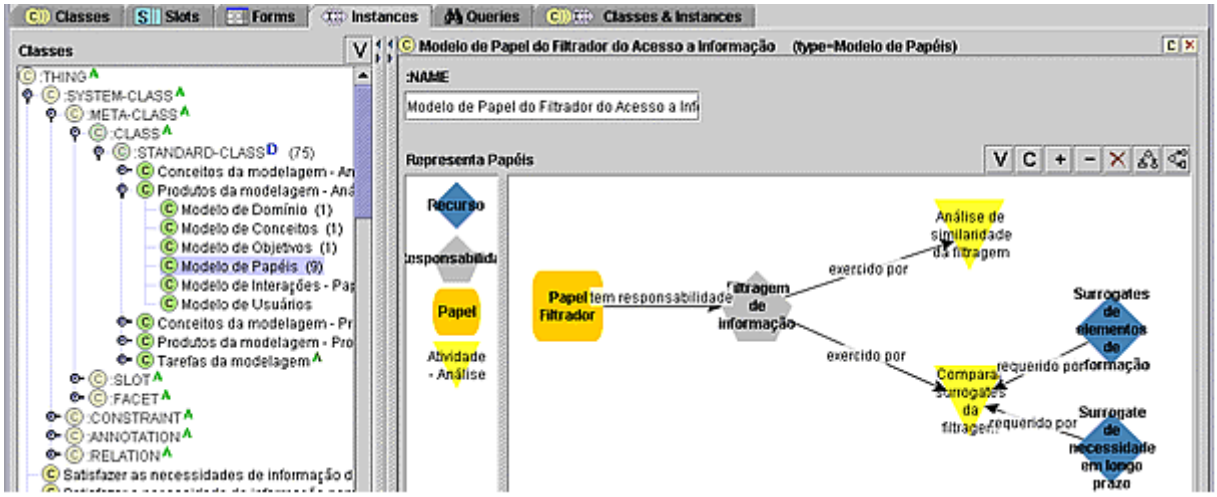

Figura 49. Exemplo de modelo de papéis do modelo de domínio ONTOINFO

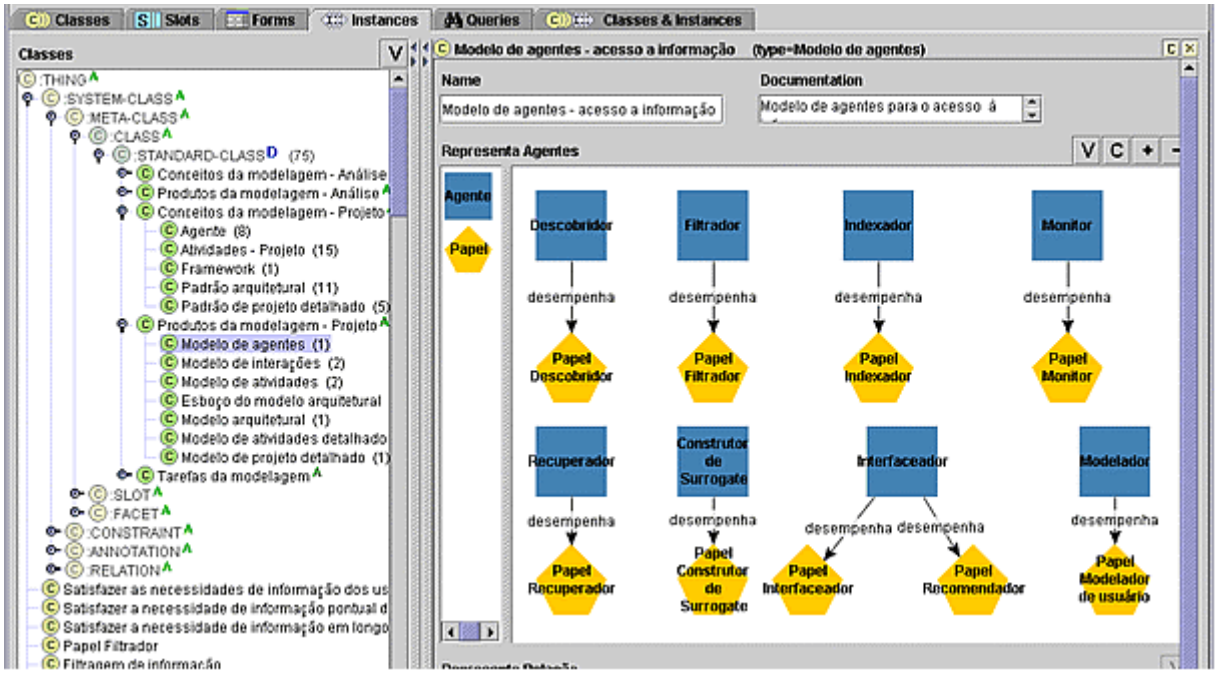

Figura 50. Exemplo de modelo de agentes

As atividades de cada agente são obtidas inicialmente nesta tarefa, a partir das atividades dos papéis correspondentes. A Figura 51 mostra o mapeamento das atividades identificadas na fase de análise à fase de projeto.

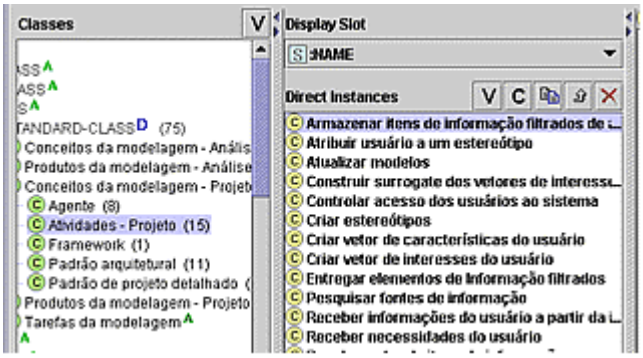

Figura 51. Instancias da meta-classe *Atividade – Projeto*

Estas atividades são documentadas no *slot Realiza* da instancia da meta-classe *Agente* criada. A Figura 52 mostra o exemplo de representação de um agente.

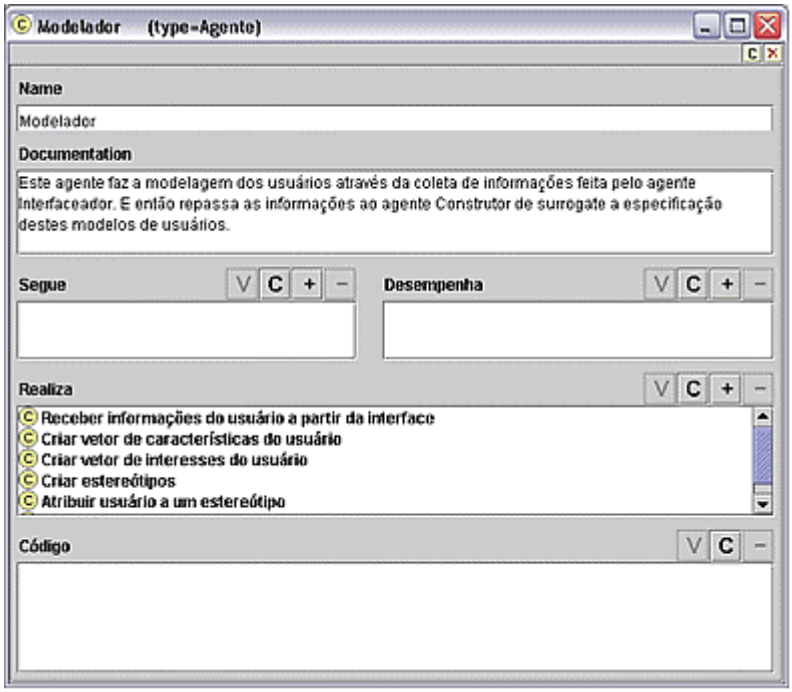

Figura 52. Exemplo de instância de um agente

5.3.2. Modelagem de interações e atividades

A tarefa de *Modelagem de interações e atividades* busca a construção do *modelo de interações* e o *modelo de atividades*, instâncias das meta-classes *modelo de interações* e *modelo de atividades.*

O *modelo de interações* (Figura 54) é obtido a partir de um *modelo de interações entre papéis e entidades externas* (Figura 53) do modelo de domínio.

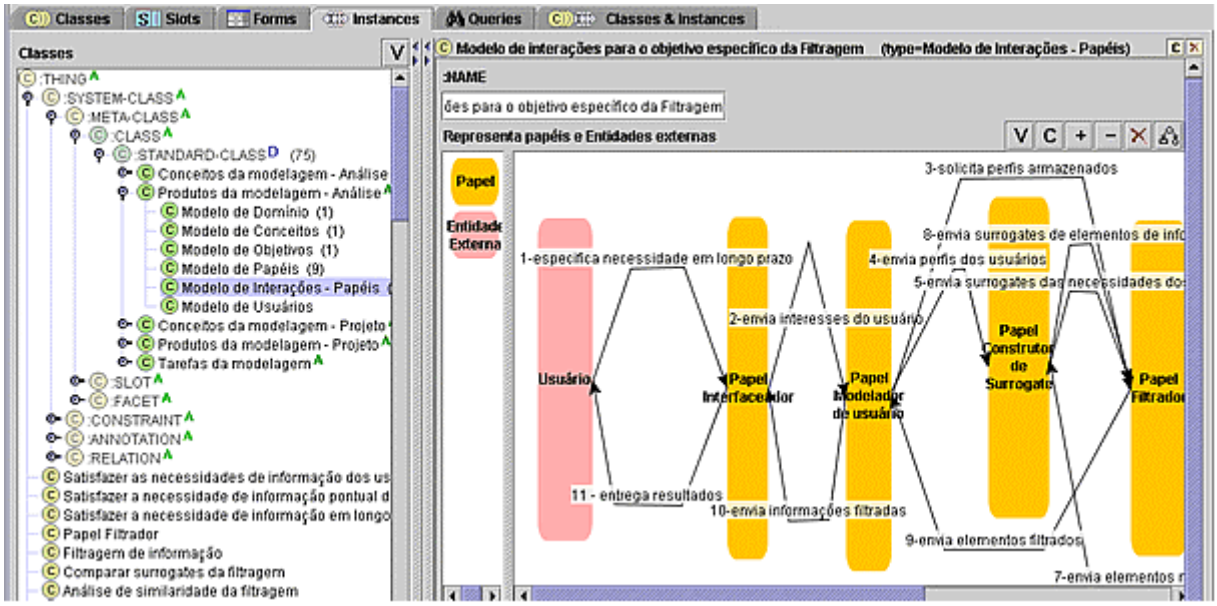

Figura 53. Modelo de interações entre papéis do modelo de domínio ONTOINFO

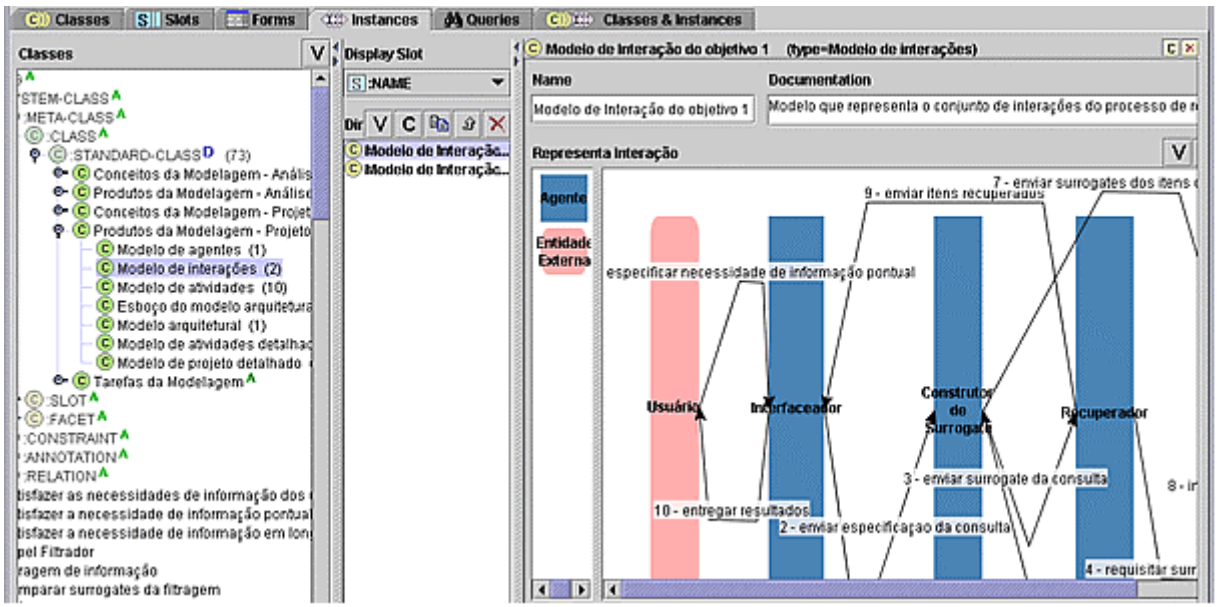

Figura 54. Exemplo de modelo de interações

As *atividades* dos agentes e suas *interações* devem ser mostradas em *modelos de atividades* (Figura 55), estas atividades são obtidas das instancias da meta-classe *agente*. Deve ser instanciado um *modelo de atividades* para cada *modelo de interações*.

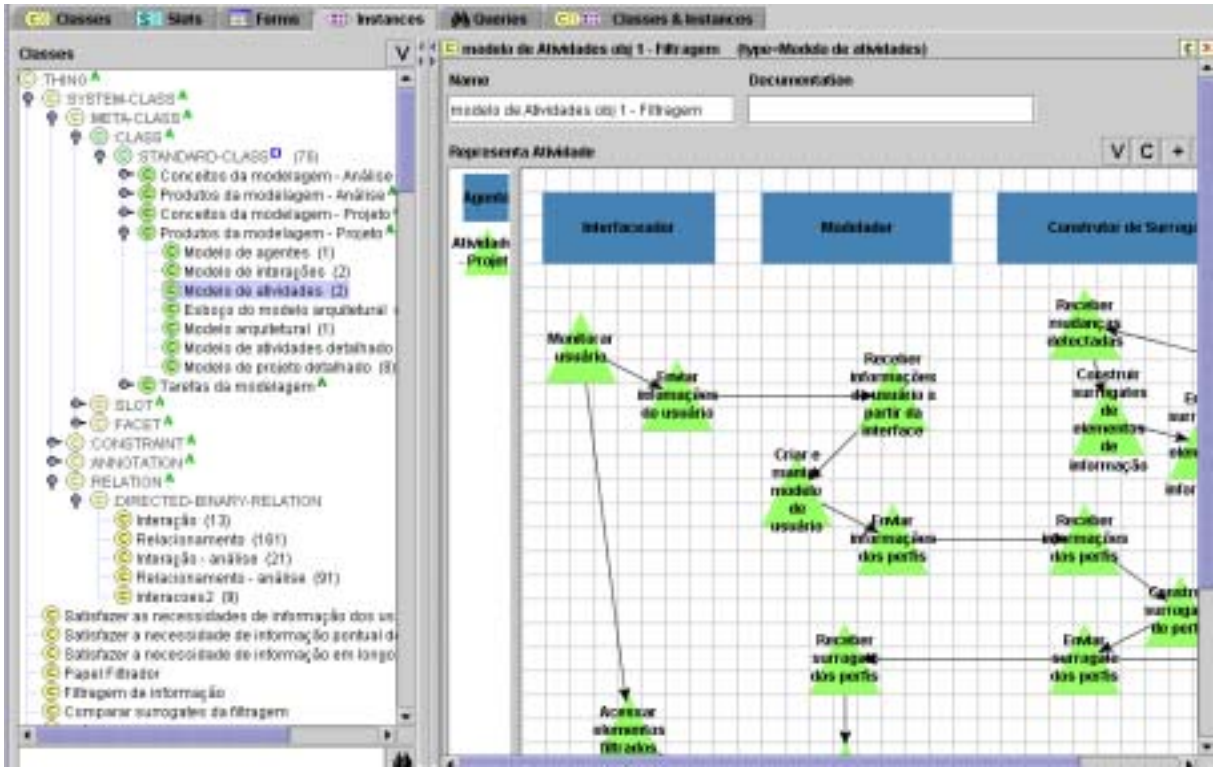

Figura 55. Modelo de atividades

5.3.3. Construção do esboço do framework

A *construção do esboço do framework* representa a primeira tarefa da fase de projeto global. Através desta tarefa o *esboço do modelo arquitetural* é instanciado, representando uma primeira estrutura do framework (Figura 56). Os agentes que compõem este esboço são extraídos do *modelo de agentes* e as comunicações entre agentes e entidades externas são extraídas dos *modelos de interações.*

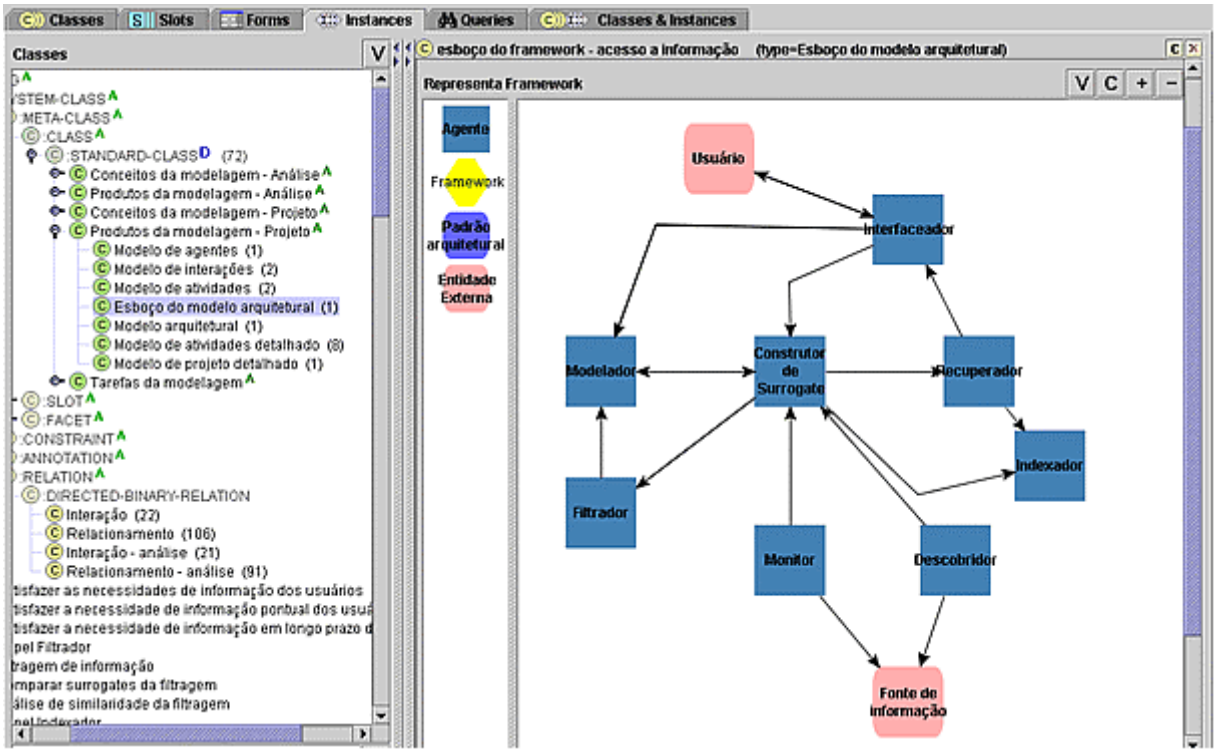

Figura 56. Exemplo de esboço do modelo arquitetural

5.3.4. Seleção de padrão arquitetural

Logo após a construção do esboço, o *framework* deve ser refinado através da seleção e utilização de *padrões arquiteturais*. Estes padrões e seus atributos estão representados na ONTODD, como instâncias da meta-classe *padrão arquitetural* (Figura 57).

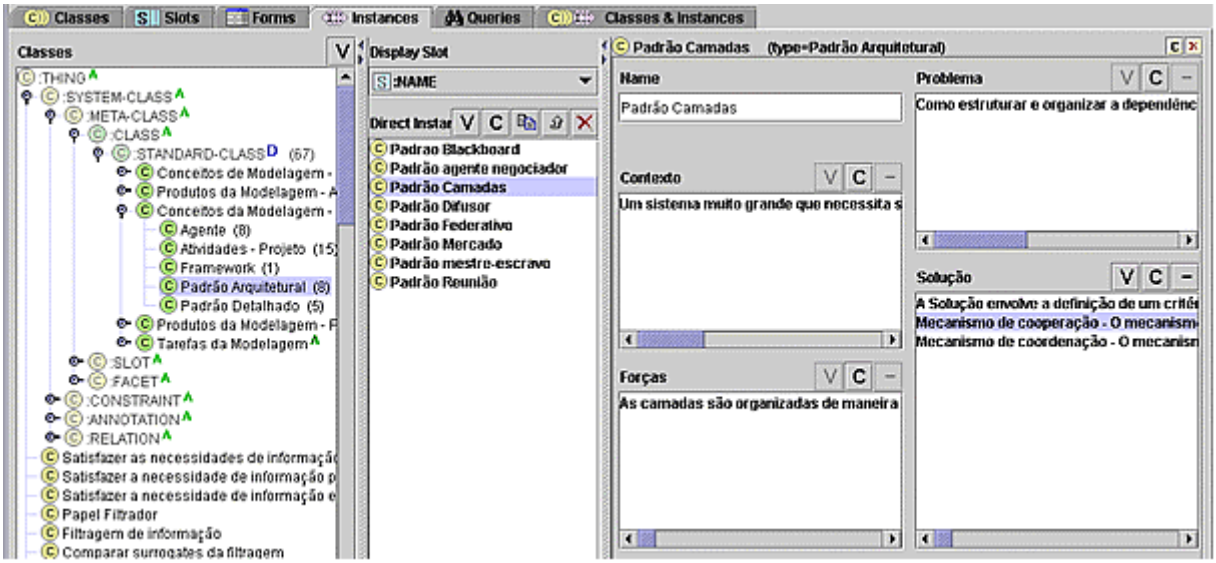

Figura 57. Padrões arquiteturais especificados

O esboço é analisado, buscando identificar características e comportamento determinantes. Estas características servem como base para a seleção de um ou mais padrões.

De posse das características identificadas deve-se verificar as instâncias da meta-classe *padrão arquitetural*, centrando-se nos *slots* referentes ao *problema* tratado pelo padrão e seu *contexto* de aplicação, estes slots poder ser observados na Figura 57.

5.3.5. Refinamento do framework

O *esboço do modelo arquitetural* é refinado e organizado seguindo a solução encontrada no(s) padrão(ões) selecionado. A definição do *modelo arquitetural definitivo* pode ser observada na Figura 58.

Esta tarefa destina-se a construção do *modelo arquitetural definitivo*, que representa o framework final, através da instanciação da meta-classe *modelo arquitetural*. Neste processo de construção o *esboço do modelo arquitetural* é refinado, aplicando-se a solução descrita no(s) padrão(ões) selecionado(s), organizando o modelo segundo mecanismos de coordenação e cooperação.

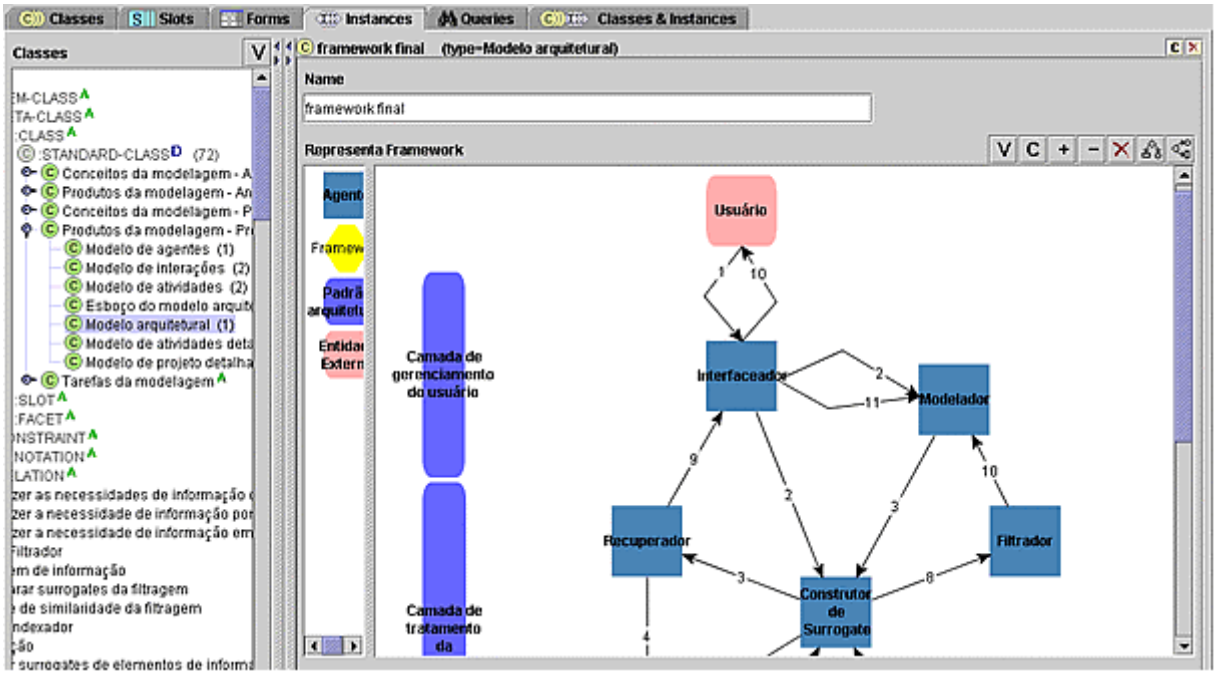

Figura 58. Exemplo de modelo arquitetural final

5.3.6. Detalhamento dos agentes do framework

Após termos definido o framework é realizado o *detalhamento dos agentes* que faz referência à primeira tarefa do projeto detalhado do framework. Através desta tarefa é criado o *modelo de atividades detalhado*.

Primeiramente são construídos *modelos de atividades detalhados* dos agentes, através do refinamento das atividades dos agentes representados nos *modelos de atividades.* Busca-se mostrar o funcionamento interno de cada agente. Temos na Figura 59 um exemplo de *modelo de atividade detalhado* criado para um agente específico.

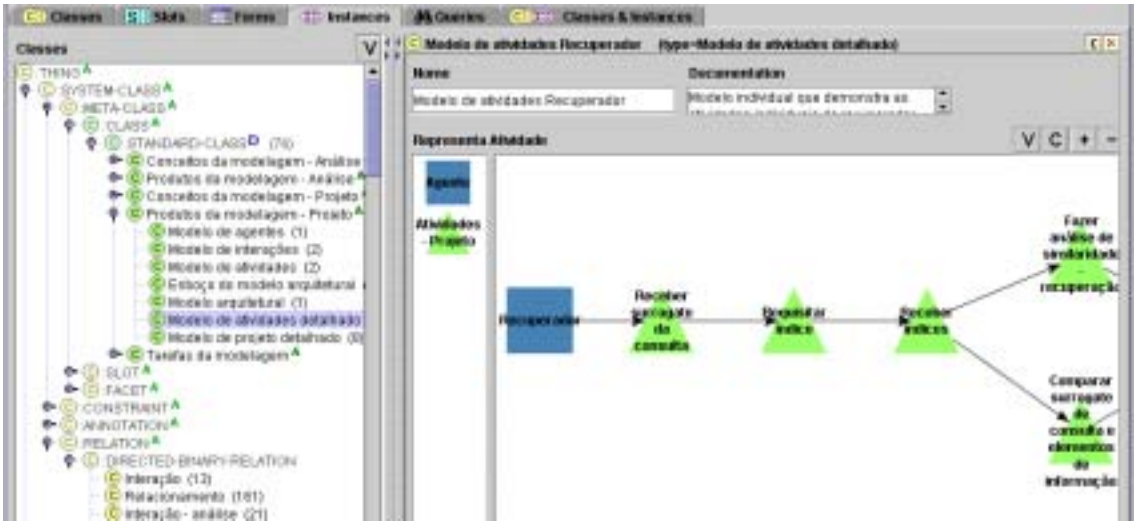

Figura 59. Exemplo de modelo de atividades detalhado

Logo após, é feito o detalhamento dos agentes onde através de uma análise do conhecimento e comportamento de cada agente. Caso surjam novas atividades estas devem ser adicionadas às instâncias dos agentes e aos *modelos de atividades*. Este detalhamento deve ser realizado observando o comportamento dos agentes em relação aos dois tipos básicos de agentes [RUSSEL, 1995]: *deliberativos* e *reativos*.

5.3.7. Seleção de padrão de projeto detalhado

Com os *modelos de atividades detalhados* construídos e descrição do conhecimento e comportamento dos agentes, deve-se agora analisar os *padrões de projeto detalhado* de acordo com os comportamentos dos agentes, centrando-se nos *slots* que descrevem o contexto e problema tratados (Figura 60).

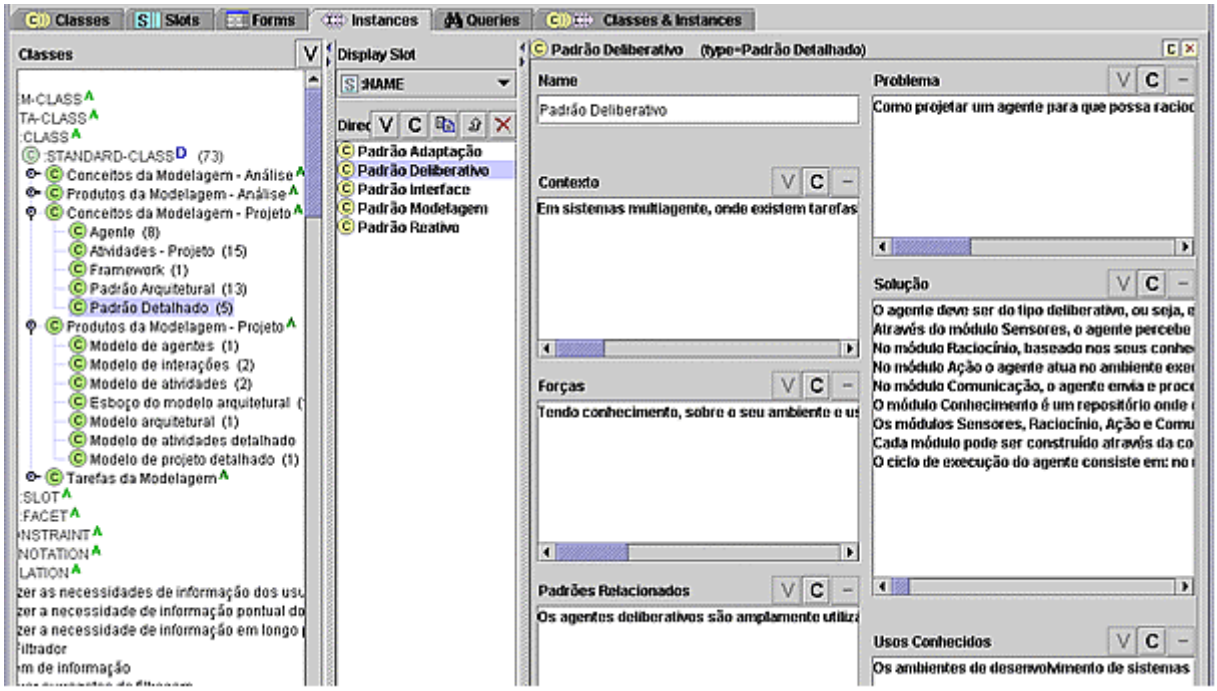

Figura 60. Instâncias da meta-classe *padrão de projeto detalhado*

5.3.8. Refinamento dos agentes

Nesta tarefa é construído o *modelo arquitetural detalhado* mostrando os *agentes* e sua arquitetura, estruturada segundo um ou mais padrões de projeto detalhado selecionados. Deve ser construído um modelo para cada agente, descrevendo seus módulos internos e funcionamento. Cada modelo conterá a descrição do agente, o padrão seguido e os módulos internos dos agentes com suas operações e conhecimento.

A Figura 61 temos ilustra o *modelo arquitetural detalhado* do agente *Interfaceador* sendo construído. Aplicando o padrão selecionado, o agente é estruturado em módulos, que foram instanciados da meta-classe *Módulo*. Cada módulo possui atividades, operações e conhecimento representados em instâncias das meta-classes *Comportamento* e *Conhecimento*.

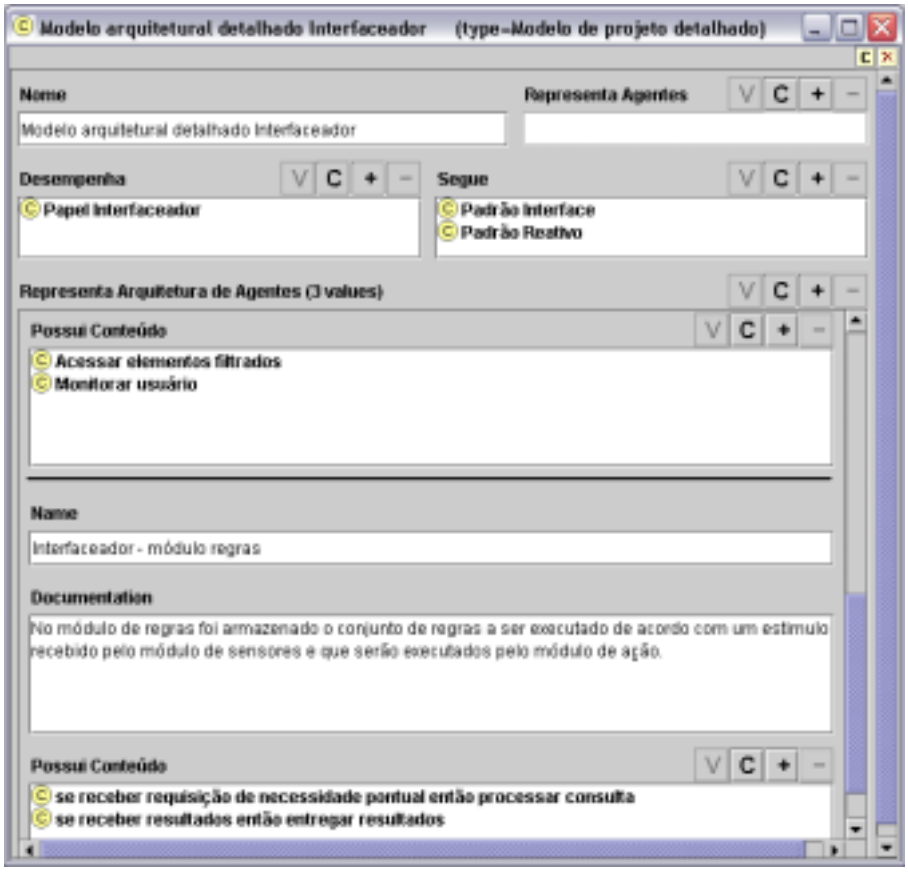

Figura 61. Exemplo de modelo arquitetural detalhado do agente Interfaceador

5.4. Resumo das tarefas e atividades a serem realizadas na instanciação da ONTODD

Abaixo estão listadas as tarefas e atividades a serem realizadas na construção de frameworks multiagente usando a técnica DDEMAS através da instanciação da ONTODD.

1. Criar modelo de agentes a partir do modelo de papéis do modelo de domínio;
2. Definir as atividades de cada agente a partir do papel desempenhado, no *slot Realiza*;
3. Mapear modelos de interações entre papeis em modelos de interações entre agentes;
4. Criar modelos de atividades para cada modelo de interações;
5. Construir esboço do modelo arquitetural, baseado nos modelos de agentes, interações e atividades;

6. Selecionar um ou mais padrões arquiteturais para o refinamento do esboço, de acordo com características e comportamentos observados;

7. Refinar o framework através da aplicação dos padrões, organizando segundo mecanismos de coordenação e cooperação;

8. Realizar o detalhamento dos agentes do framework, definindo modelos de atividades detalhados;

9. Descrever os agentes segundo os dois tipos básicos de agentes: deliberativo e reativo;

10. Selecionar um ou mais padrões de projeto detalhado para cada agente do framework de acordo com a descrição feita e com as informações contidas no padrão;

11. Refinar os agentes através da aplicação dos padrões e construção dos modelos de projeto detalhado.

5.5. Considerações finais

Neste capítulo foi apresentada a ferramenta ONTODD, uma ontologia genérica que representa o conhecimento da técnica DDEMAS para o projeto de uma família de aplicações segundo o paradigma computacional de agentes.

A ONTODD foi construída para guiar o processo de aplicação da técnica DDEMAS na construção de frameworks multiagentes. Os produtos a serem criados são instancias da meta-classe *Produtos da modelagem*, que, no final, irão compor a solução multiagente reutilizável.

6. Estudo de caso – Construção de um framework para o acesso à informação

Este estudo de caso tem como objetivo avaliar a técnica DDEMAS e a ferramenta ONTODD através da construção da ONTODD_INFO, um framework baseado em ontologias para o acesso à informação.

O estudo de caso desenvolve uma solução reutilizável para o problema do acesso à informação, em particular para os processos de recuperação e filtragem de informação. É criado um framework para o acesso a informação através da utilização da ONTODD. O framework é especificado em uma instância da ONTODD chamada de ONTODD_INFO. Os conceitos relativos à fase de análise utilizados na construção deste framework estão descritos na ONTOINFO [SERRA, 2003], um *modelo de domínio* baseado em ontologias para a área do acesso à informação.

6.1. O problema do acesso à informação

O acesso à informação [SALTON, 1983] visa prover mecanismos para que seja possível localizar e recuperar informações relevantes aos usuários da forma mais eficiente possível. Os principais conceitos e processos envolvidos no acesso à informação são descritos a seguir.

A princípio, tem-se uma necessidade de informação que deve ser expressa em uma consulta ou perfil de usuário. Uma consulta caracteriza uma necessidade de informação pontual. Um perfil de usuário caracteriza uma necessidade de informação em longo prazo. Se a necessidade de informação for pontual, o sistema deve acessar uma base indexada e recuperar os elementos de informação relevantes após uma avaliação de similaridade. Se a necessidade de informação for em longo prazo, o sistema deve fazer a filtragem, que consiste em estar continuamente monitorando fontes de informação a procura de elementos de informação que possam ser relevantes a seus usuários representados através de modelos de usuário.

Alguns conceitos se destacam no âmbito do acesso à informação, sendo os principais:

- Necessidade de informação: é uma carência de informação por parte de um ou mais usuários. A necessidade de informação pode ser pontual, ou seja, uma

necessidade a ser satisfeita imediatamente, ou em longo prazo, isto é, uma necessidade que se prolonga por um certo tempo.

- Elemento de informação: é um item que contém informação. Os elementos de informação podem ser dos seguintes tipos: documentos textuais e hipermídia, vídeo, som, imagem.
- Fonte de informação: é um repositório de elementos de informação. As fontes de informação podem ser estruturadas ou não estruturadas, podem ainda ser dinâmicas ou estáticas.Um exemplo de uma fonte de informação dinâmica e não estruturada é a Web.

Outros conceitos existentes no acesso à informação são:

- Modelo de usuário: é uma representação do usuário a partir da qual o sistema filtra elementos de informação que possam ser relevantes. O modelo de usuário pode ser capturado de forma implícita ou explicita.
- Consulta: é a representação da necessidade de informação do usuário expressa utilizando uma linguagem de especificação de consulta.
- Surrogate: é a forma como as consultas, os elementos de informação e os modelos de usuários são representados internamente no sistema.
- Filtragem: é a modalidade de acesso à informação utilizada no caso de uma necessidade de informação em longo prazo.
- Recuperação: é a modalidade de acesso à informação utilizada no caso de uma necessidade de informação pontual.
- Indexação: é o processo de criação de um índice invertido com os surrogates dos elementos de informação para agilizar o processo de comparação de surrogates.
- Comparação de surrogate: compara as representações internas dos itens de informação com a representação interna da consulta ou perfil do usuário e para selecionar uma lista de itens de informação que casam parcial ou totalmente com a consulta.
- Análise de similaridade: executa uma medida de similaridade ou determina o valor de similaridade que avalia quando um item de informação satisfaz uma necessidade de informação.

6.2. ONTOINFO - um modelo de domínio para a área do acesso à informação

A ONTOINFO é o resultado da aplicação da técnica GRAMO [FARIAb, 2003] à área do acesso a informação, contendo a especificação dos requisitos genéricos de sistemas para o acesso a informação.

O *modelo de conceitos* (Figura 62) representa os conceitos relevantes da área de acesso a informação e seus relacionamentos existentes entre eles. No *modelo de objetivos* (Figura 63) estão mostradas os objetivos gerais e específicos. Os objetivos específicos estão vinculados ao objetivo geral. Existem ainda neste modelo responsabilidades que estão vinculadas aos objetivos específicos.

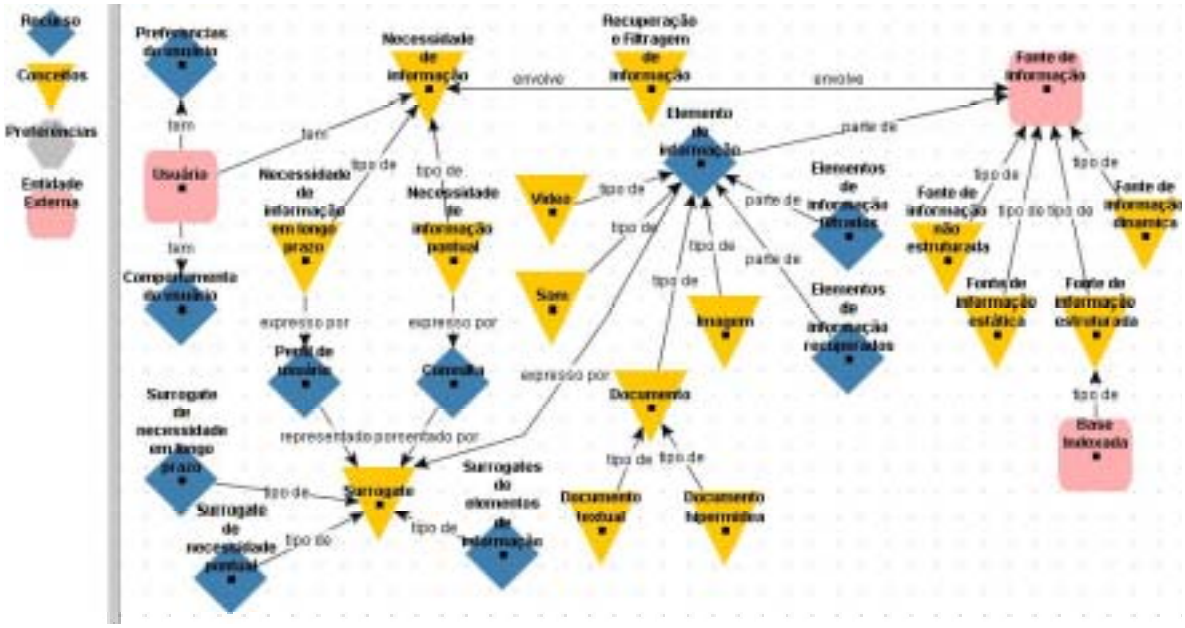

Figura 62. Modelo de conceitos da ONTOINFO

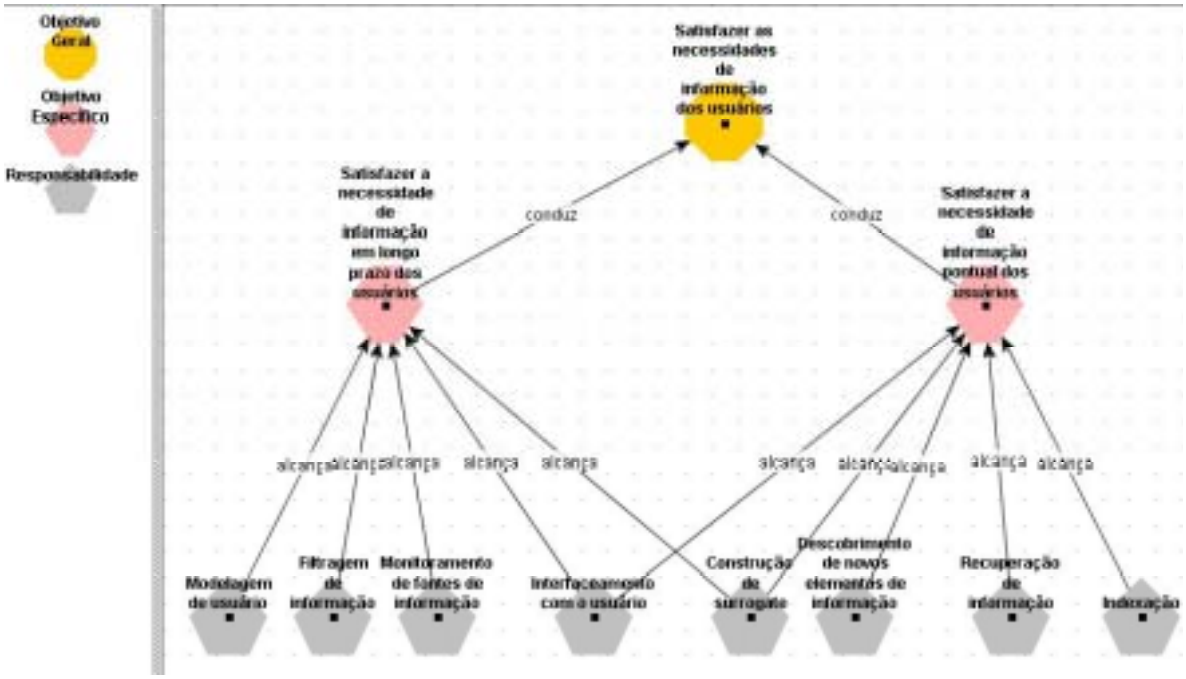

Figura 63. Modelo de objetivos da ONTOINFO

No *modelo de papéis* (Figura 64) estão representados os papéis do sistema. Os papéis estão associados às responsabilidades. O *modelo de interações entre papéis* (Figura 65) representa as interações entre papéis ou entre papéis e entidades externas envolvidas no alcance de um objetivo específico. É construído um *modelo de interações entre papéis* para cada objetivo específico especificado no *modelo de objetivos.*

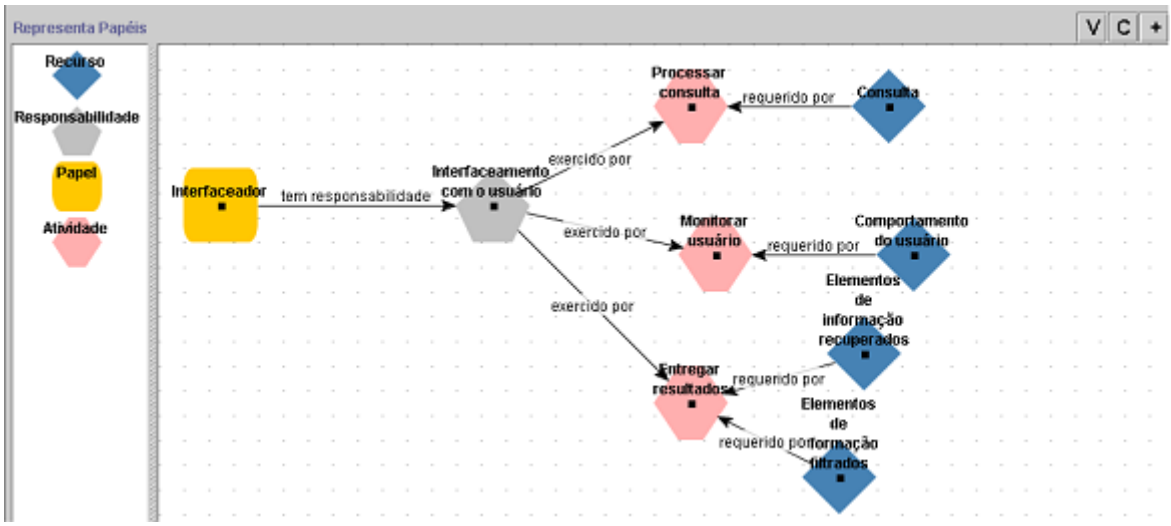

Figura 64. Exemplo de modelo de papéis da ONTOINFO

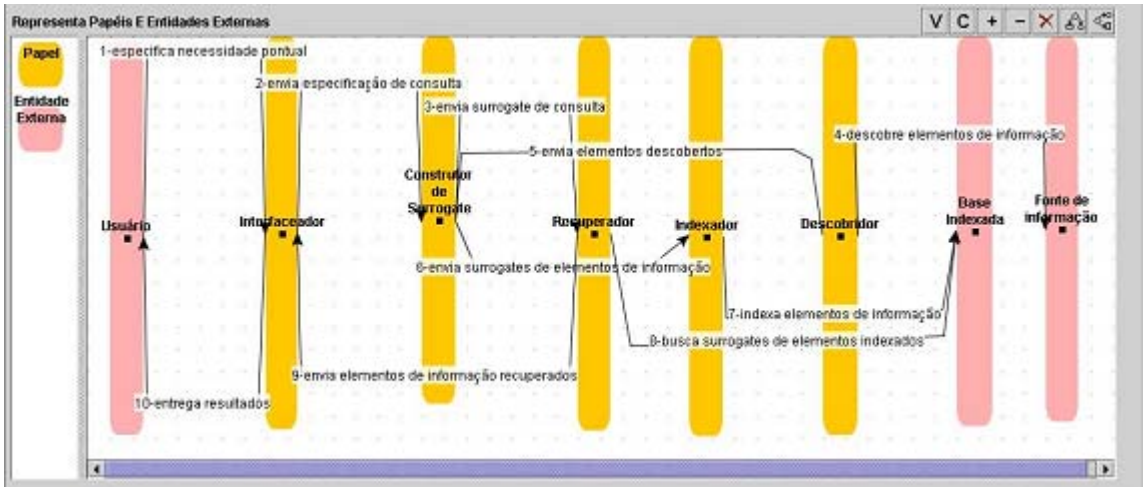

Figura 65. Modelo de interações entre papéis da ONTOINFO

6.3. Construção do framework multiagente – ONTODD_INFO

A seguir é apresentado o processo de instanciação da ONTODD para a construção do framework ONTODD_INFO, de acordo as especificações do modelo de domínio ONTOINFO (Figura 66), seguindo as tarefas e atividades da DDEMAS resumidas na seção 5.4.

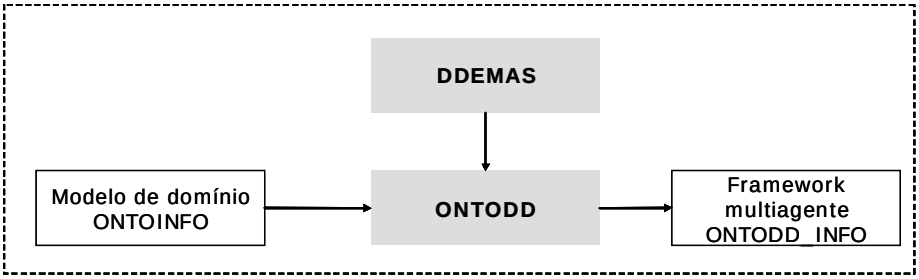

Figura 66. Construção da ONTODD_INFO

6.3.1. Modelagem de agentes

Tarefa 1: Criar modelo de agentes a partir do modelo de papéis do modelo de domínio.

Nesta tarefa foi criada uma instância da meta-classe *modelo de agentes* onde os papéis do *modelo de papéis* na ONTOINFO foram mapeados em agentes. Foram criadas neste modelo 8 instâncias da meta-classe *agente* que desempenham os 8 papéis descritos no *modelo de papéis*. Ao agente *Recuperador* foi atribuído o papel de *Recuperador*. Ao agente *Modelador* foi atribuído o papel de *Modelador*. Ao agente *Interfaceador* foi atribuído o papel de *Interfaceador* e assim por diante (Figura 67).

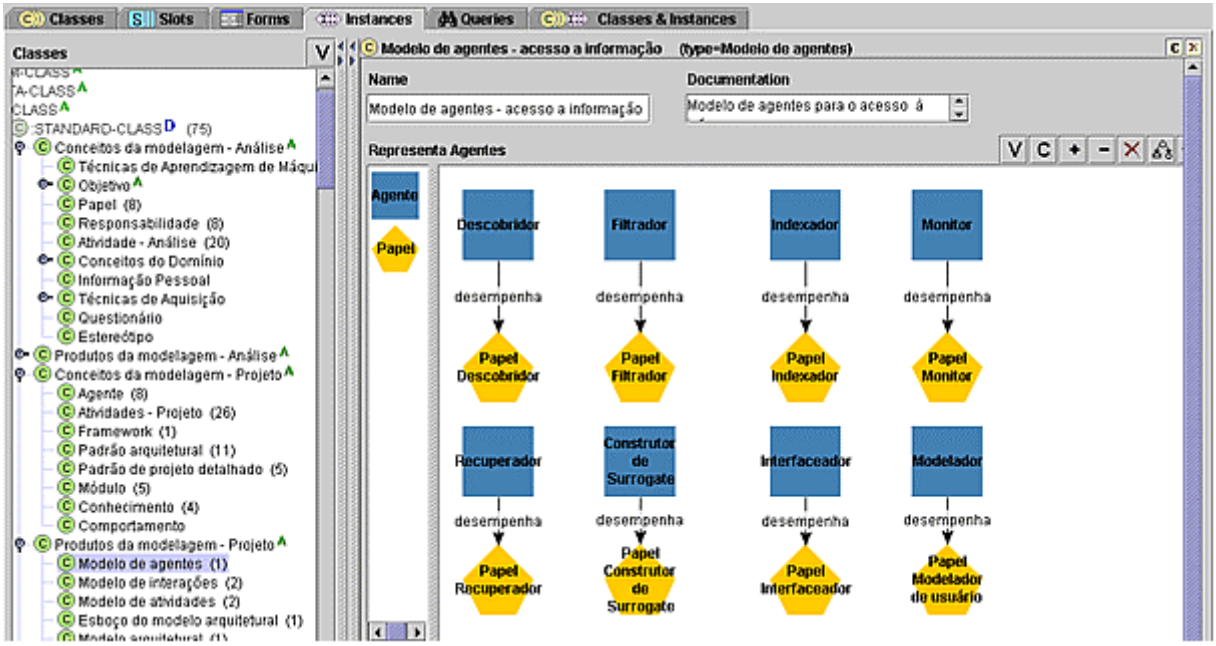

Figura 67. Modelo de agentes

Tarefa 2: Definir as atividades de cada agente a partir do papel desempenhado, nos slots Realiza.

As atividades dos agentes foram derivadas a partir dos papéis que eles desempenham (Figura 68). As atividades foram representadas no *slot Realiza* de cada agente.

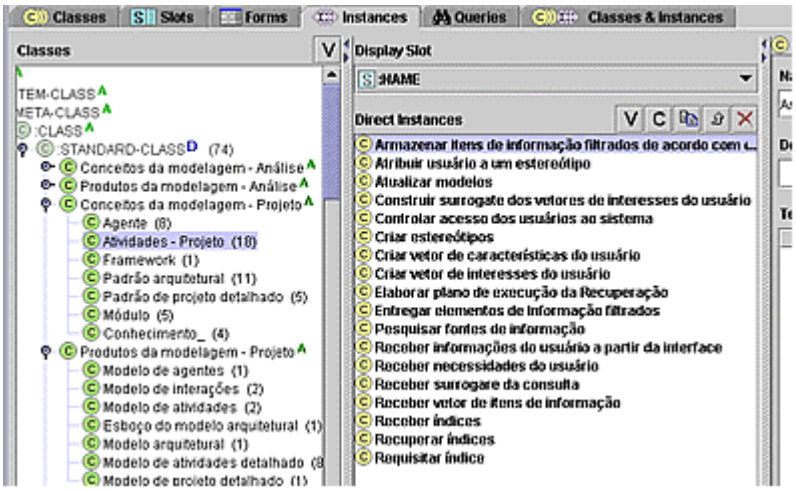

Figura 68. Instâncias da meta-classe A*tividade – Projeto* na ONTODD_INFO

A Figura 69 mostra o resultado desta etapa na representação do agente *Construtor de surrogate* onde estão especificadas no *slot Realiza* as atividades realizadas pelo agente.

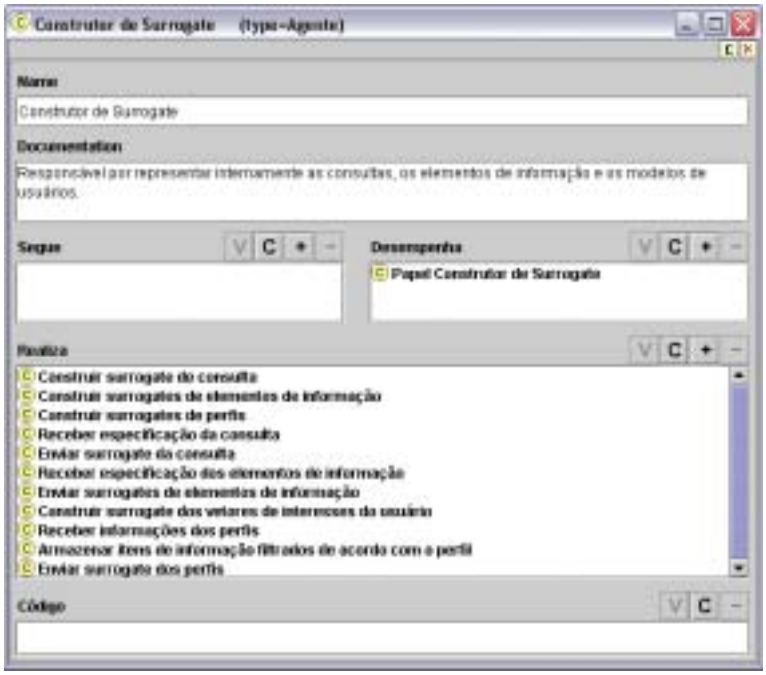

Figura 69. Instância do agente Construtor de surrogate

A Figura 70 mostra as atividades realizadas pelo agente *Descobridor*.

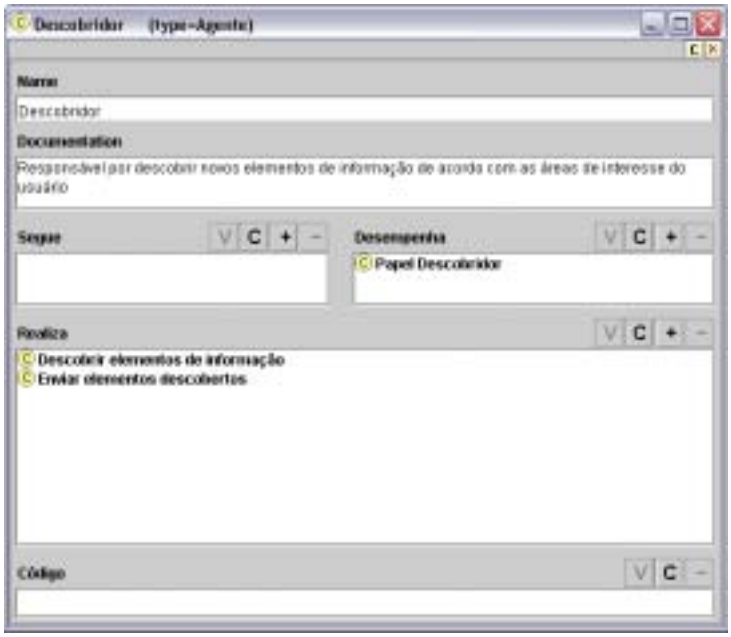

Figura 70. Instância do agente *Descobridor*

A Figura 71 mostra as atividades realizadas pelo agente *Filtrador*.

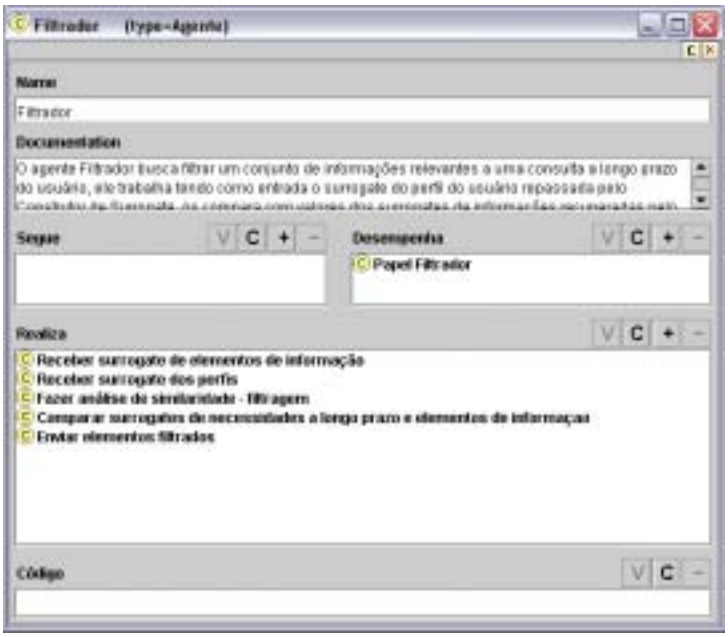

Figura 71. Instância do agente Filtrador

A Figura 72 mostra as atividades realizadas pelo agente *Indexador*.

Figura 72. Instância do agente Indexador

A Figura 73 mostra as atividades realizadas pelo agente *Interfaceador*.

Figura 73. Instância do agente Interfaceador

A Figura 74 mostra as atividades realizadas pelo agente *Modelador*.

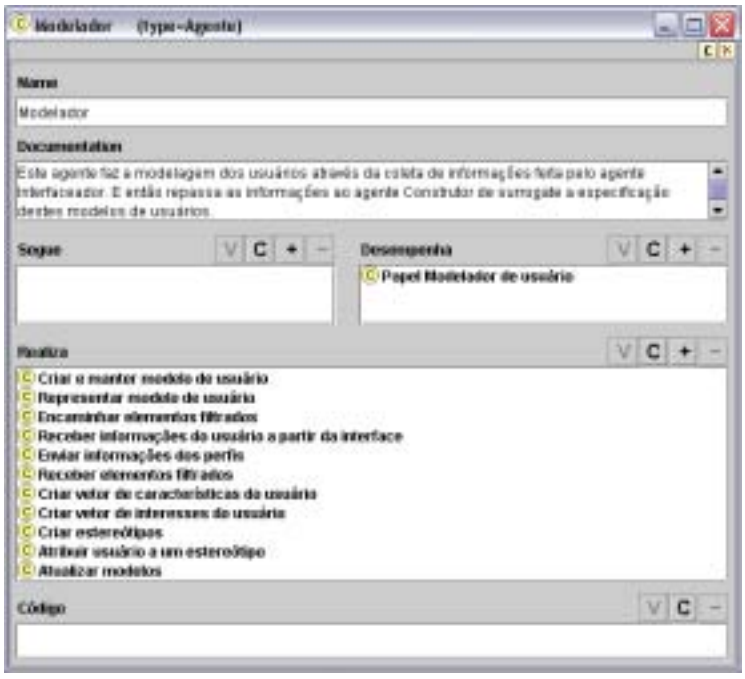

Figura 74. Instância do agente Modelador

A Figura 75 mostra as atividades realizadas pelo agente *Monitor*.

Figura 75. Instância do agente Monitor

A Figura 76 mostra as atividades realizadas pelo agente *Recuperador*.

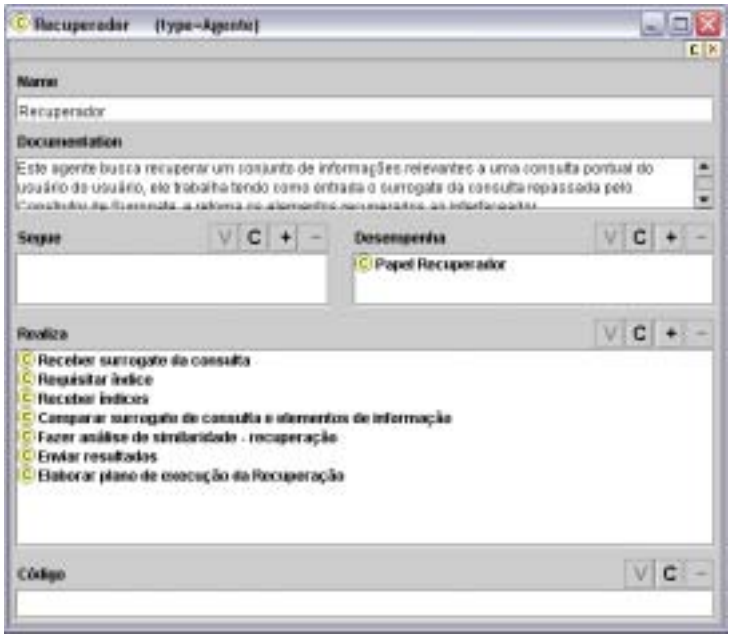

Figura 76. Instância do agente Recuperador

6.3.2. Modelagem de interações e atividade

Tarefa 3: Mapear modelos de interações entre papeis em modelos de interações entre agentes.

Foram analisados os *modelos* de *interações entre papéis e entidades externas* (Figura 65) da ONTOINFO. Estes modelos foram mapeados em *modelos de interações entre agentes*. As interações entre os agentes foram derivadas diretamente das interações entre os papéis.

Os *modelos de interações entre agentes* foram criados para os objetivos específicos *satisfazer a necessidade pontual dos usuários* (Figura 77) e *satisfazer a necessidade contínua dos usuários*(Figura 78). O primeiro *modelo de interações* (Figura 77) mostra o *usuário* fazendo uma requisição, baseada em uma necessidade pontual, ao agente *Interfaceador*, que envia a especificação desta consulta ao *Construtor de surrogate*, este por sua vez envia o surrogate da consulta ao *Recuperador*, que requisita índices ao *Indexador*, índices estes criados a partir das páginas recuperadas pelo agente *Descobridor*. O *Recuperador* de posse dos índices e do surrogate da consulta, realiza o processo de recuperação. Daí então envia os itens de informação recuperados ao *Interfaceador*, a fim de que este os enviem ao usuário.

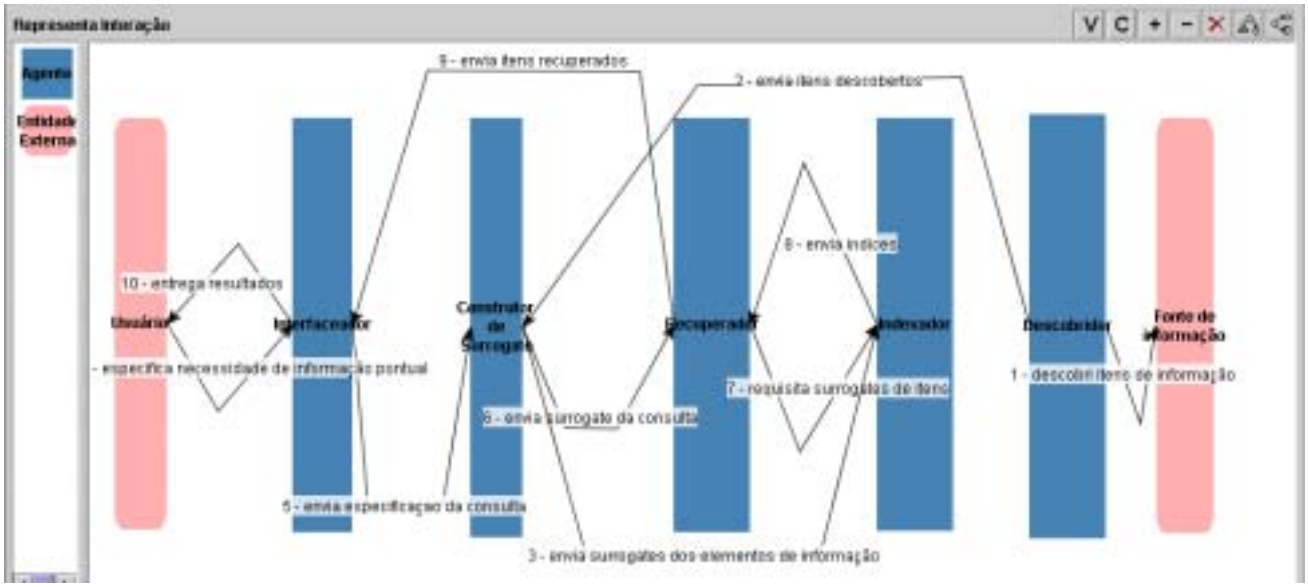

Figura 77. Modelo de interações para o objetivo *satisfazer a necessidade pontual dos usuários*

O segundo *modelo de interações* (Figura 78) mostra o *usuário* fazendo uma requisição baseada em uma necessidade a longo prazo ao agente *Interfaceador*, que envia as informações do usuário ao *Modelador*, este gera um perfil para o usuário e o envia ao *Construtor de surrogate* para representação do mesmo, finalizando esta primeira etapa com o *Construtor de surrogate* interagindo com o *Modelador* com o envio do surrogate dos perfis e requisição de armazenamento do mesmo. A segunda etapa inicia com o agente *Monitor* que monitora as fontes de informação e ao perceber mudanças envia os elementos de informação monitorados ao *Construtor de surrogate* para representação, este envia os surrogates das necessidades do perfil juntamente com os surrogates dos elementos monitorados ao *Filtrador* para a realização do processo de filtragem. Após a realização deste processo o *Filtrador* envia os elementos de informação filtrados ao *Modelador* para este atualizar os modelos. Posteriormente o agente *Interfaceador* irá acessar os elementos de informação filtrados e armazenados no modelo do usuário, finalizando com a entrega destes resultados ao usuário.

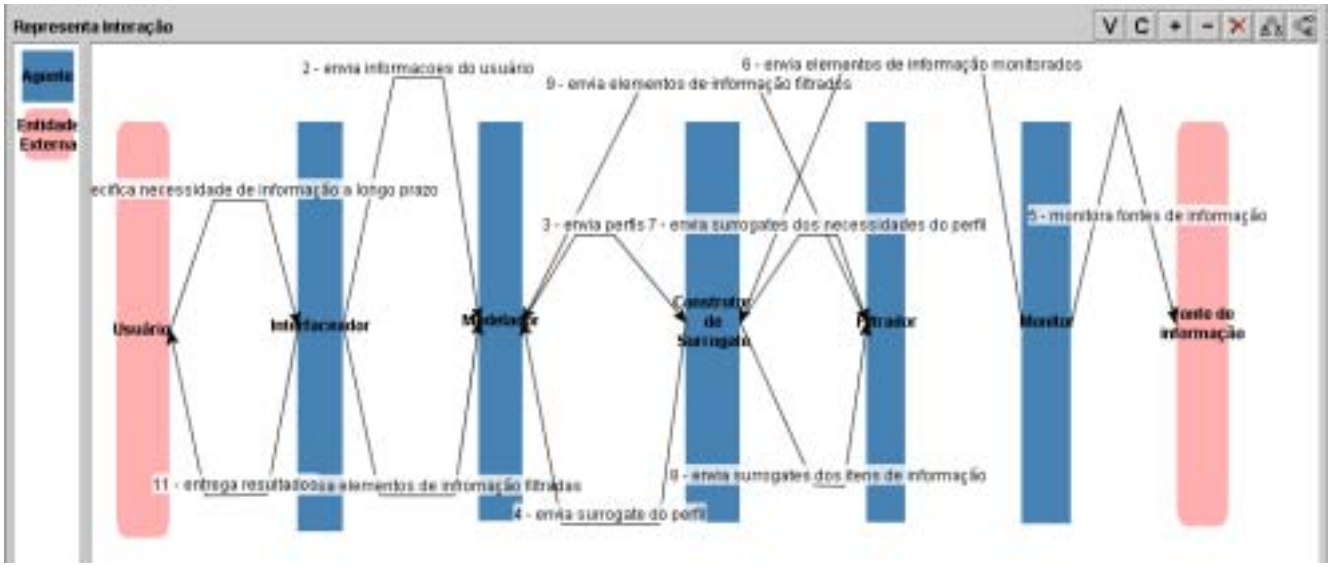

Figura 78. Modelo de interações para o objetivo *satisfazer a necessidade a longo prazo dos usuários*

Tarefa 4: Criar modelos de atividades para cada modelo de interações.

Depois de modelados os agentes e suas interações, foi realizada a construção dos *modelos de atividades* para cada *modelo de interações*. As atividades utilizadas neste modelo foram obtidas das instancias das meta-classes *Agente* e *Atividade-Projeto*.

A Figura 79 mostra o *modelo de atividades* referente ao objetivo de *satisfazer necessidade pontual do usuário*, o processo inicia com uma requisição do usuário ao

Interfaceador, a atividade de *receber necessidades do usuário* é iniciada seguida de *processar atividade* e *enviar da especificação da consulta* ao *Construtor de surrogates*, são executadas por este agente as atividades *receber especificação da consulta* e *construir surrogate de consulta*, finalizando sua atuação com *enviar surrogate da consulta* ao agente *Recuperador*, que ativa *requisitar índice* ao agente *Indexador*, que recebe esta requisição, *recupera índices e os envia* de para o *Recuperador*, o qual realiza o processo de Recuperação através das atividades *comparar surrogates da consulta e elementos de informação* e *fazer análise de similaridade*, finalizando com as atividades *enviar resultados* e *entregar resultados*, realizada pelo *Interfaceador*

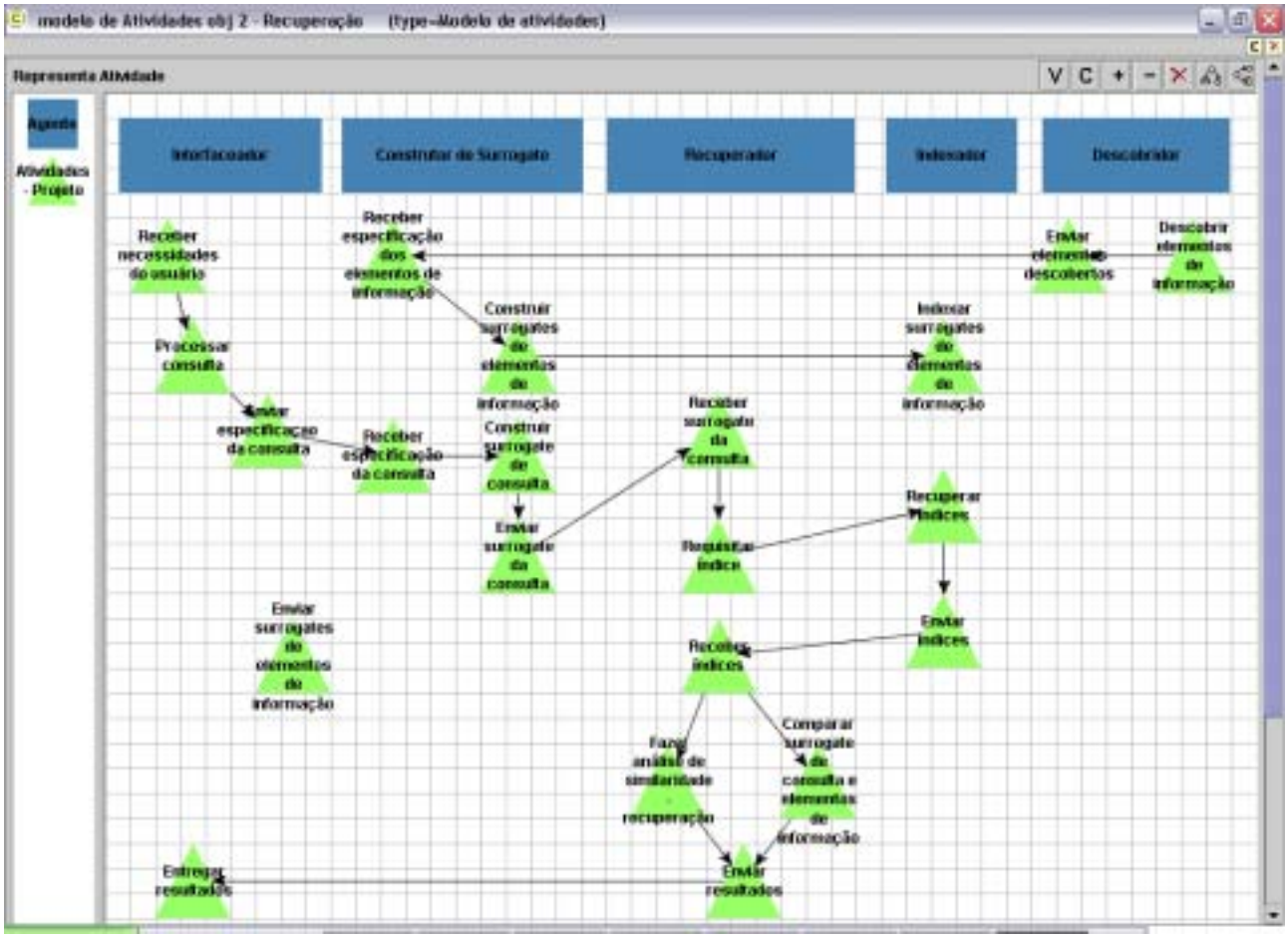

Figura 79. Modelo de atividades para o objetivo *satisfazer necessidade pontual do usuário*

A Figura 80 mostra o *modelo de atividades* referente ao objetivo de *satisfazer necessidade a longo prazo do usuário*, após o início do processo, é realizada pelo agente *Interfaceador* a atividade *monitorar usuário*, que após realizada ativa *enviar informações do usuário*, seguida por *receber informações do usuário a partir da interface* e *criar e manter modelos*, realizadas pelo agente *Modelador*, que então realiza *enviar especificação dos perfis*

comunicando-se com o agente *Construtor de surrogates* que se encarrega de *receber informações dos perfis, construir surrogates de perfis* e *enviar surrogate dos perfis* de volta ao agente *Modelador*.

A segunda etapa no processo referente a este objetivo se dá quando o agente *Monitor* realiza a atividade de *detectar mudanças em fontes de informação e enviar mudanças detectadas*, é iniciado o processo de Filtragem, com as atividades de *receber especificação dos elementos de informação* e *construir surrogates de elementos de informação* realizadas pelo agente *Construtor de surrogate*, que de posse dos resultados passa a atividade de *enviar surrogates de elementos de informação* ao *Filtrador*, que em seguida realiza *receber vetor de itens de informação, comparar surrogates* e *fazer análise de similaridade*, buscando encontrar itens de interesse dos usuários. Ao fim destas duas últimas atividades é realizada *entregar elementos de informação filtrados*, é então feita a *atualização dos modelos* dos usuários pelo agente *Modelador*.

A terceira etapa é executada quando a atividade de *monitorar o usuário* pelo *Interfaceador* inicia a atividade *acessar elementos filtrados*, os elementos são retornados pelo *Modelador* através de *enviar elementos filtrados*, finalizando o processo com a atividade *entregar resultados*, a ser realizada pelo agente *Interfaceador*.

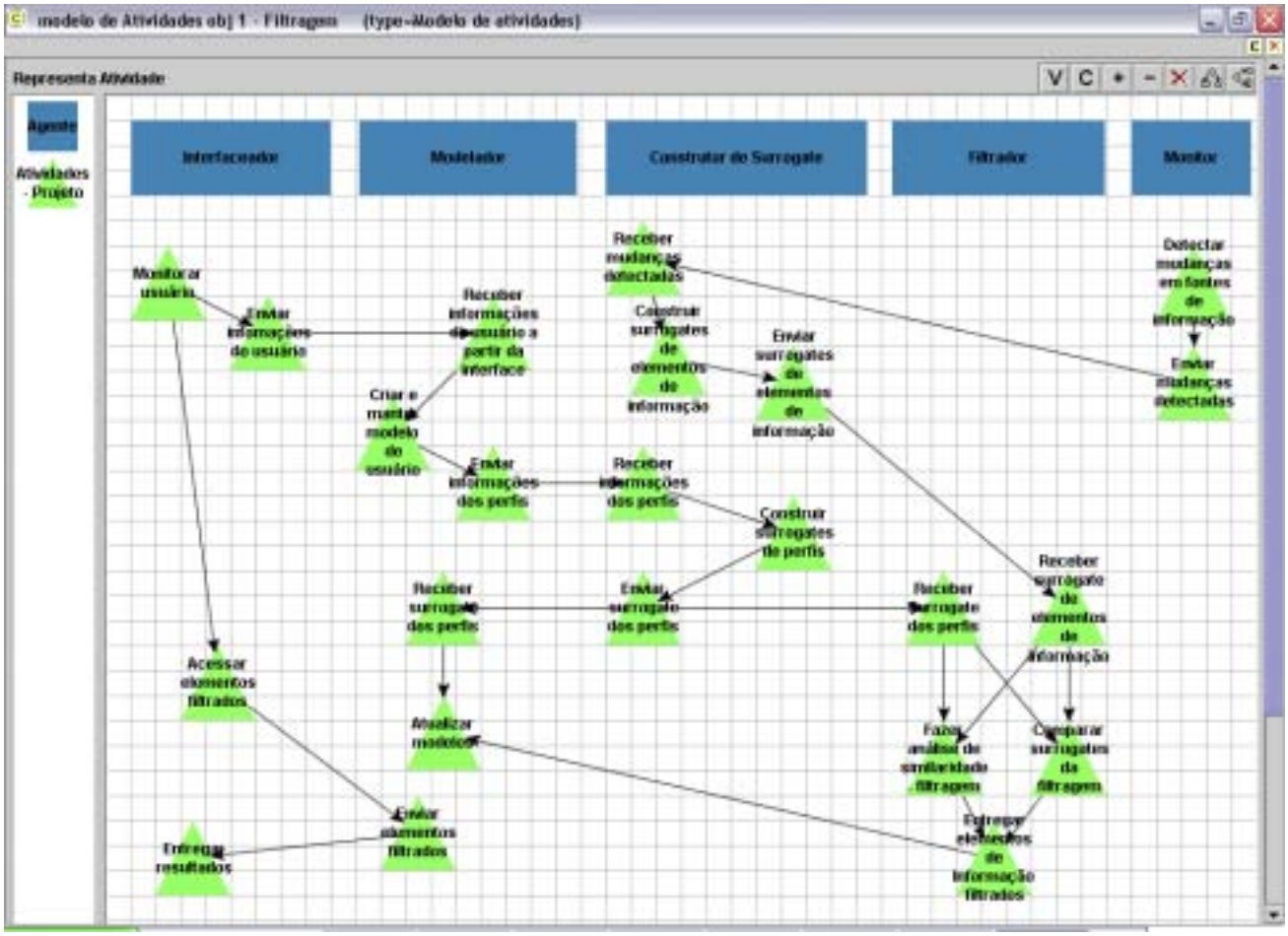

Figura 80. Modelo de atividades para o objetivo *satisfazer necessidade a longo prazo do usuário*

6.3.3. Construção do esboço do framework

Tarefa 5: Construir esboço do modelo arquitetural, baseado nos modelos de agentes, interações e atividades.

A tarefa de *construção do esboço do framework* foi realizada a partir do *modelo de agentes* e dos *modelos de interações*.

Com as informações coletadas o esboço foi criado e é ilustrado na Figura 81.

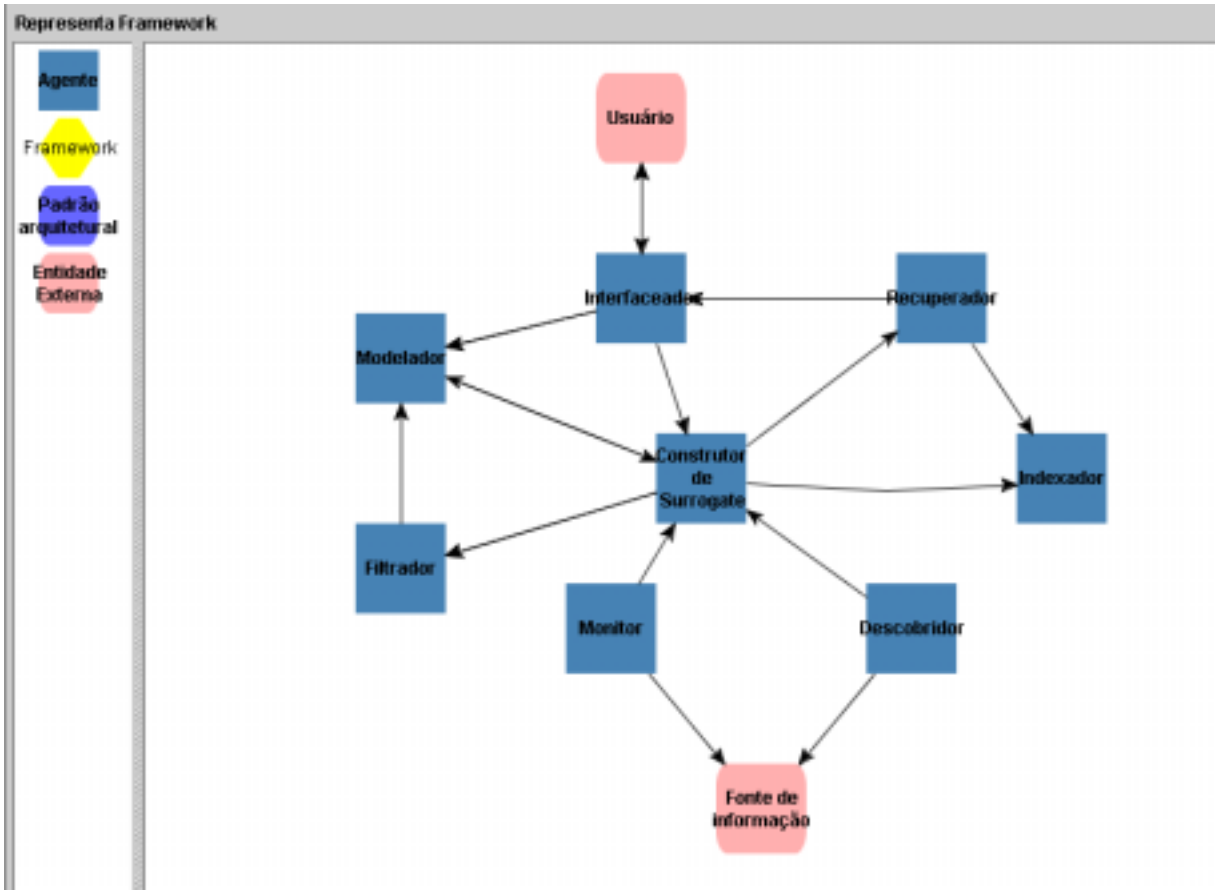

Figura 81. Esboço do modelo arquitetural

6.3.4. Seleção de padrão arquitetural

Tarefa 6: Selecionar um ou mais padrões arquiteturais para o refinamento do esboço, de acordo com características e comportamentos observados.

Logo após a construção do esboço, ele foi analisado em conjunto com os modelos criados anteriormente. Durante a análise verificou-se conjuntos de agentes com responsabilidades afins, de dados e tarefas similares, possibilitando um agrupamento de

agentes baseado em responsabilidades, dados e atividades comuns, como os agentes *Recuperador, Filtrador e Construtor de Surrogate* que possuem responsabilidades ligadas ao tratamento das informações inseridas no processo de recuperação e filtragem.

Ainda foi verificado através das interações existentes um conjunto de tarefas integradas estabelecendo uma relação de requisitante e requisitado, com agentes delegando e coordenando atividades, como por exemplo um agente *Interfaceador* ao receber uma requisição do usuário delega atividades ao agente C*onstrutor de surrogate*.

Após a verificação das instâncias da meta-classe *padrão arquitetural*, centrada nos *slots* referentes ao *problema* tratado pelo padrão e seu *contexto* de aplicação, os padrões *Camadas* e *Mestre-escravo* foram selecionados. O primeiro por prover uma solução onde as camadas dividem os agentes de acordo com suas responsabilidades e o segundo por prover uma solução de delegação de tarefas e coordenação de execução.

6.3.5. Refinamento do Framework

Tarefa 7: Refinar o framework através da aplicação dos padrões, organizando segundo mecanismos de coordenação e cooperação.

Com o *esboço do modelo arquitetural* definido e os padrões arquiteturais selecionados, o esboço foi organizado seguindo a solução descrita nos padrões.

A solução do primeiro padrão é a seguinte [SILVA, 2003]: a definição de um critério de abstração para a divisão das responsabilidades entre as camadas. Em seguida, determina-se o número de níveis de abstração, dá-se um nome as camadas e associa-se o serviço de cada uma delas.

A solução foi então aplicada (Figura 82), onde a divisão em camadas ficou definida da seguinte forma:

- *Camada de gerenciamento com usuário*, camada que faz a intermediação entre o sistema e os usuários, incluindo a captura de necessidades de informação do usuário, monitora os usuários, entrega de resultados, exibe atualizações de informação de interesse do usuário e modela informações referentes aos interesses do usuário. Esta camada é composta pelos agentes *Interfaceador* e *Modelador*, ela recebe do usuário as informações referentes às suas necessidades, informações que identificam perfil, trata as informações

recolhidas dos usuários para a montagem de necessidades a longo prazo e o perfil destes e envia os resultados recuperados e filtrados.

- *Camada de tratamento da informação*, camada que engloba o tratamento de informações para a filtragem, recuperação, representação interna das informações e indexação. Esta camada é composta pelos agentes *Construtor de Surrogate*, *Recuperador*, *Filtrador* e *Indexador*, ela recebe da Camada de gerenciamento com usuários as informações referentes aos interesses do usuário e devolve para esta camada os elementos filtrados e recuperados. Ela também recebe da Camada de interação com as fontes de informação as páginas descobertas para a criação dos índices.
- *Camada de interação com as fontes de informação*, camada que faz a ligação do sistema com as fontes de informação da Web, capturando páginas utilizadas para construção do índice e monitorando as mudanças das páginas da Web. Esta camada é composta pelo agente *Monitor* e *Descobridor*, ela possui um conjunto de URLs a serem monitoradas e envia para a camada acima os resultados das buscas. Ela ainda captura da Web páginas para a construção do índice e o conteúdo de cada página monitorada.

Existe ainda uma "*camada externa*" (Web) é nela onde estão localizadas as fontes de informações, as quais são utilizadas como base de dados. As fontes são utilizadas tanto para o processo de recuperação quanto para o de filtragem.

O mecanismo de cooperação no padrão *Camada* é evidenciado através da relação de dependência entre as camadas, ou seja, uma camada requisita tarefas e a outra fornece serviços à camada que os solicitou.

O mecanismo de coordenação foi aplicado utilizando-se o padrão *mestre-escravo*, os agentes *Interfaceador*, *Modelador*, *Monitor* e *Descobridor* servem como mestres fazendo requisições e delegações aos agentes *Construtor de Surrogate*, *Recuperador*, *Filtrador e Indexador*.

Após a aplicação dos padrões, o *modelo arquitetural definitivo* foi definido (Figura 82), seguindo os critérios de organização, cooperação e coordenação, aspectos que não estavam bem definidos no momento da construção do *modelo arquitetural* referente ao esboço do framework. Os fluxos de execução são demonstrados no modelo, as numerações

repetidas ocorrem devido as duas maneiras de realização de um ciclo, filtragem ou recuperação.

O processo é realizado da seguinte forma: Temos uma comunicação direta do usuário com o agente *Interfaceador*, este por sua vez de acordo com as requisições do usuário inicia um processo de comunicação com os agentes *Modelador*, *Construtor de surrogate* e *Recuperador*, realizando um conjunto de requisições e recebimento de resultados. Após estas interações é iniciado um procedimento que pode ser de Recuperação ou Filtragem de informações.

Caso seja de Recuperação, identificada pelas linhas pontilhadas, primeiramente ocorre as seguintes interações e atividades, o agente *Descobridor* realiza o processo de descoberta de páginas (fluxo 1), estas são representadas internamente (fluxo 2) pelo *Construtor de surrogate*, são indexadas (fluxo 3) e armazenadas pelo *Indexador*. Após isso realizado, podemos analisar o segundo conjunto de interações no processo de recuperação. O agente *Construtor de surrogate* recebe a requisição do *Interfaceador* (fluxo 4), e interage (fluxo 5) com o *Recuperador*, este por sua vez interage com *Indexador* (fluxo 6), recebendo deste os índices (fluxo 7) para a realização do processo de recuperação, finalizando com a devolução (fluxo 8) dos resultados ao *Interfaceador* e entrega destes ao usuário (fluxo 10).

Por outro lado, num processo de Filtragem, identificado pelas linhas sólidas, ao ocorrer uma interação (fluxo 1) do usuário com *Interfaceador*, este agente interage (fluxo 2) com o *Modelador* que se comunica (fluxo 3) com o *Construtor de surrogate*, para armazenar as informações dos usuários (fluxo 4), será dado início a uma tarefa continua executada pelo *Monitor* monitorando fontes de informação (fluxo 5), que repassa (fluxo 6) novidades ao *Construtor de surrogate*, o qual entrega (fluxo 7) ao *Filtrador* os surrogates destas novidades, que juntamente com o envio (fluxo 8) do surrogate do perfil realiza a filtragem. Caso seja selecionado algo de interesse dos usuários o *Filtrador* os envia (fluxo 9) ao *Modelador* para atualização dos modelos, que poderão ser posteriormente acessados (fluxo 10) pelo agente *Interfaceador* para entregar (fluxo 11) ao usuário os resultados.

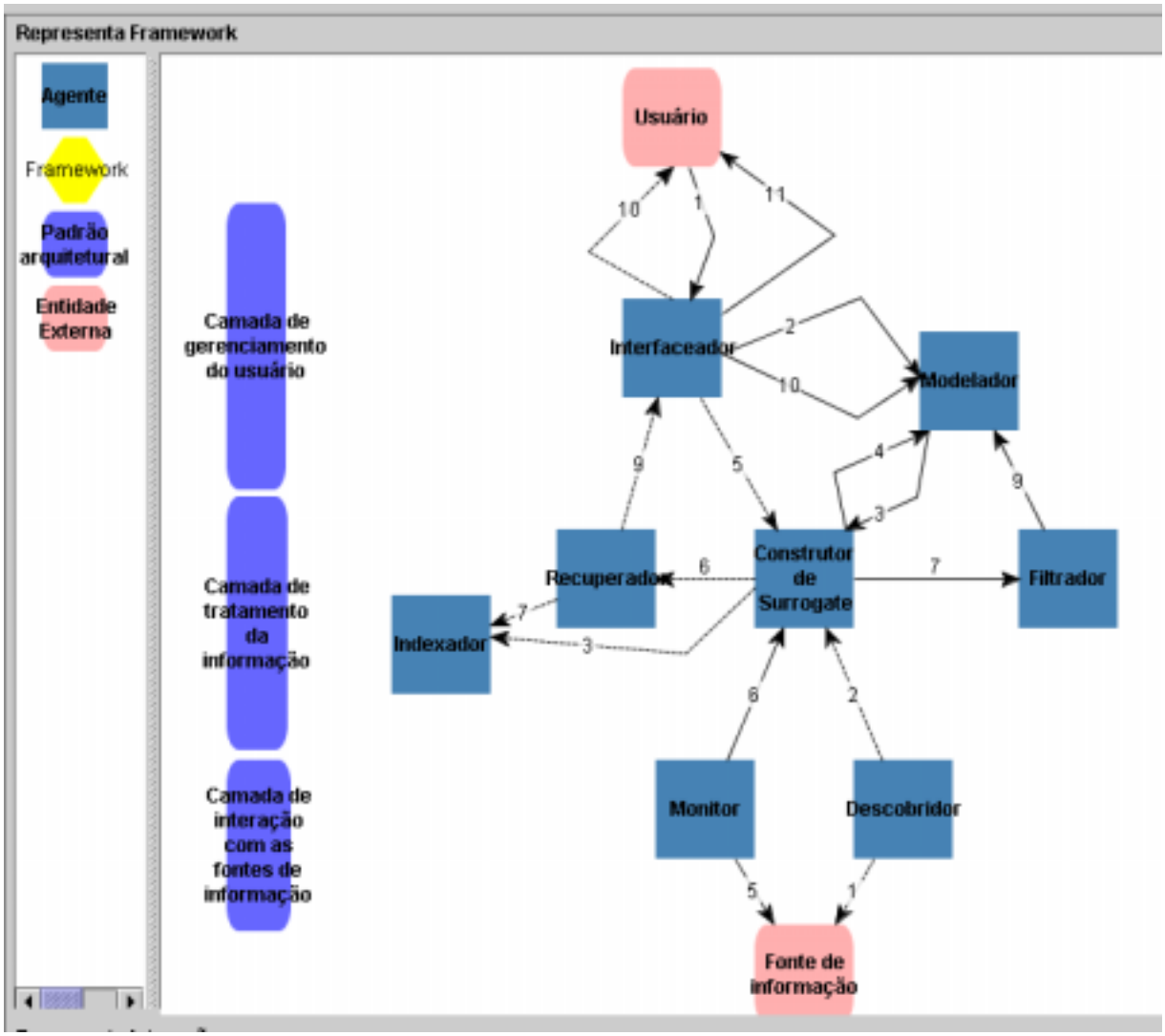

Figura 82. Modelo arquitetural final - framework

6.4. Detalhamento dos agentes do framework

Tarefa 8: Realizar o detalhamento dos agentes do framework, definindo modelos de atividades detalhados.

Após definido o framework foi realizada a tarefa de *detalhamento dos agentes do framework,* foram criados os *modelos de atividades detalhados* para cada agente.

Temos mostrado na Figura 83 o *modelo de atividades detalhado* do agente *Construtor de surrogate*, neste modelo aparecem as atividades realizadas por este agente, tendo as atividades *receber especificação da consulta*, *receber especificação dos elementos de informação* e *receber informações dos perfis* podendo ser realizadas em paralelo. Caso seja requisitado um processo representação de especificação da consulta é realizada a atividade *construir surrogate de consulta,* finalizando com *enviar surrogate da consulta.* Em

especificações dos elementos de informação são realizadas as atividades de *construir surrogates de elementos de informação* e *enviar surrogates de elementos de informação*. Em caso de especificações de perfis são executadas *construir surrogates de perfis* e *enviar surrogate dos perfis*.

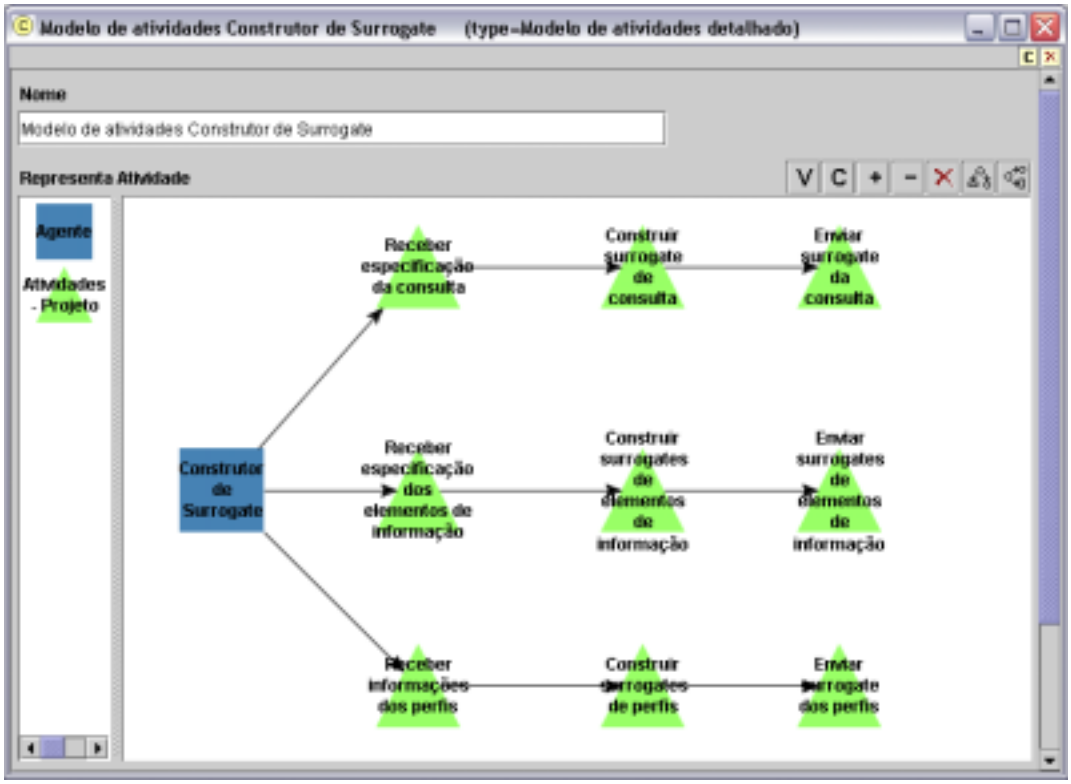

Figura 83. Modelo de atividades detalhado do agente Construtor de Surrogate

No *modelo de atividades detalhado* do agente *Descobridor* (Figura 84), inicialmente é realizada a atividade *descobrir elementos de informação*, de posse destes elementos descobertos ele realiza *enviar elementos descobertos*.

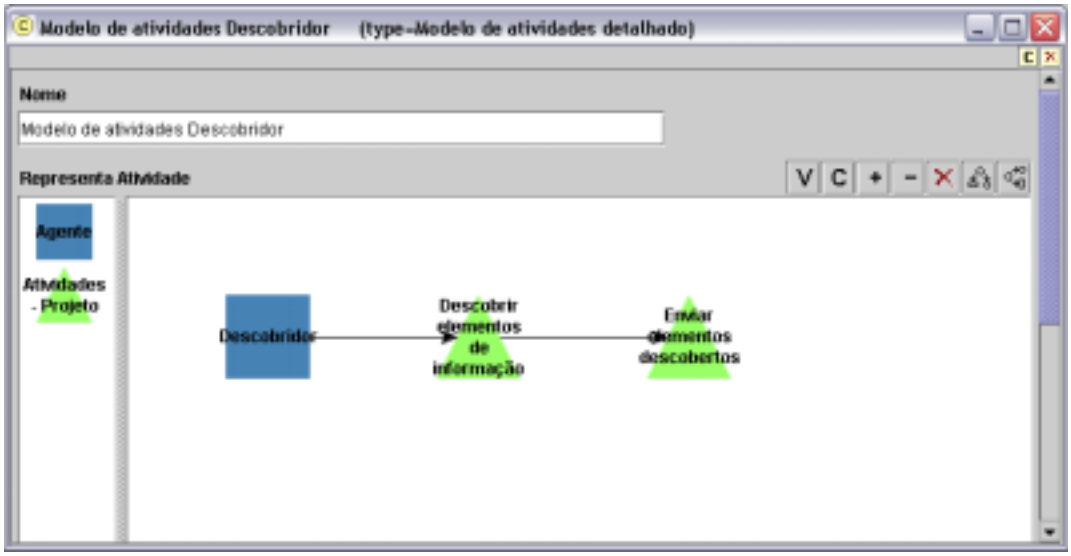

Figura 84. Modelo de atividades detalhado do agente Descobridor

No *modelo de atividades detalhado* do agente *Filtrador* (Figura 85), o processamento inicia com as atividades de *receber surrogate de elementos de informação* e *receber surrogate dos perfis*, que ativam o processo de filtragem através da realização de *fazer análise de similaridade da filtragem* e *comparar surrogates de necessidades a longo prazo e elementos de informação*. Após a realização da filtragem o agente realiza *enviar elementos filtrados.*

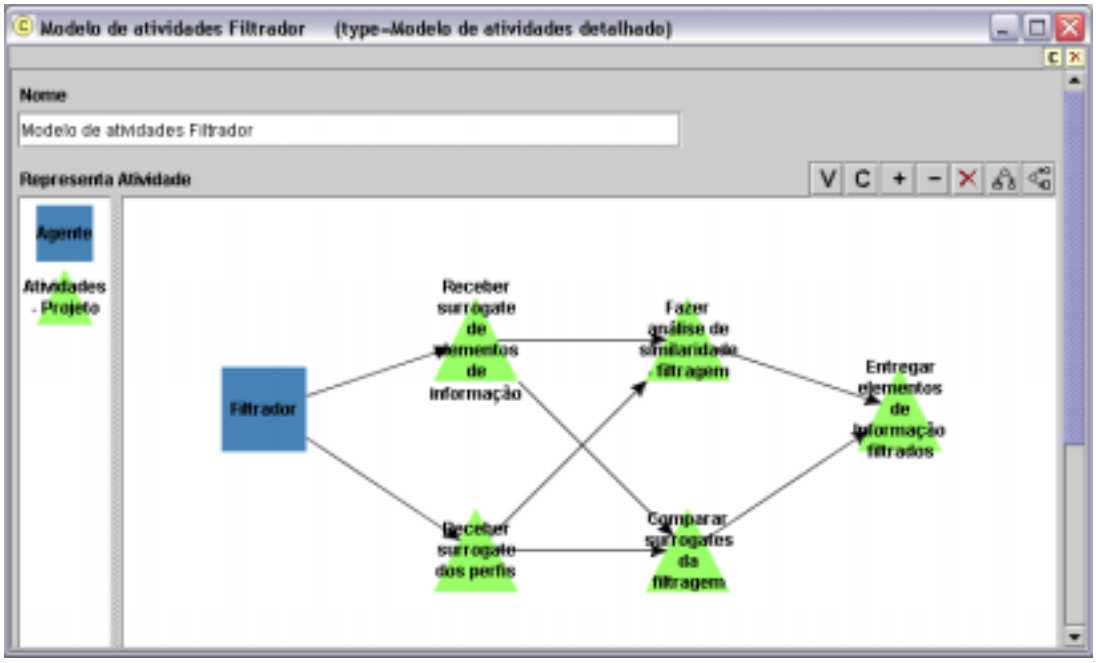

Figura 85. Modelo de atividades detalhado do agente Filtrador

No *modelo de atividades detalhado* do agente *Indexador* (Figura 86), a execução das atividades inicia com *receber surrogate de elementos de informação* para que seja realizada a atividade *indexar surrogates de elementos de informação.* Durante um processo de recuperação os índices são necessários, este agente então realiza as atividades *recuperar índices* e *enviar índices.*

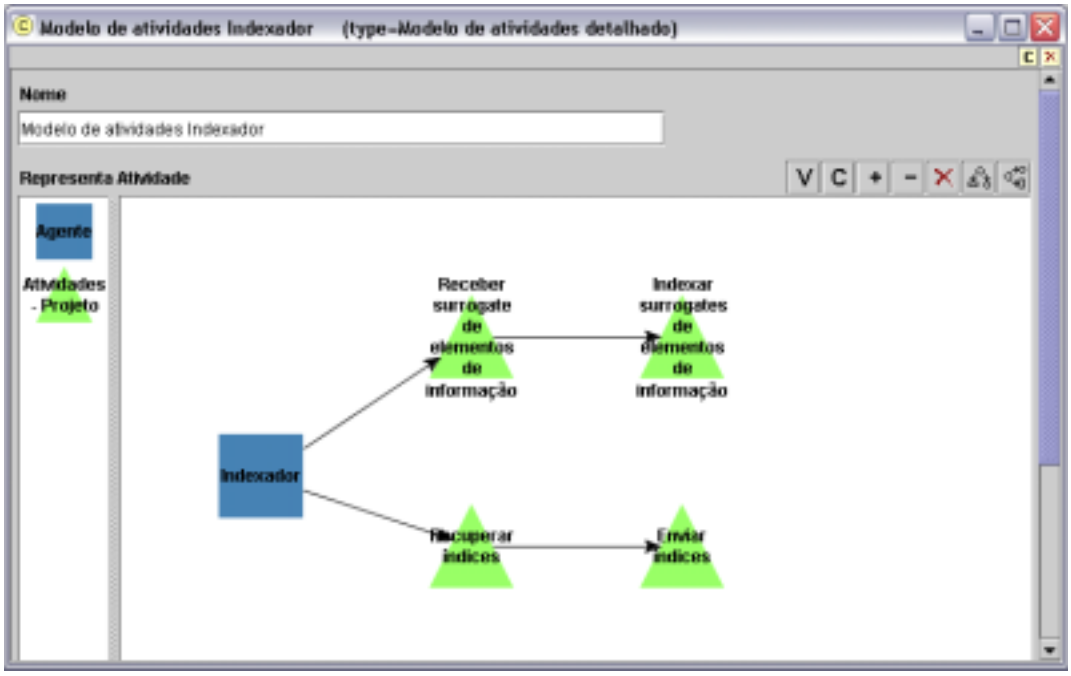

Figura 86. Modelo de atividades detalhado do agente Indexador

No *modelo de atividades detalhado* do agente *Modelador* (Figura 87), as atividades *receber informações do usuário a partir da interface, receber elementos filtrados* e *receber surrogate dos perfis* podem ser executadas em paralelo, as duas primeiras são seguidas da atividade *atualizar modelos,* A ultima é seguida das atividades de *criar e manter modelo de usuário* e *enviar informações dos perfis.*

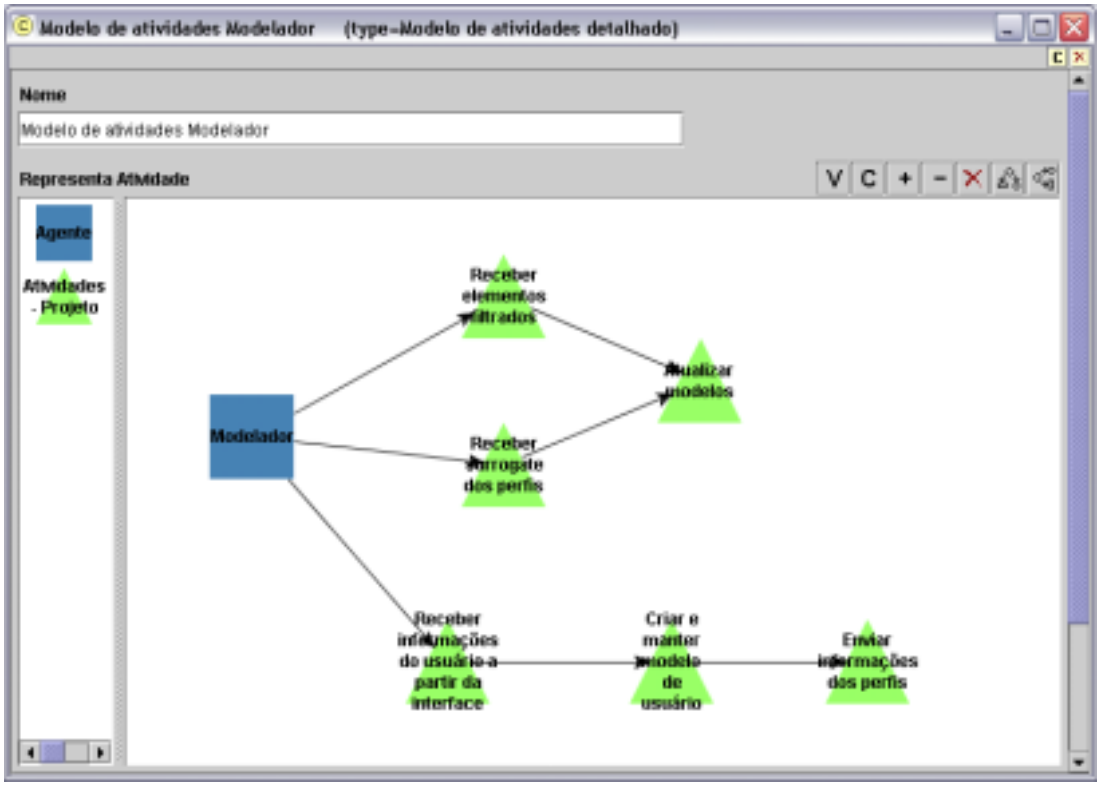

Figura 87. Modelo de atividades detalhado do agente Modelador

No *modelo de atividades detalhado* do agente *Interfaceador* (Figura 88), o processamento é iniciado com a atividade *monitorar usuário*, executada em paralelo a *receber necessidades do usuário.* Em caso de monitoramento podem ser realizadas duas atividades, *enviar informações do usuário* e *acessar elementos filtrados*, dependendo de qual tarefa esta sendo requisitada a este agente. Por outro lado, quando são recebidas necessidades do usuário, é realizada a *atividade processar consulta* seguida por *enviar especificação da consulta* que no final converge em *entregar resultados.*

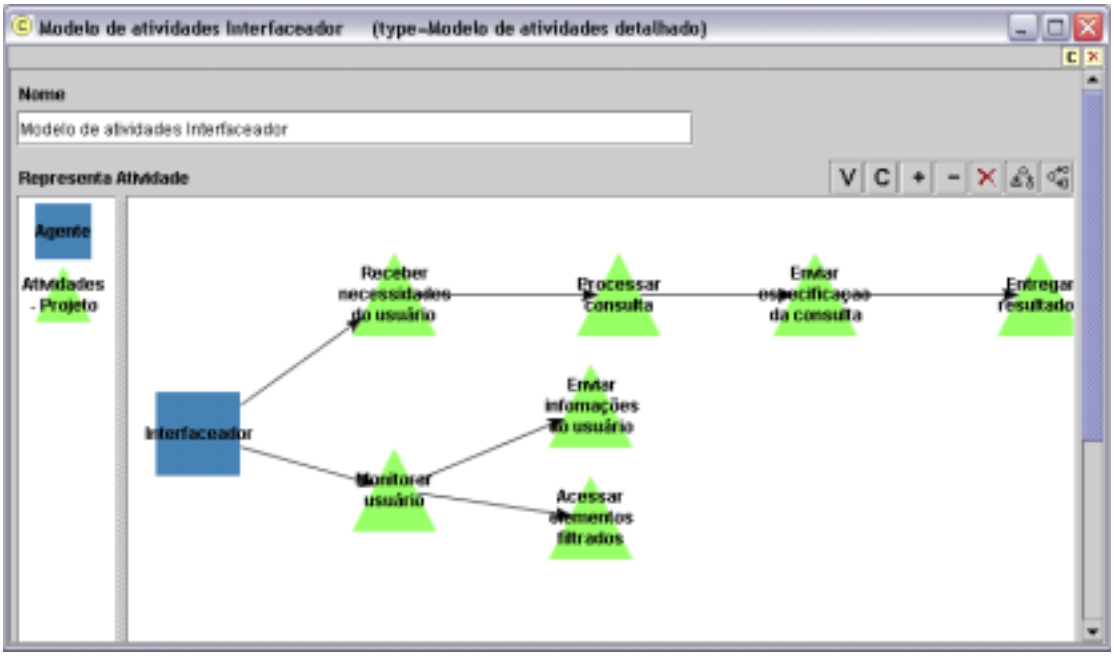

Figura 88. Modelo de atividades detalhado do agente Interfaceador

No *modelo de atividades detalhado* do agente *Monitor* (Figura 89), o agente trabalha de acordo com mudanças em fontes de informação, a todo tempo ele fica vasculhando a Web, e ao perceber mudanças ele executa *detectar mudanças em fontes de informação*, seguida de *enviar mudanças detectadas*.

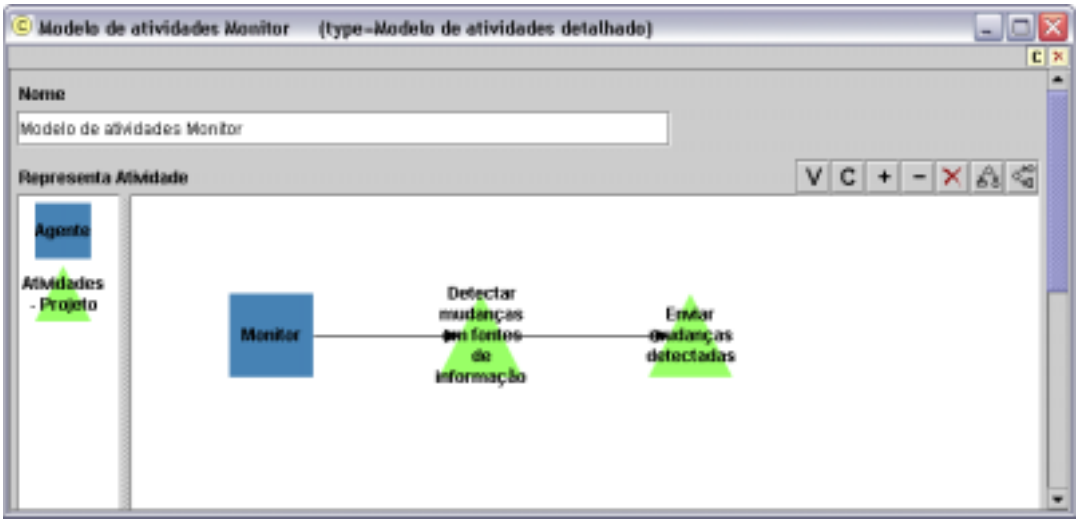

Figura 89. Modelo de atividades detalhado do agente Monitor

No *modelo de atividades detalhado* do agente *Recuperador* (Figura 90), inicialmente é realizada a atividade *receber surrogate da consulta*, seguida de *requisitar índice* e *receber índices*, são então realizadas as atividades de *comparar surrogates da*

consulta e elementos de informação e *fazer análise de similaridade*, finalizando com *enviar resultados*.

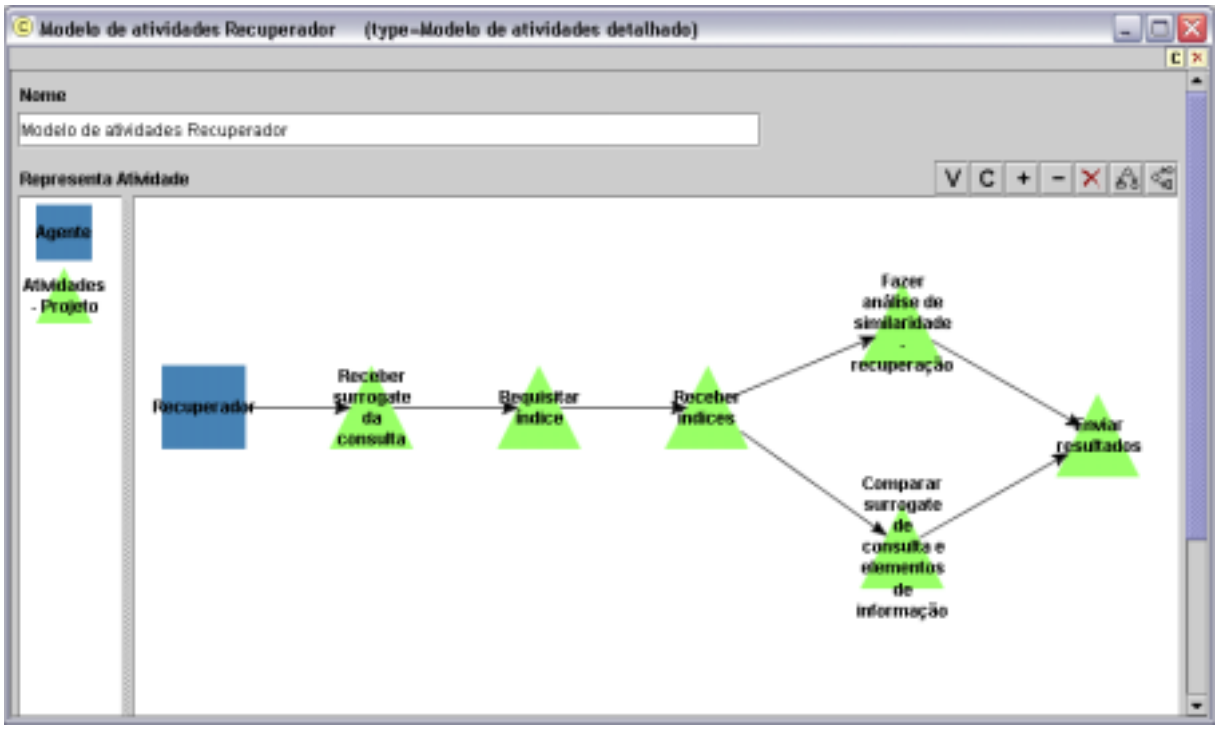

Figura 90. Modelo de atividades detalhado do agente Recuperador

Tarefa 9: Descrever os agentes segundo os dois tipos básicos de agentes: deliberativo e reativo.

Neste momento foram descritos o comportamento e conhecimento de cada agente de acordo com os tipos básicos de agentes: deliberativo e reativo.

O agente *Construtor de surrogate* é responsável por representar internamente as consultas, os elementos de informação e os modelos de usuários. Este conjunto de atividades não requer raciocínio para a sua realização, pois construção da representação de informações é feita através de regras que definirão medidas ou pesos aos dados recolhidos. Portanto, este agente possui as características de um agente reativo.

O *Descobridor* é responsável por descobrir novos elementos de informação de acordo com as áreas de interesse do usuário, esta tarefa não requer nenhum tipo de raciocínio e pode ser baseada em conjuntos de regras, caracterizando um exemplo de agente reativo.

O agente *Recuperador* possui um conjunto de atividades que não necessitam de raciocício sendo tratáveis através da definição de regras de condição-ação, caracterizando um agente do tipo deliberativo.

O *Filtrador* é responsável por filtrar informações relevantes aos interesses do usuário, realiza um conjunto de operações semelhantes ao do *Recuperador*, encaixando-se no perfil dos agentes reativos.

O *Indexador* é responsável por organizar, armazenar e recuperar os índices, atividades perfeitamente tratáveis por um agente do tipo reativo.

O *Interfaceador* possui um comportamento baseado em requisições do usuário, repassando estas ao agente *Construtor de surrogate* e *Modelador*, verificando se a tarefa já foi realizada para no final entregar os resultados ao usuário. Este conjunto de atividades não requer raciocínio, sendo resolvido apenas com um conjunto de regras do tipo: *verificar se já terminou a recuperação, se sim então recebe e entrega resultados ao usuário*. Portanto, este agente possui as características de um agente reativo.

O *Modelador* é responsável pela criação e manutenção dos modelos de usuário. A criação destes modelos é uma tarefa que requer a realização de atividades complexas, como a definição de padrões de modelos e de modelos de usuário. Deve ser um agente do tipo deliberativo.

O *Monitor* age em função de mudanças ocorridas nas fontes de informação. Este agente apenas monitora e detecta mudanças nestas fontes, sendo perfeitamente construído como um agente reativo.

6.4.1. Seleção de padrões de projeto detalhado

Tarefa 10: Selecionar um ou mais padrões de projeto detalhado para cada agente do framework de acordo com a descrição feita e com as informações contidas no padrão.

Depois de descritos os agentes, seus comportamentos e conhecimentos, foram selecionados os padrões para cada agentes.

Foi observado que o *Interfaceador* tem como tarefa principal ser o intermediador entre o sistema e o usuário, para este tipo de problema existe o padrão *Interface*, que pode ser

uma extensão tanto do padrão reativo quanto deliberativo, este último foi selecionado para este agente.

O agente *Modelador* possui tarefas que encaixam no problema que o padrão modelagem pretende resolver: “Como criar e manter modelos de usuários que serão utilizados como referência para que a aplicação ofereça serviços personalizados aos seus usuários?”. Por esta razão este padrão foi selecionado para este agente. Este padrão poderia ser aplicado como uma extensão dos padrões reativo e deliberativo, neste caso foi usado o deliberativo.

Os outros agentes seguiram os padrões básicos definidos na coletânea; os agentes *Descobridor*, *Indexador* e *Monitor* seguem o padrão reativo e os agentes *Construtor de surrogate*, *Filtrador* e *Recuperador* seguem o padrão deliberativo.

6.4.2. Refinamento dos agentes

Tarefa 11: Refinar os agentes através da aplicação dos padrões e construção dos modelos de projeto detalhado.

Após a seleção dos padrões foi então construído o *modelo arquitetural detalhado* mostrando os *agentes* e sua arquitetura, estruturados segundo um padrão de projeto detalhado selecionado.

O agente *Construtor de surrogate* segue o padrão reativo, possuindo um conjunto de regras que indicarão qual ação tomar. A solução envolve divisão da estrutura em quatro módulos: *comunicação, ação, regras* e *sensores* (Figura 91).

Através do *módulo sensores*, o agente irá receber informações referentes a consulta, usuários e elementos de informação. As operações deste módulo são: *receber especificação da consulta*, *receber especificação dos elementos de informação* e *receber informações dos perfis.*

No *módulo de regras* foi armazenado o conjunto de regras a serem executadas de acordo com um estimulo recebido pelo módulo de sensores e que serão executados pelo módulo de ação, estas regras são instancias da meta-classe *Conhecimento*, um exemplo de regra contida é: *se receber requisição de construção da representação de consulta então executar construir surrogate de consulta seguida de enviar surrogate da consulta.*

Através do *módulo ação* o agente atua no ambiente executando as operações de acordo com o módulo de regras. As ações contidas neste módulo referem-se às operações *construir surrogate de consulta, construir surrogates de elementos de informação* e *construir surrogates de perfis*. A técnica escolhida para ser utilizada na construção dos surrogates é a do modelo do espaço vetorial, as informações são vetores num espaço de n dimensões. Esta técnica foi escolhida pelo fato de ser um dos modelos mais utilizados e simples de implementar.

O *módulo ação* serve ao *módulo comunicação*, que é responsável por enviar e processar mensagens, realizando as operações *enviar surrogate da consulta, enviar surrogates de elementos de informação* e *enviar surrogate dos perfis*.

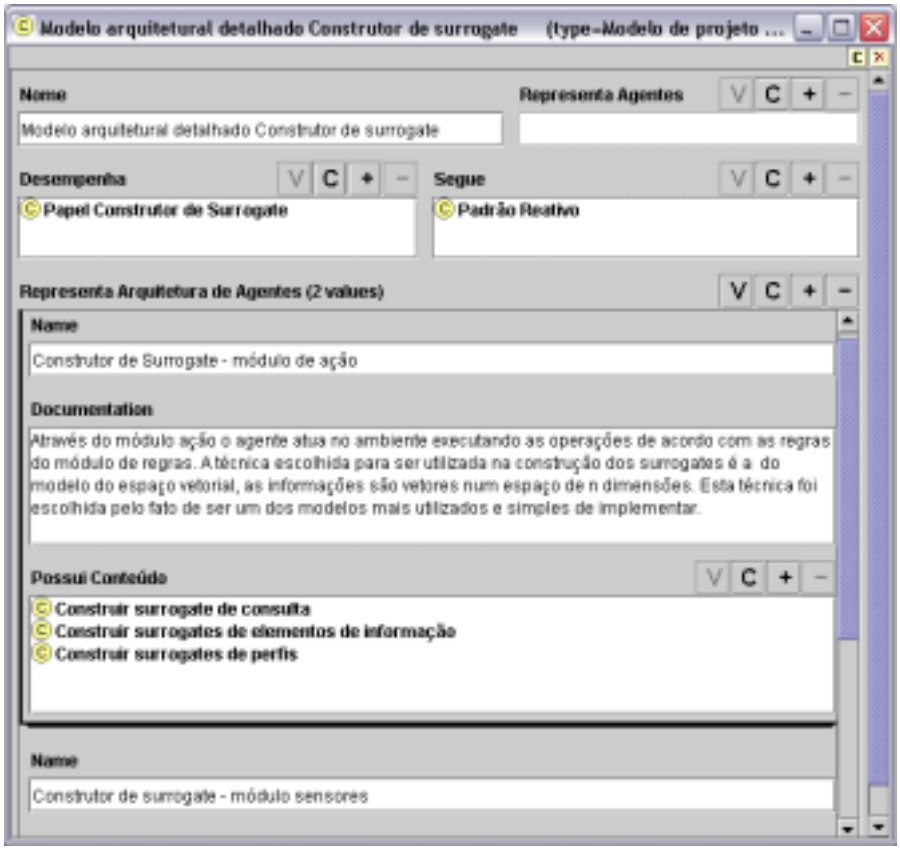

Figura 91. Modelo de projeto detalhado do agente Construtor de Surrogates

O agente *Interfaceador* segue o padrão Interface, este padrão foi utilizado neste caso como uma especialização do padrão Reativo. A estrutura interna do agente seguira á descrita no padrão reativo. A solução envolve divisão da estrutura em quatro módulos: *comunicação, ação, regras* e *sensores* (Figura 92).

Através do *módulo sensores* o agente recebera requisições, informações do usuário e os resultados a serem entregues. As operações contidas neste módulo são: *receber requisições do usuário* e *receber informações do usuário* refinada da atividade *monitorar usuário*.

No *módulo de regras* foi armazenado o conjunto de regras a serem executadas, algumas destas regras são: *se receber requisição de necessidade pontual então processar consulta, se receber resultados então entregar resultados.*

Através do *módulo de ação* o agente *Interfaceador* irá atuar através das operações de *monitorar usuário* e *acessar elementos filtrados*. Este módulo em conjunto com o *módulo de comunicação* executa as operações *enviar especificação da consulta, enviar informações do usuário* e *entregar resultados.*

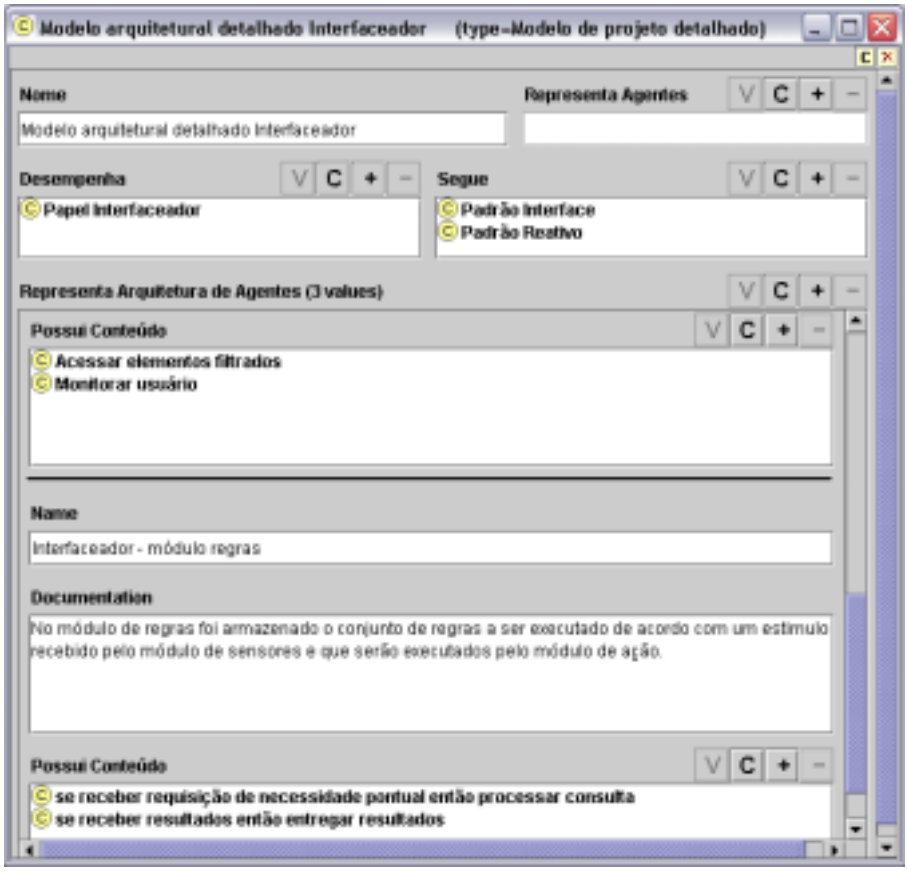

Figura 92. Modelo de projeto detalhado do agente Interfaceador

Para o agente *Recuperador* (Figura 93) foi selecionado o padrão reativo, este agente não necessita de raciocínio, irá possuir um conjunto de regras, estas regras guiarão as

ações executadas pelos agentes. Segundo o *padrão reativo* o agente é estruturado em quatro módulos: *comunicação, ação, regras* e *sensores*.

Através do *módulo sensores*, o agente irá receber as requisições e informações advindas dos agentes *Construtor de surrogate* e *Indexador*. Neste módulo ficarão localizadas as operações *receber surrogate da consulta* e *receber índices indexados*. O recebimento de um surrogate da consulta indica o início do processo de recuperação.

No *módulo regras* são armazenadas o conjunto de regras a serem executadas de acordo com um estimulo recebido pelo *módulo de sensores* e que serão executados pelo *módulo de ação*, algumas das regras contidas é: *se receber requisição surrogate de consulta então requisitar e receber índice; se recebeu índice então realizar comparar surrogates e fazer análise de similaridade*.

Através do *módulo ação* o agente atua no ambiente executando as regras definidas no módulo de regras. As ações contidas neste módulo referem-se às operações de *comparar surrogates* e *fazer análise de similaridade.*

O *módulo ação* serve ao *módulo comunicação* que é responsável por enviar e processar mensagens, realizando as operações *requisitar índice* e *enviar resultados.*

Figura 93. Modelo de projeto detalhado do agente Recuperador

Para o agente *Modelador* (Figura 94) foi selecionado o padrão deliberativo, este agente foi estruturado com um modelo simbólico do ambiente e de si mesmo e mais cinco módulos: *comunicação, ação, raciocínio, sensores* e *conhecimento*.

Através do *módulo sensores*, o agente irá receber as requisições e informações advindas dos agentes *Interfaceador, Construtor de surrogate e Filtrador*. Neste módulo ficarão localizadas as operações *receber elementos filtrados, receber surrogate de perfis, receber informações do usuário pa partir da interface*.

No *módulo raciocínio,* baseado nas informações sobre o usuário que chega a ele, o agente elabora metas e planos para a criação dos modelos de usuário. Neste módulo ficará localizada a operação de *elaborar planos para a criação e atualização de modelos de usuário*.

Através do *módulo ação* o agente atua no ambiente executando as ações do plano gerado pelo *módulo raciocínio*. As ações contidas inicialmente seriam às operações de *criar e manter modelo de usuário* e *atualizar modelo,* que se desdobraram nas seguintes atividades: *construir vetor de interesses do usuário, criar estereótipos, criar características do usuário e atribuir um estereotipo ao usuário.* Estas atividades utilizam a técnica para a aquisição implícita do modelo do usuário.

Durante a execução de um plano, o agente *Modelador* pode determinar a necessidade de cooperação com outros agentes, requerendo informações ou pedindo para executar ações, como por exemplo uma a criação de um surrogate de um perfil. Esta cooperação é suportada pelo *módulo comunicaçã*o.

O *módulo ação* serve ao *módulo comunicação,* que é responsável por enviar e processar mensagens, realizando as operações *enviar informações do perfil e enviar elementos filtrados.*.

No *módulo conhecimento* são armazenadas informações sobre o estado corrente do ambiente através do histórico das percepções do agente, ações executadas e resultados das ações. Este ainda possui conhecimento sobre as habilidades dos outros agentes da sociedade. Neste módulo estarão armazenados os modelos de usuário e os estereótipos de usuário.

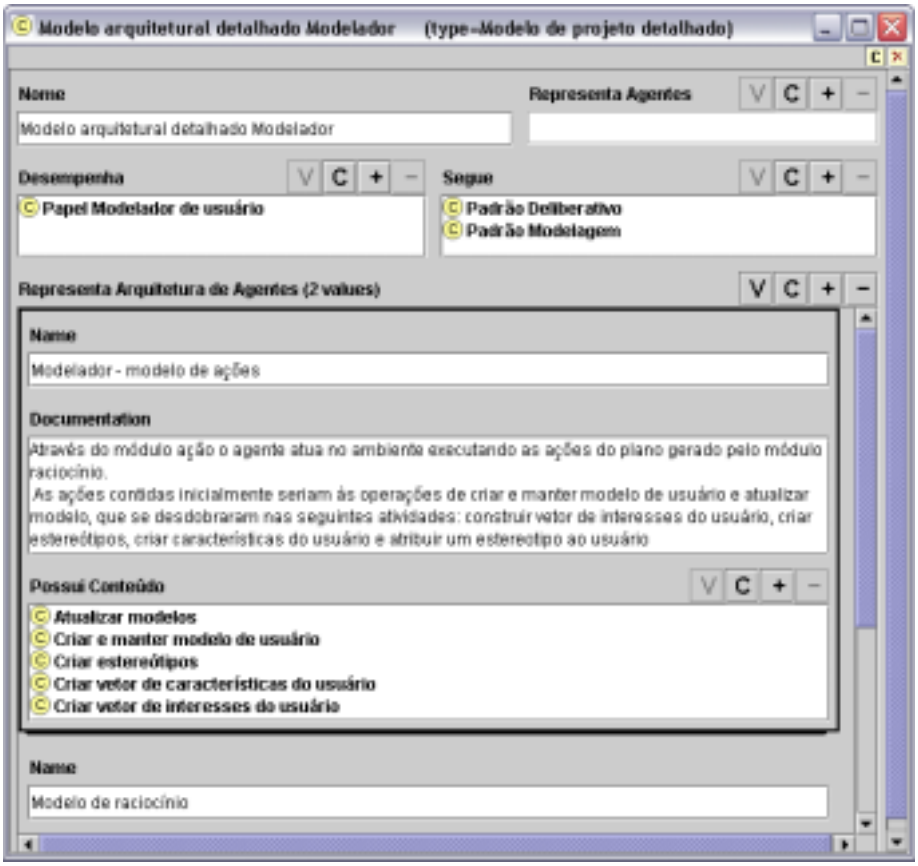

Figura 94. Modelo de projeto detalhado do agente Modelador

As estruturas dos outros agentes seguem os padrão reativo e deliberativo, estes ficaram semelhantes aos agentes reativos e deliberativos mostrados anteriormente nesta tarefa.

6.5. Considerações Finais

Neste capítulo foi apresentado um estudo de caso para a avaliação da técnica DDEMAS e da ferramenta ONTODD, onde foi feita uma análise da área do acesso a informação. O estudo possibilitou o aprimoramento tanto da técnica DDEMAS quanto da ferramenta ONTODD.

Um ponto a ser destacado é que a experiência de implementação provavelmente leve a uma reavaliação da estrutura do framework assim como das estruturas internas dos agentes.

7. Conclusão

A construção sistemática de frameworks multiagente é uma prática crescente em termos de aplicabilidade e importância. Os frameworks fornecem soluções de projeto para uma família de aplicações, permitindo a redução de esforços e de custos na construção destas aplicações.

Este trabalho apresentou DDEMAS, uma técnica para o Projeto de Domínio na Engenharia de Domínio Multiagente e a ONTODD, uma ferramenta que auxilia a aplicação da técnica. A ONTODD é uma ontologia genérica que representa o conhecimento acerca do desenvolvimento de frameworks multiagente segundo a DDEMAS.

A DDEMAS e a ONTODD foram avaliadas e refinadas através da sua aplicação na construção de frameworks para a recuperação e filtragem da informação. Também estão sendo estendidas suas aplicações na resolução de problemas específicos, como o acesso a informação jurídica [LINDOSO, 2003]

O objetivo final é a elaboração de uma metodologia para a Engenharia de Domínio Multiagente, com o desenvolvimento de técnicas para as fases de Análise de Domínio e Usuários, Projeto e Implementação de Domínio.

7.1. Resultados e contribuições da pesquisa

As principais contribuições desta pesquisa foram:

- Análise do estado da arte da fase de projeto no desenvolvimento de software e dos estilos arquiteturais existentes;
- Análise do estado da arte da Engenharia de Domínio, centrada no projeto de domínio;
- Análise do estado da arte das técnicas de projeto no nível global e detalhado no desenvolvimento de software baseado em agentes;
- Análise e aplicação de sistemas de padrões na construção de sistemas multiagente desenvolvidos no contexto do projeto MaAE [RIBEIRO, 2004] [SILVA, 2003];
- Elaboração de uma técnica (DDEMAS) e uma ferramenta (ONTODD) para o projeto de domínio global e detalhado de sistemas multiagente;

- Avaliação preliminar da técnica e ferramenta propostas através do desenvolvimento de um estudo de caso na construção de um framework multiagente para o acesso à informação.

7.2. Trabalhos futuros

Alguns trabalhos futuros serão e estão sendo desenvolvidos a partir desta proposta. A integração das técnicas GRAMO [FARIAb, 2003] e DDEMAS é o principal trabalho a ser abordado, consistindo de uma proposta de metodologia para a Engenharia de Domínio Multiagente, constituindo assim um processo completo, baseado em ontologias, para o desenvolvimento de sistemas multiagentes.

Outro aspecto é o aprimoramento da fase de projeto detalhado da DDEMAS para abordar níveis maiores de detalhamento do comportamento e conhecimento do agente, possibilitando a aproximação maior do projeto detalhado à implementação dos agentes.

A DDEMAS e a ONTODD estão também sendo utilizadas no desenvolvimento de frameworks em outras áreas de aplicação, como os sistemas de recomendação.

Por fim um trabalho de grande importância que esta sendo desenvolvido é a implementação completa do framework desenvolvido no estudo de caso, bem como a utilização desta implementação em uma aplicação especifica.

Bibliografia

[AMBRIOLA,1990] Ambriola, V., Ciancarini, P., Montangero, C.Software Processes as a Hierarchy.ISPW6, Hokkaido, Japan, Oct., 1990.

[APPLETON, 2002] Appleton, Brad. Patterns and Software: Essential Concepts and Terminology, disponível em: http://www.enteract.com/~bradappdocpatterns-intro.html, acessado em 2002.

[BACH, 1986] Bach, M. J., The Design of the UNIX Operating System, ch. 5.12, pp. 111-119. Software Series, Prentice-Hall, 1986.

[BALZER, 1986] Balzer, R. M., Living with the next generation operating system, in Proceedings of the 4th World Computer Conference, September 1986.

[BARBACCI, 1988] Barbacci, M. R., Weinstock, C. B., and Wing, J. M., Programming at the processor-memoryswitch level, in Proceedings of the 10th International Conference on Software Engineering, (Singapore), pp. 19-28, IEEE Computer Society Press, April 1988.

[BATORY, 1991] Batory, D. and O'Malley, S., "The design and implementation of hierarchical software systems using reusable components," Tech. Rep. TR-91-22, Department of Computer Science, University of Texas, Austin, June 1991.

[BUSCHMANN, 1996] Buschmann, F., Meuniers, R. et al. A System of Patterns. Pattern-Oriented Software Architecture. Wiley, 1996.

[CASTRO, 2001] Castro, Jaelson, Kolp, Manuel, Mylopolus, John. "A Requirement-Driven Software Development Methodology", 13th International Conference on Advanced Information Systems Engineering CAISE01, Interlaken, Switzerland, 4-8 June, 2001.

[COPLIEN, 2003] Coplien, Jim. Software Design Patterns: Common Questions and Answers, disponível em: ftp://st.cs.uiuc.edu/pub/patterns/papers/PatQandA.ps, acessado em 2003.

[CORNWELL, 1996] Cornwell, P.C. HP Domain Analysis: Producing Useful Models for Reusable Software. Hewlett-Packard Journal, 1996.

[COSSENTINO, 2002] Cossentino, M., Burrafato, P., Lombardo, S., Sabatucci, L. - Introducing Pattern Reuse in the Design of Multi-Agent Systems - AITA'02 workshop at NODe02 - 8-9 October 2002 - Erfurt, Germany.

[DELISLE, 1990] Delisle, N. And Garlan, D., Applying formal specification to industrial problems: A specification of an oscilloscope, IEEE Software, September 1990.

[DILEO, 2002] DiLeo, Jonathan, Jacobs, Timothy, Deloach, Scott. Integrating Ontologies into Multiagent Systems Engineering. 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002), 15-16 July 2002, Bologna (Italy).

[FARIAa, 2003] Faria, Carla Gomes de. & GIRARDI, Rosario. Especificação de uma Ontologia Genérica para a Análise de Requisitos da Engenharia de Aplicações Multiagente. Terceiro Congresso Brasileiro de Computação (CBComp 2003), UNIVALI, Itajaí, SC, Brasil. 2003.

[FARIAb, 2003] Faria, Carla Gomes de. & GIRARDI, Rosario. GRAMO: Uma Técnica para a Construção de Modelos de Domínio Reutilizáveis no Desenvolvimento de Sistemas Multiagente, XII Seminário de Computação (SEMINCO 2003). Centro de Convenções Willy Sievert, PROEB, Blumenau, Santa Catarina, Brasil. 2003.

[FARIAc, 2003] Faria, Carla Gomes de. e Ribeiro, Ismênia de Oliveira e Girardi, Rosario. Uma Ontologia Genérica para a Análise de Domínio e Usuário na Engenharia de Domínio Multiagente, Anais do Simpósio de Informática da Região Centro/RS (SIRC/RS 2003), Ed. UNIFRA. Santa Maria, Rio Grande do Sul, Brasil. 20 a 22 de agosto de 2003.

[FARIAd, 2003] Faria, Carla Gomes de. e Ribeiro, Ismênia de Oliveira e Girardi, Rosario. Especificação de uma Ontologia Genérica para a Construção de Modelos de Usuários, Anais da Terceira Jornada Iberoamericana de Engenharia de Software e Engenharia do Conhecimento (JIISIC 2003). Valdivia, Chile. 26 a 28 de novembro de 2003.

[FERBER, 1999] Ferber, J. Multi-Systems: An Introduction to Distributed Artificial Intelligence. 1. ed. Addison-Wesley, 1999.

[FERREIRA, 2002] Ferreira, Steferson Lima Costa. Arquiteturas de Software Baseadas em Agentes: do Nível Global ao Detalhado. Revista Eletrônica de Iniciação Científica, 2002.

[FERREIRAa, 2003] Ferreira, Steferson Lima Costa, GIRARDI, Rosario. Especificação de uma Ontologia Genérica para o Projeto de Domínio de Aplicações Multiagente, Proceedings of Chilean Computing Week 2003 (CCW2003), Chillán, Chile, Universidad del Bio-Bio, 2003.

[FERREIRAb, 2003] Ferreira, Steferson Lima Costa e Girardi, Rosário Uma Técnica para o Projeto de Domínio de Sistemas Multiagente. Workshop de Informática do IMESA, 2003.

[FOREMAN, 1996] Foreman, John. “Product Line Based Software Development - Significant Results Future Chalenges”, Software Technology Conference, Salt Lake City, UT, April 23, 1996.

[FRIDMAN, 2001] Fridman, N., Mcguinness, D. Ontology Development 101: A Guide to Creating Your First Ontology, Knowledge Systems Laboratory, March, 2001.

[FRIDRICH, 1985] M. Fridrich and W. Older,. Helix: The architecture of the XMS distributed file system, IEEE Software, vol. 2, pp. 21-29, May 1985.

[GARLAN, 1988] D. Garlan, G. E. Kaiser, and D. Notkin, Using tool abstraction to compose systems, IEEE Computer, vol. 25, June 1992.

[GARLAN, 1994] D. Garlan and M. Shaw. An Introduction to Software Architecture. Carnegie Mellon University Technical Report CMU-CS-94-166, January 1994. 88

[GERETY, 1989] C. Gerety, HP Softbench: A new generation of software development tools, Tech. Rep. SESD-89-25, Hewlett-Packard Software Engineering Systems Division, Fort Collins, Colorado, November 1989.

[GIRARDI, 2001] Girardi, Rosario. Agent-Based Application Engineering, In: 3rd Internacional Conference on Enterprise Information Systems (ICEIS 2001). Setúbal, Portugal. 2001.

[GIRARDI, 2002] Girardi, R. Reuse in Agent-based Application Development. 1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS´2002), 2002.

[GIRARDIa, 2003] Girardi, Rosario. & RIBEIRO, Ismênia. & BEZERRA, Geovane. Towards a System of Patterns for the Design of Agent-based Systems, Proceedings of The Second Nordic Conference on Pattern Languages of Programs” (VikingPLoP 2003). Bergen, Norway. 2003.

[GIRARDIb, 2003] Girardi, Rosario. & FARIA, Carla. A Generic Ontology for the Specification of Domain Models, Proceedings of 1st International Workshop on Component Engineering Methodology (WCEM'03) at Second International Conference on Generative Programming and Component Engineering. Erfurt, Germany. 2003.

[GIRARDIc, 2003] Girardi, Rosario. Engenharia de Domínio Multiagente, Anais da Terceira Jornada Iberoamericana de Engenharia de Software e Engenharia do Conhecimento (JIISIC 2003), Seção de

Oportunidades de Cooperação . Valdivia, Chile. 26 a 28 de novembro de 2003.

[GIRARDI, 2004] Rosario Girardi. Engenharia de Software baseada em Agentes, Anais do IV Congresso Brasileiro de Ciência da Computação (a ser publicado). Itajai, SC. 2004.

[GOMMA 1984] Gomma, H., A Software Design Method for Real-Time Systems, Communication of the ACM, Vol. 27, No. 9, pp 938-949. September 1984.

[GRENN, 2002] Grenn S., Hurst L. et al. Softwares Agents: A Review. Disponível na Internet no endereço: http://www.cs.tcd.ie/research_groups/iag/iag.html, 2002.

[GUIZZARDI, 2000] Guizzardi, G. Uma abordagem para o desenvolvimento de software orientado ao reuso, baseado em ontologias formais de domínio. Universidade Federal do Espirito Santo. Dissertação de Mestrado. 2000

[HABERMANN, 1986] A. N. Habermann and D. S. Notkin, Gandalf: Software development environments, IEEE Transactions on Software Engineering, vol. SE-12, pp. 1117-1127, December 1986.

[HABERMANN, 1991] A. N. Habermann, D. Garlan, and D. Notkin, Generation of integrated task-specific software environments, in CMU Computer Science: A 25th Commemorative (R. F. Rashid, ed.), Anthology Series, pp. 69-98, ACM Press, 1991

[HERMANS, 2001] Hermans, B. Intelligent Software Agents on the Internet; http://www.firstmonday.dk/issues/, acessado em 2001.

[HEWITT, 1969] C. Hewitt, Planner: A language for proving theorems in robots, in Proceedings of the First International Joint Conference in Artificial Intelligence, 1969.

[HORSTMANN,1997] Horstmann, C. S., Cornell, G. Core Java: Fundamentals. 2. Ed.. Sun Press, 2 v. 1997.

[JOHNSON, 1997] Johnson, R. E. Frameworks = (Components + Patterns): How frameworks compare to other object-oriented reuse techniques, Communications of the ACM, v. 40, n. 10, 1997.

[KAHN, 1974] G. Kahn, The semantics of a simple language for parallel programming, Information Processing, 1974.

[KNAPIK, 1998] Knapik, M. e Johnson, J. Developing Inteligent Agents for Distributed Systems. Computing McGraw-Hill, NY:McGraw-Hill, 1998.

[KRASNER, 1988] Krasner, E. K.; Pope, S.T. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal of Object Oriented Programming, p. 26-49, 1988.

[KYO, 1990] Kyo. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. "Feature-Oriented Domain Analysis (FODA) Feasibility Study". Technical Report, CMU/SEI-90TR -21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990.

[LAUER,1979] Lauer, H. C. and Satterthwaite, E. H., Impact of MESA on system design, in Proceedings of the Third International Conference on Software Engineering, (Atlanta, GA), pp. 174-175, IEEE Computer Society Press, May 1979.

[LINDOSO, 2003] Alisson Lindoso, Ivo Serra e Rosario Girardi. ONTOINFOJUS: Um Modelo de Domínio baseado em Ontologias para o Acesso à Informação na Área Jurídica, Anais do V Encontro de Estudantes de Informática do Tocantins (ENCOINFO 2003), Ed. ULBRA, pp. 251-260. Palmas, Tocantins, Brasil. 29 a 30 de outubro de 2003.

[MAGNAN, 2001] Magnan, M. A. S., Murta, L. G. P., Souza, J. M., Werner, C. M. L. Modelos de domínio e Ontologias: uma comparação através de um estudo de caso prático em hidrologia. IV International Symposium on Knowledge Management/Document Management (ISKM/DM'2001). Curitiba, agosto 2001.

[MCCLAIN, 1991] G. R. McClain, ed., Open Systems Interconnection Handbook. New York, NY: Intertext Publications McGraw-Hill Book Company, 1991.

[MEEKEL,1997] Meekel, J. et al. From Domain Models to Architecture Frameworks. In: SSR'97 - ACM SYMPOSIUM ON SOFTWARE REUSABILITY, 1997, Boston, EUA. Anais… Boston, 1997.

[MOORE, 1991] Moore, J. M., Bailin, S. C. Domain Analysis: Framework for reuse. IEEE Computer Society Press. Tutorial, pp 179-202. 1991.

[MYLOPOULOS, 2000] Mylopoulos, J. and Castro, J., Tropos: A Framework for Requirements-Driven Software Development, Brinkkemper, J. and Solvberg, A. (eds.), Information Systems Engineering: State of the Art and Research Themes, Springer-Verlag, pp. 261-273. June 2000.

[ODELL, 2000] J. Odell, H. Van Dyke Parunak, and B. Bauer. Extending UML for agents. In Proceedings of the 2nd Int. Bi-Conference Workshop on Agent-Oriented Information Systems, AOIS'00,

pages 3–17, Austin, USA, July 2000.

[POULIN,1997] Poulin, J. S. On the Contributions of Reuse Research and Development to the State-of-the-Practice in Reuse. In: SSR'97 - ACM SYMPOSIUM ON SOFTWARE REUSABILITY, 1997, Boston, EUA. Anais… Boston, 1997.

[PROTÉGÉ, 2003] Protégé Project. (2003) http://protege.stanford.edu. Acesso em: 05 de maio de 2003.

[REISS, 1990] S. P. Reiss, Connecting tools using message passing in the field program development environment, IEEE Software, July 1990.

[RIBEIRO, 2001] Ribeiro, Ismênia e Girardi, Maria del Rosário. Uma Análise de Padrões de Projeto para o Desenvolvimento de Software Baseado em Agentes. Monografia de Graduação, UFMA, 2001.

[RIBEIRO, 2003] Ribeiro, Ismênia. & GIRARDI, Rosario. Padrões Arquiteturais e de Projeto para a Modelagem de Usuários baseada em Agentes, In: Proc. of The Third Latin American Conference on Pattern Languages of Programming - SugarLoafPlop, Porto de Galinhas, 2003.

[RIBEIRO, 2004] Ribeiro, Ismênia. Um Sistema de Padrões Baseados em Agentes para a Modelagem de Usuários e Adaptação de Sistemas. Dissertação de Mestrado. Universidade Federal do Maranhão. São Luís, 2004.

[RUSSELL, 1995] Russell, S, Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.

[SALTON, 1983] Salton, G., e M. Mcgill. "An Introduction to Modern Information Retrieval". New York: McGraw-Hill. 1983

[SERRA, 2003] Serra, I. C., Girardi, R. (2003) "Uma Abordagem Gerativa para a Engenharia de Domínio Multiagente", Anais do III Encontro de Informática do Tocantins (ENCOINFO 2003), Palmas, Tocantins, Brasil, pp. 261-270, Ed. ULBRA. 29 a 30 de outubro de 2003.

[SHAW, 1983] M. Shaw, E. Borison, M. Horowitz, T. Lane, D. Nichols, and R. Pausch, Descartes: A programming-language approach to interactive display interfaces, Proceedings of SIGPLAN '83: Symposium on Programming Language Issues in Software Systems, ACM SIGPLAN Notices, vol. 18, pp. 100-111, June 1983.

[SHAW, 1996] Shaw, M.; Garlan, D. Software architecture - perspectives on an emerging discipline. Upper Saddle River: Prentice Hall, 1996.

[SILVA, 2003] Silva Junior, Geovanne Bezerra da. Padrões Arquiteturais para o Desenvolvimento de Aplicações Multiagente. Dissertação (Mestrado em Ciência da Computação) – Curso de Pós-Graduação em Engenharia de Eletricidade, Universidade Federal do Maranhão, 2003.

[SODRÉ, 2002] Sodré, Alídia Clícia Silva. MADS: Uma Metodologia para o Desenvolvimento de Sistemas Baseados em Agentes. Conferência Ibero-americana em Sistemas, Cibernética e Informática (CISCI 2002), julho 2002.

[TROY, 1993] Troy, R. Software Re-use, In: Objectworld Conference. Anais. 1993.

[ZAND, 1997] Zand, M. et al. Reuse Research and Development: Is it on the right track?. In: SSR'97 - ACM SYMPOSIUM ON SOFTWARE REUSABILITY, 1997, Boston,EUA. Anais… Boston, 1997.

[ZIMMERMANN, 1980] Zimmermann, H., OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on communications COM-28, No. 4: April 1980.

[YOURDON, 1978] Edward Yourdon and L. Constantine. Structured Design. Yourdon Press, New York, NY, 1978.

[YOURDON, 1990] Yourdon, Edward. Análise Estruturada Moderna; tradução Dalton Conte de Alencar. Rio de Janeiro, Campus, 1990.

[YU, 1995] E. Yu. Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Department of Computer Science, 1995.

[WERNER, 2000] Werner, C. M. L., Braga, R. M. M. "Desenvolvimento Baseado em Componentes", XIV Simpósio Brasileiro de Engenharia de Software, Mini-curso, João Pessoa, Outubro, 2000.

[WOOLDRIDGE, 1995] Wooldridge, Michael, Jennings, Nicholas. Intelligent Agents: Theory and Practice, Knowledge Engineering Review, October 1994, Rev. January, 1995.

[WOOLDRIDGE, 2000] Wooldridge, M. & Jennings, N. R. & Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. International Journal of Autonomous Agents and Multi-Agent Systems, 3, 2000.