UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

ÁREA: CIÊNCIA DA COMPUTAÇÃO

PADRÕES ARQUITETURAIS PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIAGENTE

GEOVANE BEZERRA DA SILVA JUNIOR

São Luis 2003

GEOVANE BEZERRA DA SILVA JUNIOR

PADRÕES ARQUITETURAIS PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIAGENTE

Dissertação de Mestrado submetida à Coordenação do curso de Pós-Graduação em Engenharia de Eletricidade da UFMA para obtenção do título de Mestre em Ciência da Computação.

Orientadora: Prof. Dra. Maria del Rosário Girardi

São Luís

2003

PADRÕES ARQUITETURAIS PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIAGENTE

GEOVANE BEZERRA DA SILVA JUNIOR

DISSERTAÇÃO APROVADA EM __ / __ / 2003

Profa. Dra. Rosário Girardi

(Orientador)

Prof. Dr. Evandro de Barros Costa

(Membro da Banca Examinadora)

Prof. Dr. Zair Abdelouahab

(Membro da Banca Examinadora)

PADRÕES ARQUITETURAIS PARA O DESENVOLVIMENTO DE APLICAÇÕES MULTIAGENTE

MESTRADO

Área de Concentração: CIÊNCIA DA COMPUTAÇÃO

GEOVANE BEZERRA DA SILVA JUNIOR

Orientador: Dra. Rosário Girardi

Curso de Pós-Graduação

Em Engenharia de Eletricidade da

Universidade Federal do Maranhão

DEDICATÓRIA

Aos meus pais, fontes de inspiração para a realização de todos os meus sonhos.

AGRADECIMENTOS

Em primeiro lugar, à Deus, ser maior e supremo, razão da minha existência.

Aos meus pais, Geovane e Maria das Graças, pela oportunidade que me proporcionaram de estar realizando mais este sonho.

À Profa. Dra. Rosário Girardi, minha orientadora e amiga, pelos ensinamentos e segura orientação durante o desenvolvimento da pesquisa.

À minha noiva, Patrícia Nunes, pela compreensão e força.

A todos os meus familiares, pelo apoio e incentivo.

Aos meus amigos, Roberto Pires, Bruno Valente, Ednaldo Santos em especial pelas palavras de estímulo.

Aos meus companheiros de curso, em especial, Edil, Paulino, Luís Carlos, Carla, Wagner e Ismênia, pelos momentos de estudo, angústia, nervosismo, alegria, descontração e, principalmente, muita amizade no decorrer do curso.

A toda Coordenação do Mestrado, coordenadora, funcionários e professores, pelos bons serviços oferecidos e que foram fundamentais para a conclusão do mestrado.

Enfim, a todas as pessoas que, direta ou indiretamente, contribuíram para a concretização deste sonho e, conseqüentemente, para o meu futuro como um bom profissional em Ciência da Computação.

RESUMO

Este trabalho propõe uma coleção de padrões arquiteturais para o desenvolvimento de sistemas multiagente. Os principais problemas arquiteturais tais como comunicação, cooperação e mecanismo de coordenação entre os agentes são analisados e descritos em cada padrão. A geração de novos padrões pela extensão ou composição dos padrões propostos também é abordada.

A metodologia utilizada para extração de padrões está baseada no estudo de arquiteturas de sistemas multiagente freqüentemente utilizadas, como as arquiteturas quadro-negro e federativas e nos mecanismos de cooperação e coordenação geralmente identificados em tais arquiteturas, como os mecanismos mestre-escravo, reunião e negociador.

A descrição dos padrões está baseada em AUML e KQML. Os principais diagramas da AUML, como o diagrama de pacotes, o diagrama de agente e o diagrama de interação são usados para representar a estrutura e o comportamento da sociedade. Nesses diagramas, a representação das interações entre os agentes é feita com a utilização das *performatives* de KQML.

Os padrões propostos são validados através da construção de três estudos de caso relacionados ao desenvolvimento de sistemas multiagente para recuperação e filtragem de informações. Nesses estudos de caso, são analisadas as arquiteturas RETSINA, AMALTHAEA e ABARFI e é identificado a potencial reutilização ou aplicação dos padrões propostos.

Palavras-chave: Padrões de Software, Reutilização de Software, Agentes de Software, Arquiteturas de Software, KQML e AUML.

ABSTRACT

This work proposes a collection of architectural patterns for the development of multi-agent systems. Main architectural concerns like communication, cooperation and coordination mechanisms between the agents of the society are particularly analyzed in each described pattern. The generation of new patterns through the extension or composition of the proposed ones is also approached.

The methodology for pattern extraction is based on the study of frequently used architectures of multi-agent systems, like blackboard and federative architectures, and on mechanisms of cooperation and coordination usually identified in such architectures, like master-slave, meeting and negotiator mechanisms, as well.

Pattern description is based on AUML and KQML. Main AUML diagrams, like agent packages, agent diagrams and agent interaction diagrams are used to represent the structure and behavior of the society. In those diagrams, agent interactions are represented as KQML performatives.

Proposed patterns are validated through the construction of three case studies related to the development of multi-agent systems for information retrieval and filtering. In these case studies, the architectures RETSINA, AMALTHAEA and ABARFI are analyzed and the reuse or potential application of proposed patterns is identified.

Keywords: *Software Patterns, Software Reuse, Multi-agent Systems, Software Architectures, KQML, AUML, Software Design.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

ABARFI Arquitetura baseada em Agentes para a Recuperação e Filtragem de Informação

ACL Linguagem de Comunicação entre Agentes *("Agent Communication Language")*

BDI Crença-Desejo-Intenção ("*Belief-Desire-Intention")*

HTML Linguagem de Marcação de Hipertextos ("*Hypertext Markup Language")*

IAD Inteligência Artificial Distribuída

KQML Linguagem de Consulta e Manipulação Conhecimento *("Knowledge Query and Manipulation Language")*

MADS Metodologia baseada em Agentes para o Desenvolvimento de Software

RETSINA *("Reusable Task Structure-based Intelligent Network Agents")*

SGDB Sistema de Gerenciamento de Banco de Dados

SMA Sistema Multiagente

SRI Sistema de Recuperação de Informação

URL Recurso universal de localização *("Universal Resource Location")*

WKVG Gerador de Vetor de Palavra Chave Ponderado *("Weighted Keyword Vector Generator")*

CAPÍTULO 1

INTRODUÇÃO

CAPÍTULO 1 INTRODUÇÃO

1.1 Motivação

A melhoria na qualidade e a redução dos custos das soluções computacionais são os principais desafios dos desenvolvedores de software. Uma maneira de ajudar a cumprir esses objetivos é maximizando o reuso e disponibilizando produtos de software reutilizáveis, tais como padrões e frameworks. O reuso permite a redução do tempo e custo do desenvolvimento e incrementa a qualidade do software na medida que os desenvolvedores possam encontrar, utilizar e adaptar a novos contextos, aquelas soluções que já tenham sido provadas e usadas com sucesso. Pode se fazer reuso em um domínio de problema; de uma classe, de um componente, de um projeto, de uma idéia ou de um padrão.

Um padrão é uma solução que foi útil em um determinado contexto e provavelmente o será em outros. A descrição do padrão fornece as informações necessárias para sua reutilização através da declaração do problema, suas restrições, a representação de uma solução adaptável ao problema e uma discussão das conseqüências dessa solução.

Os padrões ajudam a minimizar a complexidade do software nas várias fases do ciclo de vida. Capturam idéias e soluções obtidas através da experiência e as tornam acessíveis a desenvolvedores menos experientes. Os principais benefícios que eles podem trazer são a capacidade de capturar conhecimento e experiências de desenvolvimento, bem como a facilidade de manutenção e compreensão dos sistemas documentados com padrões.

As etapas de análise e projeto são fundamentais para o desenvolvimento de software. Na fase de análise estuda-se e especifica-se o domínio do problema dentro do contexto dos objetivos e metas pré-estabelecidas, buscando compreender os detalhes e complexidade do problema. No projeto, especifica-se uma solução computacional para o problema. Em particular, podemos encontrar algumas soluções de projeto bem-sucedidas que podem ser reutilizadas. Porém, a identificação dessas soluções, e sua associação a um problema em particular e sua representação em forma de padrões é uma tarefa complexa.

O paradigma de desenvolvimento orientado a objetos tem facilitado e promovido à prática do reuso de software. Os avanços na construção de padrões para esse paradigma refletem-se em várias propostas tais como, o sistema de padrões criado por Buschmann [4], o catálogo de padrões criado por Gamma [15] e a coleção de padrões especificada por Vlissides

[50]. A utilização desses padrões tem auxiliado os desenvolvedores na construção de componentes reutilizáveis complexos como é o caso dos *frameworks*.

A tecnologia orientada a agentes veio complementar a orientação a objetos, introduzindo uma nova abstração de software com propriedades, tais como: autonomia, habilidade social e aprendizagem, que facilitam o desenvolvimento de arquiteturas de software complexas. Portanto, a identificação e especificação de padrões baseado em agente são fundamental para auxiliar na construção de sistemas multiagente de qualidade.

Este trabalho é realizado no contexto do projeto MaAE ("Multi-agent Application Engineering") que procura a construção de abstrações reutilizáveis de alto nível para o desenvolvimento de sistemas multiagente [16],[17],[18].

1.2 Objetivo do trabalho

Com essa motivação, pretende-se analisar as propostas existentes de padrões de projeto para o desenvolvimento de sistemas multiagente e, a partir das experiências de desenvolvimento adquiridas no contexto projeto MaAE, extrair e propor uma coletânea de padrões arquiteturais para construção de frameworks conceituais multiagente.

1.3 Estrutura da dissertação

Esta dissertação encontra-se estruturada em seis capítulos.

O segundo capítulo apresenta uma síntese do estado da arte dos padrões para o projeto de software de forma a facilitar a compreensão, análise e o agrupamento da coletânea de padrões que é proposta neste trabalho.

O terceiro capítulo introduz uma visão geral do projeto de sistemas multiagente, onde são abordados os diferentes níveis arquiteturais (global e detalhado) e as arquiteturas comumente utilizadas que compõem esses níveis. É descrita a linguagem AUML (Linguagem de Modelagem Unificada para Agentes) utilizada para a descrição dos padrões propostos. A KQML também é objeto de análise pela sua utilização na definição dos processos de coordenação e cooperação nos padrões propostos.

O quarto capítulo descreve os padrões identificados segundo a visão da tecnologia orientada a agentes na forma de uma coletânea de padrões, enfatizando os mecanismos de cooperação e coordenação dos agentes. Também, é explorada a capacidade de evolução dos padrões propostos.

No quinto capítulo são apresentados três estudos de casos desenvolvidos para validar os padrões encontrados. Nestes estudos de caso trabalhou-se com três arquiteturas baseadas em agentes para recuperação e filtragem de informações (RETSINA, AMALTHAEA, ABARFI), onde foram identificados alguns dos padrões propostos.

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

CAPÍTULO 2

PADRÕES DE SOFTWARE

CAPÍTULO 2 PADRÕES DE SOFTWARE

2.1 Introdução

Os padrões de software são de fundamental importância para a reutilização e projeto de software. No projeto de software se define uma solução computacional a um problema especificado durante a análise de requisitos. A reutilização de padrões na construção dessa solução de projeto, tanto no nível arquitetural como no nível detalhado, é fundamental para atender os requisitos de qualidade e produtividade do desenvolvimento de software.

Neste capítulo, são abordados os padrões de software de maneira especifica, detalhando os atributos, qualidades de um padrão, as diversas categorias e as formas nas quais os padrões podem ser agrupados. Essa abordagem detalhada permite uma uniformização dos diversos conceitos relacionados aos padrões de software, tais como, formas de agrupar os padrões e critérios de descrição de um padrão.

A criação e o reuso de padrões também é discutido neste capítulo. São abordadas as técnicas para de identificar padrões e desenvolver sistemas usando padrões. Essas técnicas irão auxiliar a construção de uma coletânea de padrões e orientar a construção de frameworks usando padrões.

Os conceitos de padrões abordados nesse capítulo são oriundos da tecnologia orientada a objetos.

2.2 Principais conceitos

2.2.1 Reutilização de software

O objetivo da reutilização de software é permitir que qualidade, baixo custo e rapidez no desenvolvimento sejam atingidos no desenvolvimento de software. A reutilização também promove a prática da prototipação e simplifica tanto a manutenção corretiva como evolutiva [19].

Os elementos a serem reutilizados podem ser produtos (especificações de requisitos, padrões, frameworks e código fonte), bem como conhecimento do processo de desenvolvimento, por exemplo, experiências na construção de aplicações em um determinado domínio de problema.

O paradigma de orientação a objetos trouxe contribuições fundamentais no suporte às atividades associadas à reutilização, tais como: encapsulamento, polimorfismo, herança.

O conhecimento adquirido na Engenharia do Software é vasto, e é grande a experiência acumulada na resolução de problemas típicos de desenvolvimento. Essa experiência estende-se desde as técnicas de programação até as metodologias de análise, projeto, teste e controle de qualidade. Como podemos aproveitar essas experiências? Capturando-las e reutilizando-as.

Entretanto, é necessário destacar algumas das principais dificuldades na reutilização do software:

- Necessita de esforço extra: a construção de software reutilizável exige maior esforço que o desenvolvimento de software específico;
- Exige boas abstrações: exige uma abstração sobre os detalhes não-essenciais e a identificação das abstrações corretas, o que não é fácil;
- Requer características difíceis de conseguir: flexibilidade, modularidade são qualidades que facilitam a reutilização, mas nem sempre se atingem facilmente;

O desenvolvimento de software baseado na reutilização tem duas abordagens: o desenvolvimento "para" reuso e o desenvolvimento "com" reuso. No desenvolvimento para reuso, os artefatos reutilizáveis são produzidos com o objetivo de serem posteriormente reutilizados, deverão se encaixar em uma arquitetura, seguir regras e padrões de elaboração. No desenvolvimento com reuso, supõe-se existir um conjunto de artefatos disponíveis que serão utilizados para compor um sistema.

Este trabalho propõe a construção de padrões de software. Portanto, aborda as atividades de desenvolvimento "para" reuso e seu agrupamento em uma coletânea auxiliara a construção de aplicações específicas no desenvolvimento "com" reuso.

2.2.2 O projeto de software

O projeto de software é a atividade realizada durante o desenvolvimento do software onde ocorre a definição de uma solução computacional a um problema especificado na fase prévia de análise de requisitos, com detalhe suficiente para permitir sua realização.

Fundamentalmente, o projeto de software é a atividade que envolve uma sistemática de decomposição da solução, a começar pela descrição em mais alto nível dos principais elementos do sistema e, em seguida, criando uma visão mais detalhada de como as características e funções deste sistema deverão estar integradas [55]. O resultado da aplicação dos padrões arquiteturais no projeto de software são as arquiteturas de software.

2.2.2.1 Arquiteturas de software

As arquiteturas provêm uma visão global do sistema, permitindo assim observar se o sistema satisfaz certos requisitos e sugere um modelo para a construção e composição dos sistemas [53].

A arquitetura de um software identifica um conjunto de componentes que colaboram para atingir os propósitos do sistema. A arquitetura especifica as propriedades externamente visíveis dos componentes e define as ligações entre os mesmos [2].

Segundo Shaw [42], a arquitetura de um software é a descrição de elementos, padrões que guiarão suas composições, e limitações destes padrões. Em geral, um sistema é definido em termos de uma coleção de componentes e interações entre estes componentes. Tal sistema pode, ainda, ser usado como elemento de composição em projeto de sistema maior.

2.2.2.2 Frameworks

Um framework é definido por Coad [5] como um esqueleto de classes, objetos e relacionamentos agrupados para construir aplicações específicas e por Johnson [27] como um conjunto de classes abstratas e concretas que provê uma infra-estrutura genérica de soluções para um conjunto de problemas.

As arquiteturas reutilizáveis formadas da implementação e composição de um conjunto de padrões arquiteturais são chamadas de frameworks [21].

2.3 Padrões de Software

Os padrões e a reutilização estão intimamente ligados uns aos outros, pois os padrões dão suporte necessário para a construção de elementos de software maiores, como é o caso dos frameworks.

A origem dos padrões deu-se com o trabalho feito pelo arquiteto Christopher Alexander no final dos anos 70. Ele escreveu dois livros: *"A Pattern Language"* e *"A Timeless Way of Building"* que, além de exemplificarem, descrevem seu método para

documentação de padrões. O trabalho de Alexander, apesar de ser voltado para a construção civil, possui uma fundamentação básica que pode ser abstraída para a área de software [32].

Um padrão é uma solução comum bem sucedida para um problema também comum. Todavia, em um padrão identificamos não só a solução, como também onde e em que contexto deve aplicar-se essa solução, podendo ser considerado como um par "problema/solução". Os padrões são de grande ajuda na abordagem da complexidade do software nas diversas fases do ciclo de vida [4],[37]. Os projetistas familiarizados com certos padrões podem aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los [15].

2.3.1 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 [1]:

- **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;
- **Problema:** estabelece o problema a ser resolvido pelo padrão, descreve a intenção e objetivos do padrão perante o contexto e forças específicas;
- **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;
- **Forças:** descreve os impactos, influências, restrições relevantes ao problema e como estes elementos interagem ou são conflitantes entre si;
- **Solução:** São instruções que descrevem como o problema é resolvido, podendo para isso utilizar texto, diagramas e figuras. A notação gráfica utilizada neste trabalho é a AUML (Linguagem de Modelagem Unificada baseada em Agente) [35].

2.3.2 Qualidades e benefícios de um padrão

Um bom padrão possui qualidades tais como [37]:

- **Abstração:** cada padrão abstrai um problema bem definido e sua solução em um domínio em particular;
- **Flexibilidade:** cada padrão pode integrar-se com outros padrões para resolver um problema maior em uma variedade de situações;
- **Capacidade de geração e composição:** um padrão, uma vez aplicado, gera um contexto resultante que pode coincidir com o contexto inicial de outros padrões.

Podemos encontrar alguns benefícios na utilização de padrões [37]:

- Os padrões capturam conhecimento e experiência;
- Com os nomes dos padrões podem-se criar vocabulários que ajudam na comunicação entre os desenvolvedores;
- Os sistemas documentados com padrões facilitam a sua reestruturação e manutenção.

2.3.3 Anti-padrões

Os anti-padrões descrevem soluções para não ser usadas e erros comuns para não serem cometidos. Eles representam uma "lição aprendida".

Segundo Maldonado [32] um anti-padrão é:

- Um padrão que diz como ir de um problema para uma solução ruim;
- Um padrão que diz como ir de uma solução ruim para uma solução boa.

A primeira parece inútil, exceto se considerarmos o padrão como uma mensagem "Não faça isso". O segundo é definitivamente um padrão positivo, no qual o problema é a solução ruim.

2.3.4 Categorias de padrões

Os padrões abrangem diferentes tipos de abstração e podem ser classificados em diversas categorias. Alguns padrões ajudam a estruturar um sistema em subsistemas, outros refinam esses subsistemas e suas relações [4]. Em seguida apresentaremos as principais categorias de padrões:

- **Padrões de análise:** apresentam soluções para problema de análise de sistemas, que são encontrados em um domínio específico;

- **Padrões de arquitetura:** apresentam o esquema ou organização estrutural fundamental de sistemas de software;
- **Padrões de projeto:** refina os componentes de uma arquitetura de software e descrevem soluções para problemas de projeto de software. Esses padrões fornecem uma especificação explicita da interação de classes e objetos, melhorando a documentação e a manutenção de sistemas;
- **Padrões de programação:** descrevem soluções de programação particulares de uma determinada linguagem ou regras gerais de estilo de programação.

Dentre as categorias apresentadas, este trabalho aborda os padrões no nível arquitetural. Um padrão arquitetural que tem o objetivo de apoiar o desenvolvimento de sistemas orientados a objetos, através da ênfase na organização estrutural do software [4]. Cada padrão fornece um conjunto de subsistemas pré-definidos, especifica suas responsabilidades, e inclui regras e orientações para a organização dos relacionamentos entre os subsistemas.

Os padrões descrevem os princípios fundamentais das arquiteturas em um sistema de software [4]. Portanto, os padrões arquiteturais possibilitam soluções para o projeto de software tanto ao nível de sistema (global), quanto ao nível de subsistema (detalhado).

2.3.5 Coletâneas de padrões

As coletâneas de padrões têm o objetivo de desenvolver formas de agrupar os padrões segundo um critério. Os padrões podem ser encontrados em diferentes níveis de abstração e diferentes domínios. As coletâneas podem ser divididas em: coleções de padrões, catálogos de padrões e sistemas de padrões.

2.3.5.1 Coleções de padrões

Uma coleção de padrões é uma coletânea qualquer com padrões que não possuem nenhum vínculo entre si e, em geral, não possui nenhuma padronização no formato de apresentação. A seguir destacaremos uma coleção de padrões ilustrados na **Tabela 1**. São padrões escritos por diversos autores e de maneira independente. Apresentados em um livro onde se agruparam os padrões que pertencem ao mesmo domínio ou fase do processo de desenvolvimento em capítulos [50].

Capítulo	Padrão	Autor
2-Padrões de propósitos gerais	Command Processor	Peter Sommerlad
	Shoper	Jim Doble
3- Padrões de propósitos específicos	A Structural Pattern for Designing Transparent Layered Services	Aamod Sane e Roy Campbell
	Patterns for Processing Satellite Telemetry with Distributed Teams	Stephen Berezuk
	A pattern Language for Object-RDMS Integration	Kyle Brown e Bruce Whitenack
	Transactions and Accounts	Ralph Johnson
4- Padrões de Exposição	Patterns for Classroom Education	Dana Anthony
	A Pattern Language for the Preparation of Software Demonstrations	Todd Coram
	A Pattern Language for Essay-Based Web Site	Robert Orenstein

Tabela 1-Exemplo de Coleção de Padrões [50].

2.3.5.2 Catálogos de padrões

Segundo Appleton [1], um catálogo de padrões é uma coletânea de padrões relacionados, talvez relacionados apenas fracamente ou informalmente e pode oferecer um esquema de classificação e recuperação. A **Tabela 2**, exibe um exemplo de um catálogo de padrões [15]. Classificados por: escopo e propósito.

		Propósito		
		De Criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweygth Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Tabela 2- Exemplo de catálogo de padrões [15].

No critério “escopo” decide-se se o padrão atua sobre uma classe ou sobre objetos da classe. Os Padrões de classe lidam com os relacionamentos entre classes e suas subclasses, por meio do uso de herança, sendo, portanto estabelecidos de forma estática (em tempo de

compilação). Padrões de objeto lidam com relacionamentos entre objetos, que podem ser modificados durante a execução e são mais dinâmicos. A maioria dos padrões utiliza herança de alguma forma. Portanto os únicos padrões classificados na categoria "classe" são os que se concentram em relacionamentos de classes.

Segundo o critério "propósito" distinguem-se padrões de criação, padrões estruturais e padrões comportamentais. Os padrões de criação concentram-se no processo de criação de objetos. Os padrões estruturais lidam com a composição de classes ou objetos .Os padrões comportamentais caracterizam as formas pelas quais classes ou objetos interagem e distribuem responsabilidades.

2.3.5.3 Sistemas de padrões

Um sistema de padrões é um conjunto coeso de padrões co-relacionados que trabalham juntos para apoiar a construção e evolução de arquiteturas completas. A **Tabela 3** apresenta os padrões de um sistema segundo dois critérios para classificação: categoria de padrões e categoria de problemas [4].

	Padrões arquiteturais	**Padrões de projeto**	**Idiomas**
Da lama para a estrutura	Layers Pipes and Filters Blackboard		
Sistemas Distribuídos	Brokers Pipes and Filters Microkernel		
Sistemas Interativos	MVC PAC		
Sistemas Adaptáveis	Microkernel Reflection		
Decomposição da Estrutura		Whole-Part	
Organização do Trabalho		Master-Salve	
Controle de Acesso		Proxy	
Gerenciamento		Command Processor View Handler	
Comunicação		Publisher-Subscriber Forwarde-Receiver	
Manuseio de Recursos			Counted Pointer

Tabela 3- Exemplo de sistema de padrões [4].

As categorias de padrões são: padrões arquiteturais, padrões de projeto e idiomas. Os idiomas são usados na fase de implementação para transformar a arquitetura de software em um programa escrito numa linguagem específica [4].

As categorias de problemas servem para agrupar os padrões que fazem parte do seu sistema:

- **Da lama para a estrutura *("From mud to structure"):*** estes padrões apóiam a decomposição adequada de uma tarefa do sistema em sub-tarefas que cooperam entre si;
- **Sistemas distribuídos *("Distributed systems")*:** estes padrões fornecem infra-estrutura para sistemas que possuem componentes localizados em processadores diferentes ou em diversos sub-sistemas e componentes;
- **Sistemas interativos *("Interactive systems")*:** estes padrões ajudam a estruturar sistemas com interface homem-máquina;
- **Sistemas adaptáveis *("Adaptable systems")*:** estes padrões fornecem infra-estruturas para a extensão e adaptação de aplicações em resposta a requisitos de evolução e mudança funcional;
- **Decomposição estrutural *("Structural decomposition"):*** estes padrões apóiam a decomposição adequada de sub-sistemas e componente complexos em partes cooperativas.

2.4 Criação e reuso de padrões

2.4.1 Identificação de padrões

Uma das dificuldades encontrada na formação de um padrão é justamente a sua derivação, como um padrão é descoberto? Para responder essa pergunta Buschmann [4], apresenta algumas regras de derivação de novos padrões, também conhecida como *"Pattern mining"* (extração de padrões). São elas:

1. Encontre pelo menos três exemplos nos quais um problema é resolvido efetivamente usando a mesma solução;

2. Extraia a solução, o problema e as influências;

3. Declare a solução como candidata a padrão;

4. Execute um "*writer*'s *workshop*" para melhorar a descrição do candidato e compartilhá-lo com outros;

5. Aplique o candidato a padrão em um outro projeto de desenvolvimento de software;

6. Declare o candidato a padrão como padrão se sua aplicação for bem sucedida. Caso contrário tente procurar uma solução melhor.

Na **Figura 1**, temos o que representa o "*pattern mining*" e o ciclo de vida de um padrão [30]. Nessa figura, observamos que os padrões são extraídos a partir de aplicações prontas e bem sucedidos, em seguida podemos agrupar esses padrões de forma a solucionar um problema maior, surgindo assim os frameworks que mais tarde será usado para formar uma aplicação completa.

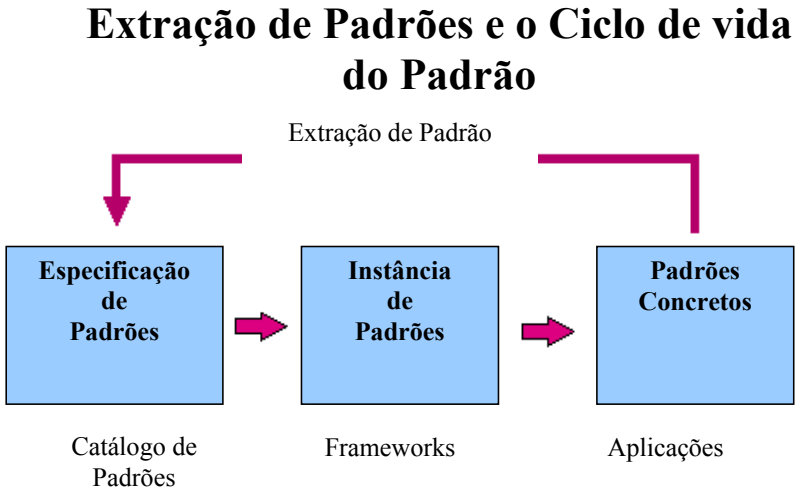

Figura 1-Extração de padrões e o ciclo de vida de um padrão [30].

2.4.2 Desenvolvimento de sistemas usando padrões

O uso de padrões não veio criar um novo método de desenvolvimento de software, mas sim facilitar e complementar o desenvolvimento nas fases de criação do software (análise e projeto). Algumas heurísticas para o desenvolvimento de um sistema usando padrões de software são [4]:

1. Utilize seu método preferido para o processo de desenvolvimento de software em cada fase de desenvolvimento;

2. Utilize um sistema de padrões adequado para guiar o projeto e implementar soluções para problemas específicos, isto é, sempre que encontrar um padrão que resolva um problema de projeto presente no sistema, utilize os passos de implementação associados a esse padrão. Se esses se referirem a outros padrões, aplique-os recursivamente;

3. Se o sistema de padrões não incluir um padrão para seu problema de projeto, tente encontrar um padrão em outras fontes conhecidas;

4. Se nenhum padrão estiver disponível, aplique as diretrizes de análise e projeto do método que você está usando. Essas diretrizes fornecem pelo menos algum apoio útil para resolver o problema de projeto em mãos.

2.4.3 Construção de frameworks usando padrões

Os padrões podem ser vistos como descrições abstratas de frameworks que facilitam o reuso de arquiteturas de software. Todavia, os frameworks podem ser vistos como materialização das abstrações. Então, quando construímos um framework estamos transformando o conhecimento e experiência dos padrões em software semipronto [21].

CAPÍTULO 3

O PROJETO DE SOFTWARE BASEADO EM AGENTE

CAPÍTULO 3 O PROJETO DE SOFTWARE BASEADO EM AGENTE

3.1 Introdução

A prática no desenvolvimento de sistemas multiagente vem aumentando devido às vantagens do paradigma de agente em abordar a complexidade do software. A disponibilidade de várias linguagens para a construção de aplicações segundo este paradigma, tais como a AUML [35], tem ajudado a reduzir a distância entre o mundo real e o computacional.

Este capítulo aborda o desenvolvimento do projeto de software considerando dois níveis de abstração: o nível global que trata dos mecanismos de coordenação e cooperação entre os diversos agentes da sociedade e o nível detalhado referente à estrutura de funcionamento interno dos agentes.

Porém, é dado maior destaque as arquiteturas no nível global devido a sua importância na identificação e extração de padrões arquiteturais. O estudo das arquiteturas no nível global é baseado nos os mecanismos de coordenação e cooperação utilizados. Esses mecanismos serão abordados na descrição dos padrões arquiteturais propostos.

No projeto de sistemas multiagente são abordados as fases de desenvolvimento de um SMA (Sistema Multiagente), bem como a AUML, utilizada na especificação da estrutura dos padrões propostos.

Para complementar o estudo das arquiteturas no nível global é apresentada uma síntese da linguagem KQML utilizada para ilustrar a coordenação e cooperação entre os agentes na descrição dos padrões.

3.2 Níveis arquiteturais

A arquitetura de um sistema multiagente representa os tipos de agentes no sistema e a forma como eles interagem. Essas arquiteturas podem ser analisadas considerando dois níveis de abstração [10],[12]:

- *nível global,* que trata a questão da coordenação e cooperação dos diversos agentes da sociedade. Esses mecanismos podem ser representados pela configuração dos componentes que constituem o sistema e pelas conexões que coordenam as atividades entre os componentes. A **Figura 2** ilustra o comportamento dos agentes vivendo em sociedade;

- *nível detalhado,* que trata da estrutura interna do agente. Mostra como ele é implementado em relação às suas propriedades, à sua estrutura e como os módulos que o compõem interagem, garantindo sua funcionalidade. Na **Figura 2**, também, podemos observar que cada agente pode ser dividido em partes que estruturam o comportamento interno de cada agente.

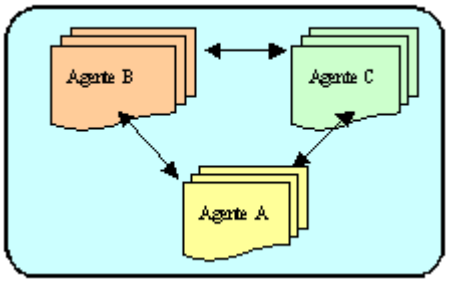

Figura 2- Representação de agentes vivendo em sociedade.

O desenvolvimento de padrões arquiteturais requer uma compreensão adequada das arquiteturas dos sistemas multiagente no nível de abstração global. Para isso, é preciso identificar os diferentes mecanismos de cooperação e coordenação que caracterizam um sistema multiagente.

3.3 A abstração *Agente*

Nas arquiteturas de sistemas multiagente, a abstração de software utilizada é o agente. Uma adequada compreensão dessas arquiteturas, apresentadas na seção 3.6.3, faz necessária uma explicação sobre o que é um agente e como ele se comporta.

O conceito de "agente" é uma metáfora utilizada em diversas áreas do conhecimento, desde a Psicologia, Sociologia, Biologia até a Ciência da Computação, principalmente na área de Inteligência Artificial (IA), cujos pesquisadores foram os primeiros a utilizar a noção de agente em seus trabalhos, na medida em que tentavam reproduzir uma entidade artificial que imitasse as habilidades humanas [9].

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 vistas e dependem muito das características e funcionalidades apresentadas pelo agente em questão. A seguir são apresentadas algumas definições geralmente apresentadas na literatura.

Um agente é uma entidade computacional que age em nome de outras entidades de forma autônoma, executa suas ações com algum nível de pró-atividade e/ou reatividade e exibe algum nível de aprendizado, cooperação e mobilidade [20]

Um agente é uma entidade computacional que está situado em algum ambiente e que é capaz de realizar ações autônomas neste ambiente visando atingir seus objetivos [54].

Pode-se, ainda, definir um agente como sendo uma entidade que percebe seu ambiente através de sensores e atua neste ambiente através de executores (**Figura 3**). Por exemplo, nos agentes humanos, os olhos e ouvidos, são sensores, e as mãos e a boca são executores. A **Figura 3** demonstra um agente genérico [40].

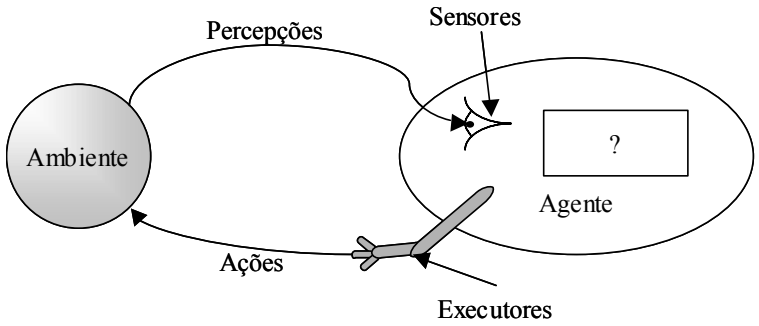

Figura 3-Interação de agentes com o ambiente através de sensores e executores [40]

Segundo Wooldridge e Jennings [54], a construção e a especificação da estrutura e funcionamento de um agente genérico pode ser realizada segundo em três tipos de arquiteturas:

- **Arquiteturas deliberativas:** Segue 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 em utilidade na classificação de Russel [40].
- **Arquiteturas Reativas:** A arquitetura reativa é aquela que não incluem nenhum tipo de modelo central e simbólico do mundo e não utiliza raciocínio complexo e simbólico. Baseia-se na proposta de 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 na classificação de Russel [40].

- **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 ocupar-se de raciocínios complexos [40].

3.4 Características

Os agentes podem ser identificados por possuírem as seguintes características principais [20],[22],[54]:

- **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;
- **Aprendizagem:** um agente é capaz de aprender a partir das ações realizadas, considerando sucessos e fracassos, de forma a melhorar seu desempenho.

Para alguns pesquisadores, mais precisamente para aqueles que trabalham na área da Inteligência Artificial, o termo agente tem um significado bem mais complexo do que o apresentado pelas características anteriores. De forma geral, eles descrevem um agente como um sistema de computação que, além de apresentar as características mencionadas acima, tem a capacidade de apresentar a idéia de estados mentais, que utiliza conceitos aplicados a seres humanos tais como crenças, desejos e intenções, entre outros.

3.5 Sistemas Multiagente

Um Sistema Multiagente (SMA) pode ser definido como uma sociedade de agentes que interagem entre si e 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 [20]:

- 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;
- Prover extensibilidade, permitindo alterar o número de processadores dedicados a um problema;
- Oferecer clareza e simplicidade conceitual ao projeto da sociedade.

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 entre si, coordenar suas atividades e negociar para solucionar 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.

Para a sua interação e troca de informações, os agentes devem ser providos de alguma linguagem de comunicação, ACL – (Linguagem de Comunicação entre Agentes). Na seção 3.7 é detalhado o estudo da ACL utilizada neste trabalho.

3.6 O Projeto de Sistemas Multiagente

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

- O projeto arquitetural do SMA para a construção da arquitetura de software do sistema que estabelece os mecanismos de comunicação entre os agentes da sociedade;

- O projeto detalhado de cada agente da sociedade baseado na construção de agentes, segundo a abordagem reativa, deliberativa ou híbrida;

Odell, Parunak e Bauder [35] desenvolveram a AUML (Linguagem de Modelagem Unificada para Agentes) para especificar, visualizar, documentar e construir sistemas baseados em agentes. A AUML surgiu como extensão da UML [24] (Linguagem de Modelagem Unificada) tendo em vista suas limitações para modelagem de agentes e sistemas baseados em agentes. Sabemos que a UML tem tido uma larga aceitação na modelagem de software orientado a objetos. Nada mais coerente que tentar expandir a UML na utilização de agentes (AUML), considerando que um agente, portanto pode ser considerado como a extensão de um objeto. Logo, essa extensão levou em consideração as exigências e as distinções da tecnologia de agentes, surgindo então a AUML.

Existem três níveis de apresentação do projeto de um sistema multiagente em AUML [35]:

- **Nível 1**: representação global do sistema (projeto arquitetural);
- **Nível 2**: representação das interações entre os agentes (projeto arquitetural);
- **Nível 3:** representação do processamento interno do agente (projeto detalhado).

A escolha da AUML para modelagem das estruturas dos padrões propostos se deu em virtude dessa linguagem de modelagem abordar o projeto de sistema multiagente, também em diferentes níveis de abstração (arquitetural e detalhado), sendo assim, fundamental para a melhor descrição dos padrões propostos.

3.6.1 Nível 1 do projeto em AUML

No nível 1 se especifica uma solução que pode ser reutilizável para outros sistemas. Para expressar a solução são utilizados dois diagramas, chamados de *pacotes* e "*templates"*.

Os pacotes são mecanismos de propósito gerais utilizados para organizar elementos do modelo em grupos, podendo estar inseridos dentro de um outro pacote. Os pacotes são utilizados com bastante eficácia para tratar uma grande funcionalidade de um sistema. Por exemplo, a **Figura 4** mostra o relacionamento entre dois pacotes (comprador e fornecedor). O pacote comprador é composto pelo agente corretor e o varejista. O fornecedor, pelo agente atacadista e pelo agente varejista que também faz parte do pacote comprador. A

interação entre os pacotes inicia com o agente corretor fazendo um pedido de proposta para o agente varejista, que faz uma consulta ao atacadista, que informa ao varejista e em seguida encaminha proposta para o corretor.

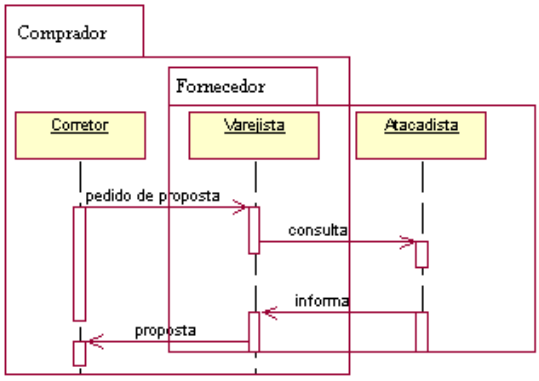

Figura 4- Exemplo da utilização de pacotes com os agentes interagindo [35]

Um "*template*" é um modelo de um pacote cujos elementos são parametrizados. Os parâmetros neste modelo são divididos em linhas horizontais em três categorias: parâmetros de regra, restrição e atos de comunicação. Por exemplo, a **Figura 5** mostra o pacote comprador com seus elementos organizados em forma de parâmetros. O corretor envia um pedido de proposta para o fornecedor, que responde com uma proposta obedecendo à restrição de dia e horário.

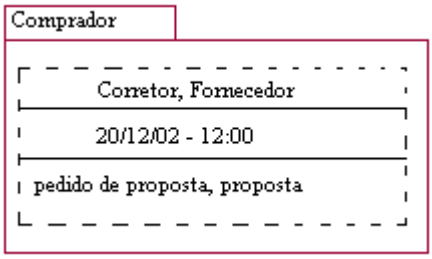

Figura 5-Exemplo de uma *Template* [35].

3.6.2 Nível 2 do projeto em AUML

No nível 2 mostra se como os agentes interagem entre si através da troca de mensagens. A seguir são apresentados alguns diagramas da UML, e suas utilidades na modelagem de sistemas baseados em agentes [35]:

- Diagrama de seqüência: mostra a seqüência cronológica das comunicações entre os agentes;
- Diagrama de colaboração: enfatiza as associações entre os agentes;

- Diagrama de atividades e diagrama de estado: mostram o fluxo do processamento da informação em um sistema multiagente.

3.6.2.1 O diagrama de seqüência em AUML

No diagrama de seqüência, a sintaxe utilizada para representar um agente (ou conjunto de agentes) é a seguinte: Nome do agente/Papel*:* Classe, onde classe se refere a que categoria de agente ele pertence; o papel identifica a "função específica" que um agente executa durante sua existência. Por exemplo: João/Empregado: Pessoa - onde João é o agente que desempenha o papel de Empregado e é instância da classe *Pessoa*. Outros exemplos da sintaxe são: Nome do agente/Papel, ou seja, João/Empregado; ou simplesmente Papel, ou seja, Empregado.

A **Figura 6** mostra que as interações entre os agentes utilizam um AC (ato de comunicação) para representar a comunicação entre o agente 1 e o agente 2, ao invés de uma "mensagem" típica da orientação a objetos como utilizada na UML.

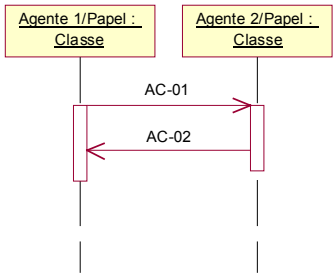

Figura 6- Exemplo da notação do diagrama de seqüência [35].

Outra extensão à UML se dá no suporte à interação concorrente, exemplificado na **Figura 7**, que não é muito empregado na orientação a objetos. A **Figura 7** mostra três formas de representar a interação múltipla:

(a): todas as ACs são enviadas ao mesmo tempo, e de forma concorrente;

(b): dependendo da decisão, nenhuma ou várias ACs são enviadas. Caso mais de uma sejam enviadas, elas o fazem de forma concorrente;

(c): ou-exclusivo - apenas uma AC é enviada.

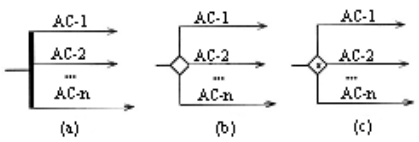

Figura 7- Extensões que dão suporte à interação concorrente [35].

A **Figura 8** mostra formas equivalentes de representar os três tipos de interações concorrentes. As barras verticais paralelas e em seqüência indicam os agentes recebedores dos atos de comunicação (AC). Um mesmo agente pode ser ativado por vários ACs e possuir instâncias do mesmo papel, ou instâncias de diferentes papéis ou ainda diferentes agentes recebedores podem ser ativados.

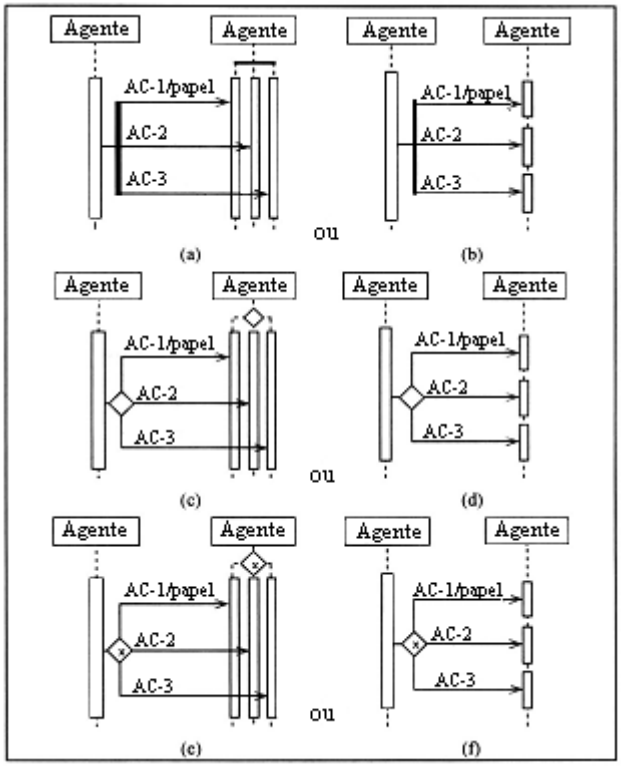

Figura 8- Exemplos de formas de representação de interações concorrentes [35].

3.6.2.2 O diagrama de colaboração em AUML

O diagrama de colaboração mostra informação equivalente ao diagrama de seqüência, ou seja, a ordem cronológica das comunicações entre os agentes; porém possui uma representação gráfica diferenciada (**Figura 9**). São características dos diagramas de colaboração:

- Os agentes (retângulos) podem estar dispostos em qualquer lugar no diagrama;
- A seqüência das interações é numerada;
- Pode fornecer maior clareza de entendimento;
- É semanticamente equivalente ao diagrama de seqüência.

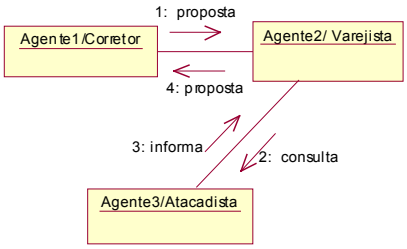

Figura 9- Exemplo da colaboração entre os agentes [35]

3.6.2.3 O diagrama de atividades na AUML

Os diagramas de atividades 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.

O exemplo da **Figura 10** mostra o protocolo utilizado no processamento de uma compra eletrônica. O agente Cliente faz um pedido. Em seguida, o agente Intermediador recebe o pedido e o processa. Depois o agente RC (Rede de Comércio) recebe o evento com o pedido e o aceita. O agente RC aguarda então a aceitação da quota pelo agente Distribuidor. Somente após a aceitação recebida é que o processamento do pedido é prosseguido. Após o pedido ter sido processado, os agentes requisitantes recebem eventos de notificação que disparam operações para conclusão do mesmo.

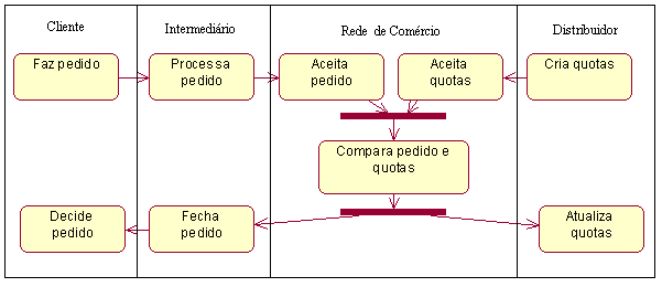

Figura 10-Exemplo de um diagrama de atividades [35].

3.6.2.4 O diagrama de estado na AUML

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

A **Figura 11** mostra um processo de pedido de compra realizado entre um agente A (cliente) e um agente B (fornecedor). O diagrama de estado indica os estados e as transições válidas no protocolo de pedido de compra.

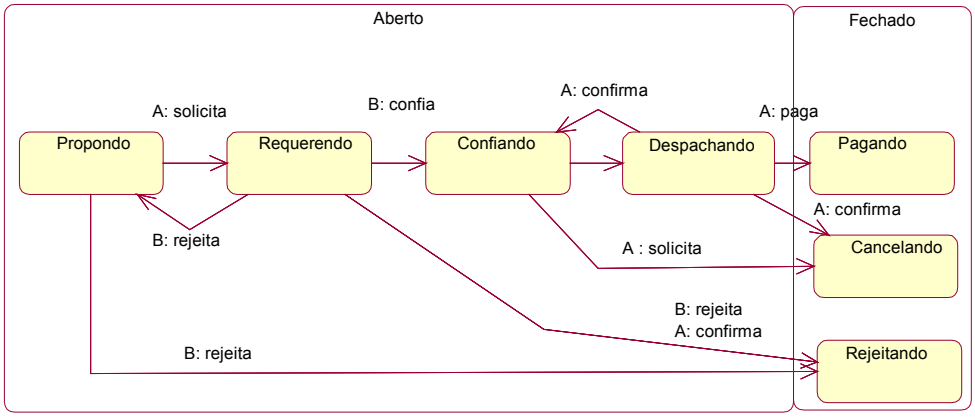

Figura 11-Exemplo do diagrama de estado [35].

O diagrama de estados não fornece uma visão centrada no processo de interação entre agentes; serve apenas como uma visão auxiliar, pois é focada nos estados permissíveis e fornece um mecanismo de verificação da integridade das mudanças de estado através de transições válidas.

3.6.2.5 O nível 3 do projeto em AUML

No nível 3 é abordado o comportamento interno dos agentes que é descrito em um nível mais baixo e detalhado Para a descrição do processamento interno de agentes utiliza-se a representação do nível 2, os diagramas de estado e de atividade [35].

3.6.3 Arquiteturas baseadas em agentes

A abordagem multiagente é adequada para a concepção e construção de sistemas computacionais complexos, distribuídos ou heterogêneos e são necessários padrões arquiteturais e frameworks para aumentar a produtividade e a qualidade das aplicações desenvolvidas segundo esse modelo. Entretanto, para a construção dessas abstrações reutilizáveis é preciso se ter uma noção bastante clara de como construir a arquitetura de um sistema multiagente. Por esta razão, abordaremos nesta seção, questões relevantes aos métodos do projeto arquitetural de sistemas multiagente, tais como os mecanismos de coordenação e cooperação utilizados e a forma em que se realiza comunicação entre os agentes da sociedade.

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

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, a arquitetura de um sistema multiagente pode ser classificada em três grupos [29]:

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

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

3.6.3.1 Mecanismo de coordenação

A coordenação entre agentes refere-se à maneira como os agentes estão organizados a fim de cooperar para alcançar um objetivo comum no sistema [10],[12].

Também, pode se conceituar a coordenação como o processo pelo qual um agente raciocina sobre suas próprias ações e as ações de outros agentes com o objetivo de garantir que a comunidade funcione de maneira coerente [26]. Algumas das razões pelas quais os agentes precisam ser coordenados são [26]:

- Prevenir o caos – onde existem vários indivíduos dando opiniões diferentes, a instituição uma anarquia se torna bem fácil;
- Respeitar restrições globais – sempre existem restrições globais as quais os agentes devem respeitar a fim de realizar seus objetivos;
- Distribuir tarefas, recursos ou informações – os agentes em uma sociedade multiagente possuem capacidades e conhecimentos especializados, bem como fontes, recursos, responsabilidades e limitações. Então, muitas das vezes, um agente sozinho não consegue resolver certos problemas;
- Dependência entre as ações dos agentes – os objetivos dos agentes são geralmente interdependentes;

- Eficiência – as informações descobertas por um agente podem ser de grande importância para outro agente, poupando este último do trabalho repetitivo e possibilitando sua realização de forma mais rápida.

A coordenação tenta maximizar as capacidades individuais de cada agente minimizando conflitos e pode ser realizada por um agente que tenha uma perspectiva ampla do sistema. Atuando como "coordenador", reuni informações sobre toda a sociedade, é responsável por criar planos e atribuir tarefas aos membros da sociedade. No entanto, esta não é uma abordagem prática em sistemas reais por ser muito difícil criar um agente que se mantenha informado sobre as intenções e crenças de todos os agentes da sociedade [26]. Também, devemos considerar o fato de que, nesta abordagem, uma falha do agente coordenador comprometeria o funcionamento de todo o sistema [26], apesar de que, neste caso, poderiam ser adotados mecanismos de tolerância as falhas onde outro agente poderia assumir o papel de coordenador.

Uma alternativa ao controle centralizado é a distribuição do controle entre vários agentes da sociedade. Porém, um problema desta abordagem é a manutenção da coerência global da sociedade sem um controle global explícito [23]. Neste caso, os agentes devem raciocinar a respeito das ações, mas também sobre o processo de coordenação em si [36]. Muitas pesquisas têm seu foco no desenvolvimento de comunidades nas quais o controle e os dados são distribuídos.

Uma desvantagem advinda da distribuição do controle e dos dados é a dificuldade de se ter conhecimento sobre o estado global do sistema, que está disperso através da comunidade, sendo que cada indivíduo possui uma visão parcial e imprecisa desta perspectiva.

Os principais requisitos para a coordenação são [23]:

- Capacidade de comunicação entre os agentes;
- Capacidade de negociação entre os agentes.

Moulin [34], considera três processos de coordenação fundamentais:

- **Ajuste mútuo** - forma mais simples de coordenação, pressupondo dois ou mais agentes que concordam em compartilhar recursos para atingir algum objetivo comum.

- **Supervisão direta** - prevê que uma relação já tenha sido estabelecida entre dois ou mais agentes na qual um agente mantém algum controle sobre outros.
- **Padronização** – estabelece uma relação entre dois ou mais agentes na qual um agente mantém o controle sobre os outros estabelecendo procedimentos padronizados a serem seguidos por seus supervisionados em determinadas situações.

Utilizando-se destes três processos de coordenação, é possível descrever mecanismos de coordenação sofisticados a partir do qual destacam-se [34]:

- **Hierarquias** - baseado no processo de supervisão direta, este mecanismo aglutina agentes em pequenos grupos coordenados por um supervisor, dispostos de forma hierárquica, onde um supervisionado pertencente a um grupo pode atuar como supervisor de outro. Este tipo de estratégia permite dividir um grupo grande em vários sub-grupos, distribuindo os fluxos de coordenação entre vários níveis de supervisão.
- **Mercados** - baseado no processo de ajuste mútuo, este mecanismo pressupõe que os agentes controlam recursos escassos e concordam em compartilhá-los com outros agentes para atingir um objetivo comum. A estes recursos são vinculados preços e, uma vez que um contrato foi fechado, há uma aceitação de que o comprador torna-se o supervisor do fornecedor.

O conceito de coordenação define aspectos gerais de interação entre agentes de forma a viabilizar a coesão entre seus comportamentos e ações em relação aos objetivos globais do sistema [23].

Paraíso [39] apresenta dois mecanismos para coordenar os agentes: arquitetura mestre-escravo e a arquitetura de mercado.

3.6.3.1.1 Arquitetura mestre - escravo

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

3.6.3.1.2 Arquitetura de mercado

Na arquitetura de mercado, todos os agentes estão em um mesmo nível e sabem as tarefas que cada agente é capaz de desempenhar. Esta arquitetura visa diminuir a quantidade de mensagens trocadas, tendo em vista que cada agente já conhece todos os outros. Baseado nesse mecanismo os agentes podem ser classificados como produtores e consumidores e são utilizados para maximizar lucros através de um processo de negociação [39].

3.6.3.2 Mecanismo de cooperação

Em um sistema multiagente à cooperação é indispensável, pois é através dela que os agentes colaboram com outros agentes a fim de realizar seus objetivos. A cooperação entre os agentes pode acontecer de duas formas [10],[12]:

- Partilha de tarefas: um agente pode necessitar de outros agentes para auxiliá-lo na realização de uma tarefa;
- Partilha de resultados: os agentes disponibilizam informações para a sociedade, prevendo que alguns outros agentes possam necessitar dessas informações em um determinado momento.

Faraco [10] considera que a cooperação acontece quando vários agentes planejam e executam suas ações de uma forma coordenada e sendo que é requerida quando:

- Um agente não consegue encontrar um plano local que contemple seu objetivo;
- O plano adequado ao objetivo envolve ações de outros agentes;
- O agente considera que um outro plano pode ser melhor (de menor custo ou mais eficiente) do que um plano local;

A cooperação também pode acontecer durante a fase de planejamento quando [10]:

- Os agentes encontram planos incompletos, que podem ser completados em cooperação com outros agentes; ou
- Um agente encontra um evento para o qual não está habilitado a responder, mas sabe que outros agentes estão.

Outros objetivos para a cooperação entre agentes são [34]:

- Diminuição do tempo de execução de uma tarefa através do paralelismo, no caso de partilha de tarefas;
- Aumento do escopo de tarefas executáveis através do compartilhamento de recursos;

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

3.6.3.2.1 Arquitetura quadro-negro

Em uma sociedade baseada na arquitetura quadro-negro (*blackboard*) os agentes não se comunicam entre si de maneira direta, mas sempre através de um quadro-negro. Este tipo de arquitetura não surgiu com o aparecimento dos sistemas multiagente, sendo utilizado antes por outros paradigmas [43].

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

Em sistemas multiagente, os quadros-negros são utilizados como um repositório de perguntas e respostas. Os agentes que necessitam de alguma informação escrevem seu pedido no quadro à espera que outros agentes respondam à medida que acessem o mesmo [43].

3.6.3.2.2 Arquitetura de troca de mensagens

Nesta arquitetura, os agentes se comunicam diretamente, uns com os outros, através de mensagens assíncronas. Não existe o papel do quadro-negro como intermediário na interação, mas pode existir um agente facilitador de comunicação [42]. Dessa forma, é necessário que cada agente saiba os nomes e endereços de todos os agentes que formam o sistema para que as mensagens possam ser trocadas. Esse método é mais eficiente no sentido de obter as mensagens em tempo hábil, mas por outro lado, sua implementação é dificultada pelo crescimento exponencial das diferentes mensagens trocadas pelos agentes que compõem a sociedade.

3.6.3.2.3 Arquitetura federativa

Considerando uma arquitetura de troca de mensagens, onde o número de agentes é muito grande, a emissão de uma mensagem em broadcasting levará um tempo que pode inviabilizar todo processo de comunicação do sistema. Diante desse problema, surgiu a arquitetura federativa onde os agentes da sociedade são divididos em grupos ou federações segundo um critério de agrupamento escolhido. Junto a cada grupo de agentes encontram-se os agentes facilitadores responsáveis por receber a mensagem que chega em cada grupo e encaminhá-la para o agente destinatário presente naquele grupo [42]. A vantagem dessa arquitetura é que o agente facilitador tem a propriedade de identificar se a mensagem que chega é destinada a algum agente de seu grupo e se for o caso, fazer a devida entrega.

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

Tendo em vista que cooperação é a capacidade de troca de informações ou partilha de tarefas entre os agentes, a comunicação dá suporte a implementação desse mecanismo como pode ser observado na seção 3.7. Todavia, podem ser encontrados problemas nesse mecanismo e no mecanismo de coordenação que influenciam no relacionamento dos agentes que vivem em sociedade. Esses problemas são descritos na seção 4.4.

3.7 Comunicação entre Agentes

A comunicação é uma importante característica que os agentes devem possuir para que possam cooperar e interagir [47]. Por isso, é necessária uma linguagem de comunicação que seja comum a todos os agentes que vivem em sociedade. A KQML (*Knowledge Query and Manipulation Language*) surge como uma linguagem de comunicação entre agentes (LCA), capaz de prover suporte aos mecanismos de coordenação e cooperação [23]. Mas, para os agentes comunicarem entre si, também é necessário que eles tenham autorização para participarem de um diálogo. Cada agente tem um papel específico em um diálogo [23].

De acordo com a característica do agente e o seu papel a desempenhar no sistema podemos destacar diferentes níveis ou capacidades de comunicação. Para participarem de um

diálogo, os agentes precisam trocar mensagens. Essas mensagens podem ser de dois tipos: consultas e afirmações [23]. Uma consulta é realizada quando um agente envia uma pergunta para um outro agente. Uma afirmação é o resultado da consulta, ou seja, o agente que recebeu uma consulta envia uma resposta ao agente que solicitou a consulta.

Huhns e Stephens [23], classificam os agentes, de acordo com a capacidade de comunicação que eles apresentam:

- **Agente básico** - aceita consultas do exterior e tem que aceitar afirmações;
- **Agente passivo** - capaz de participar de um diálogo, além de aceitar perguntas do exterior e responder na forma de afirmações;
- **Agente ativo** - capaz de participar de um diálogo em que assume um papel ativo. Além de aceitar afirmações, ele pode fazer perguntas e afirmações;
- **Agente interlocutor** - capaz de participar de um diálogo em que assume um papel de interlocutor com os outros agentes. Além de aceitar afirmações, ele pode fazer e receber perguntas e fazer afirmações.

Jaques [25] destaca a necessidade de definir uma arquitetura que permita as interações entre agentes em uma sociedade, dando assim, o suporte necessário para a cooperação. Nas interações ocorrem trocas de conhecimento, objetivos, planos ou escolhas de propostas ou contra-propostas. De acordo com a arquitetura definida a comunicação entre os agentes pode ocorrer de forma direta ou indireta:

- **Comunicação direta** - os agentes se conhecem e, por isso, trocam informações diretamente entre si;
- **Comunicação indireta** - os agentes não se conhecem e, desta maneira, a comunicação ocorre através de uma estrutura de dados compartilhada.

Segundo Demazeau e Muller [7], existem basicamente três tipos de interações entre agentes. Neste caso, os tipos de informação que são compartilhadas: conhecimento, possibilidades ou escolha, determinam o tipo de interação. As possibilidades podem ser consideradas como contra-propostas que são trocadas pelos diversos agentes na sociedade, gerando assim a possibilidade de escolha. As interações podem ser do tipo:

- **Interação forte** - quando os agentes compartilham conhecimentos, possibilidades e escolhas. Neste tipo de comunicação são utilizados protocolos

de comunicação sofisticados para os agentes, tais como informar, requisitar ou convencer;

- **Interação média** - quando os agentes compartilham apenas conhecimento e possibilidades. Este tipo de interação ocorre quando os agentes desejam apenas saber o que os outros agentes pretendem fazer. Desta maneira, os agentes irão conhecer os planos para execução de tarefas dos outros agentes, até que um plano comum seja encontrado;
- **Interação fraca** - quando os agentes trocam conhecimento apenas. Esta interação ocorre através de troca de mensagens que serão ocasionadas pela percepção de alterações no ambiente.

No capítulo 4 serão utilizadas tabelas iguais ao modelo da **Tabela 4**, onde são especificados os diversos tipos de interações, as diferentes arquiteturas de comunicação e a capacidade de comunicação dos diferentes agentes que fazem parte dos padrões a serem analisados.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor

Tabela 4 – Tabela genérica de suporte ao mecanismo de cooperação.

3.7.1 Protocolo e linguagem de comunicação KQML

Utilizou-se a KQML para representar as interações entre os agentes porque ela é uma linguagem e um protocolo de comunicação, de alto nível. A troca de mensagens independe da sintaxe do conteúdo e da ontologia aplicável [14]. A KQML é independente do mecanismo de transporte (TCP/IP, SMTP), independe da linguagem conteúdo (KIF, SQL, Prolog) e independe da ontologia assumida pelo conteúdo [14].

A KQML se preocupa com os formatos das mensagens e seus protocolos, sem focar no formato da informação propriamente dita. Isso permite que sejam realizadas operações sobre as bases de conhecimento de cada agente envolvido.

A linguagem inclui um grande conjunto de mensagens pré-definidas (*performatives*), que definem as operações que podem ser executadas pelos agentes.

Segundo Finin [14], a KQML adota o uso de ontologias. Uma ontologia é um conjunto de especificações explícitas de significado, conceitos e relacionamentos aplicáveis a algum domínio específico. Isso assegura que dois agentes utilizem a mesma linguagem para se comunicar, eliminando a ambigüidade.

A KQML é um protocolo de mensagens pré-formatadas com significados e funções bem específicos. A linguagem não sugere como deve estar estruturada a arquitetura dos agentes, sendo caracterizada como uma linguagem de propósito geral, independente do ambiente ou estrutura dos agentes. O conjunto de mensagens KQML pode ser ampliado desde que novas *performatives* criadas obedeçam à especificação original da linguagem.

As mensagens KQML codificam informação em três diferentes níveis: conteúdo, mensagem e comunicação. A sintaxe da KQML é descrita na seção 3.7.3.

3.7.2 As Camadas da KQML

Cada mensagem KQML contém informações que estão relacionadas com cada uma das suas camadas e que são importantes na troca de mensagens entre os agentes [14].

3.7.2.1 Camada de conteúdo

A camada de conteúdo abrange o conteúdo da mensagem, representado por uma expressão em alguma linguagem que encapsule a informação ou conhecimento a ser transmitido. A linguagem na qual o conteúdo da mensagem KQML está escrito deve ser conhecida pelo agente que a recebe já que este agente deve interpretá-la para tomar uma ação adequada.

Na mensagem KQML a seguir, o conteúdo está representado em linguagem natural: (`ask-if`:`sender` XX :`receiver` YY :`in-reply-to` MSG01 :`reply-with` MSG02 :language Natural :`ontology` "teste de conteúdo" :`content` "Exemplo de uma mensagem KQML"). Neste exemplo, temos indicado o conteúdo "Exemplo de uma mensagem KQML" de uma mensagem escrita na linguagem natural.

3.7.2.2 Camada de mensagem

A camada de mensagem está associada aos tipos de mensagens que são enviadas durante a comunicação, ou seja, qual *performative* deve ser usada no momento de transmitir o conteúdo.

Como o conteúdo de uma mensagem é invisível para a linguagem KQML, surge a necessidade de apresentar alguns atributos ou parâmetros que podem fornecer informações adicionais sobre o conteúdo. Dentre esses destacamos o parâmetro `:language`, que indica a linguagem na qual o conteúdo foi escrito, e o parâmetro `:ontology`, que descreve a qual domínio o conteúdo pertence.

3.7.2.3 Camada de comunicação

A camada de comunicação apresenta alguns parâmetros que servem de informação para que a mecânica de comunicação seja devidamente efetuada. Tais informações estão representadas pelos parâmetros: emissor da mensagem (`:sender`), o receptor (`:receiver`) e os identificadores únicos das mensagens enviadas e recebidas (`:reply-with` e `:in-reply-to`).

3.7.3 Sintaxe da KQML

A sintaxe de uma mensagem KQML visa a formalização das mensagens, já que deve ser legível para humanos, simples para os que realizam a análise sintática e facilmente transportável para outras aplicações.

A definição sintática é apresentada na **Figura 12** e possui poucas regras e símbolos, facilitando a especificação das mensagens.

```
<performative>::=(<word> { <whitespace> :<word> <whitespace>
<expression>}*)
<expression> ::=<word> | <quotation> | <string> | (<word>
{<whitespace> <expresion>}*)
<word> ::=<character><character>*
<character> ::=<alphabetic>|<numeric>|<special>
<special> ::=< | > | = | + | - | * | / | & | ^ | ~ | _ | @ | $
| % | : | . | ! | ?
<quotation> ::= '<expression> | '<comma-expression>
<comma-expression> ::= <word> | <quotation> | <string> |
,<comma-expression>
(<word> {<whitespace> <comma-expression>}*)
<stringchar> ::= \<ascii> | <ascii>-\-<double-quote>
```

Figura 12-Definição sintática da KQML.

3.7.4 Parâmetros das mensagens KQML

As mensagens KQML também possuem um conjunto de parâmetros reservados identificados por palavras chave que começam com ":" (dois pontos) com um valor associado a eles.

Os parâmetros em KQML possibilitam um comportamento diferente das mensagens dependendo dos valores associados a eles. Abaixo, apresentamos os principais parâmetros descrevendo a funcionalidade dos mesmos:

- `:sender` <*word*> e `:receiver` <*word*>: especificam o identificador do agente emissor de uma performative e do seu receptor;
- `:from` <*word*> e `:to` <word>: usados na mensagem forward, identificam a origem e o destino da performative;
- `:reply-with` <*word*> e `:in-reply-to` <*word*>: atribuem um identificador único para a mensagem, que o agente emissor espera como resposta do receptor;
- `:content` <*expression*>: é o que se passa como conteúdo da mensagem. O valor de <*expression*> deve ser válido com relação à linguagem de representação do conteúdo;
- `:language` <*word*>: identifica qual a linguagem de representação do conteúdo da mensagem, por exemplo SQL (Structure Query Language);
- `:ontology` <*word*>: define a ontologia da mensagem, ou seja, a especificação do domínio de conhecimento na qual o conteúdo da mensagem deve ser interpretado.

3.7.5 Semântica das mensagens KQML

A semântica deve ser representada para que as mensagens sejam adequadamente usadas. Por isso é proposta uma descrição textual das principais mensagens, para a interação entre agentes.

A grande desvantagem de uma descrição em linguagem natural da semântica é a ambigüidade na sua compreensão e aplicação. Em vista disso, uma formalização da especificação semântica se faz necessária para a perfeita compreensão dessas mensagens.

3.7.6 Teoria das ações de discurso e a KQML

A estrutura das mensagens KQML é baseada na teoria das ações de discurso (*Speech Act Theory*) que tenta criar um modelo de como se processa a comunicação humana.

Dessa forma, as *performatives* KQML se assemelham muito à maneira com que as pessoas dialogam.

A teoria das ações de discurso se preocupa com o papel da linguagem como ação, que pode ser:

- "*locution*": representa o que é dito para o ouvinte;
- "*illocution*": demonstra o modo como o emissor transmite para o ouvinte;
- *"perlocution*": é uma ação que ocorre como resultado de uma *illocution*.

Por exemplo, a frase "Apresente sua monografia" é uma "*locution*"; a maneira com que o emissor manda apresentar sua monografia é a "*illocution*" e o resultado de mandar apresentar sua monografia, podendo este apresentar ou não, é a "*perlocution*".

Na KQML as *performatives* incluem uma expressão do tipo da ação, sugerindo uma "*illocution*". A "*locution*" representa a própria mensagem que está sendo emitida e, finalmente, a "*perlocution*", representa qual a ação é executada pelo agente receptor da mensagem.

3.7.6.1 Principais mensagens KQML

As *performatives* podem ser classificadas em três categorias: *performatives* de discurso, *performatives* de intervenção e mecânica de conversação e, por último, *performatives* de rede.

3.7.6.1.1 Performatives de discurso

As *performatives* de discurso estão baseadas na teoria das ações de discurso e podem ser usadas para troca de informações e conhecimento entre os agentes. Por exemplo:

```
a) (ask-if
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

O uso da *performative* "`ask-if`" é adequado quando o emissor (`:sender`) quer saber se a expressão do conteúdo (`:content`) é verdade para o receptor (`:receiver`), ou seja, se coincide com uma das sentenças que fazem parte de sua base de conhecimento.

```
b) (ask-one
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

No exemplo (b), a *performative* "`ask-one`" deseja conhecer onde o conteúdo da mensagem está de acordo com o conteúdo existente na base de conhecimento do receptor. Normalmente, a resposta do receptor é proposta através de um valor para o caso encontrado.

```
c) (tell
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

A *performative* "`tell`" informa para o receptor que o conteúdo do parâmetro `:content` faz parte da base de conhecimento do emissor, ou seja, que esse conteúdo é verdade para o emissor.

```
d) (untell
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

Para a *performative* "`untell`", o conteúdo de `:content` é tido como falso, isto é, não está presente na base de conhecimento do emissor.

3.7.6.1.2 *Performatives* de intervenção mecânica e de conversação

A principal função das *performatives* de intervenção mecânica e de conversação é intervir no fluxo normal de conversação quando uma situação anormal acontece, através da interrupção do fluxo de mensagens ou por meio da introdução de um protocolo mais complexo na interação para que seja superada uma anomalia na comunicação.

Por exemplo, a *performative* "`error`" é usada pelo emissor para informar se algum erro ocorre na mensagem identificada pelo parâmetro `:in-reply-to`. Os três tipos

mais comuns de erros são erro sintático, erro nos parâmetros ou erro com a política ou protocolo de conversação. Outras *performatives* que também atuam na intervenção mecânica e de conversação são "`sorry`" e "`deny`".

```
(error
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>)
```

3.7.6.1.3 Performatives de rede

As *performatives* de rede servem como utilitários para facilitar a comunicação entre os agentes, permitindo encontrar os agentes que podem responder às mensagens enviadas por um agente qualquer. Neste caso, o agente facilitador está presente, gerenciando a sociedade de agentes e conhecendo todos os agentes na sociedade, seus endereços e características.

Por exemplo, a *performative* "`register`" é usada pelo emissor para anunciar ao agente facilitador a sua presença na sociedade (nome e endereço físico). Em `:content` estão as informações sobre o agente emissor que o facilitador deve conhecer.

```
a) (register
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

A *performative* "`unregister`" permite que o agente facilitador passe a não mais reconhecer determinado agente.

```
b) (unregister
:sender <word>
:receiver <word>
:in-reply-to <word>
:reply-with <word>
:language <word>
:ontology <word>
:content <expression>)
```

3.7.6.1.4 Resumo das *performatives*

A **Tabela 5** mostra a definição das *performatives* em KQML. Essas *performatives* serão utilizadas na representação dos atos de comunicação entre os agentes nos padrões propostos na seção 4.4.

Nome	Descrição
achieve	S quer que R complete uma ação.
advertise	S quer que R saiba que S pode e processará mensagens do tipo da que está em seu conteúdo.
ask-if	S quer saber se o conteúdo de sua mensagem é verdadeiro para R.
ask-all	S quer todas as instâncias de R, para as quais o conteúdo da mensagem de S é verdadeiro.
ask-one	S quer uma resposta de R para uma questão.
break	S quer suspender uma comunicação
broadcast	S quer que R envie uma performative para todos os seus contatos
broker-all	S quer que R encontre uma resposta para todas suas performatives.
broker-one	S pede a R para achar uma resposta para a performative do conteúdo da sua mensagem.
deny	S indica a R que a performative já não aplica a R.
delete	S quer que R apague uma expressão de sua base de conhecimento.
discard	S não quer as respostas seguintes de R.
eos	Fim de uma sucessão de respostas de um pedido anterior.
error	S indica a R que recebeu uma mensagem não compreendida.
forward	S quer que R repasse a mensagem para o agente ":to".
insert	S pede para R acrescentar o conteúdo da mensagem na base de conhecimento de R.
monitor	S quer que R atualize suas respostas para um fluxo contínuo.
next	S quer a próxima resposta a um pedido anterior.
pipe	S quer que R redirecione todo as performatives seguintes a um agente.
ready	S está pronto para responder a uma performative de R.
recommend-all	S quer que todos os agentes respondam a uma determinada performative.
recommend-one	S pede a R para sugerir um agente que possa processar seu conteúdo.
recruit-all	S quer todos os agentes capazes respondam a uma performative.
recruit-one	S quer um agente capaz de responder a uma performative.
register	S anuncia para R (facilitador) sua presença e nome simbólico associado com seu endereço físico.
reply	S responde a um pedido esperado.
rest	S quer todas as respostas seguintes.
sorry	S diz a R que compreende sua mensagem, mas não pode prover uma resposta.
standby	S quer que R esteja pronto para responder a uma performative.
subscribe	S quer que R atualize suas respostas a uma performative.
tell	S informa para R que seu conteúdo é verdadeiro, ou seja, que a sentença está em sua base de conhecimento.
transporte-address	S anuncia um novo endereço físico na rede.
unregister	cancela um register feito anteriormente.
untell	Indica que uma expressão não forma parte da base de conhecimento de S

Tabela 5- Lista de *performatives* em KQML[11].

É importante destacar que a utilização dessas *performatives* auxilia no processo de coordenação e cooperação entre os agentes, aumentando assim a eficiência dos sistemas.

CAPÍTULO 4

PADRÕES ARQUITETURAIS BASEADOS EM AGENTES

CAPÍTULO 4 PADRÕES ARQUITETURAIS BASEADOS EM AGENTES

4.1 Introdução

O estudo de padrões arquiteturais iniciou-se com Buschmann [4]. Ele extraiu e identificou alguns padrões arquiteturais baseados na tecnologia orientada a objetos (objeto passivo). Entretanto, com o advento da tecnologia orientada a agentes (objeto ativo) [3], surgiram novos conceitos (autonomia, reatividade, aprendizagem) que não estavam inseridos no contexto da orientação a objeto. Então, naturalmente há uma necessidade de complementar a descrição dos padrões existentes e de identificar e extrair novos padrões para o desenvolvimento de aplicações baseadas em agentes.

Neste capítulo, propõe-se uma coletânea de padrões arquiteturais o desenvolvimento de sistemas multiagente enfatizando os mecanismos de cooperação e coordenação que caracterizam a sociedade. A capacidade de evolução dos padrões propostos através da extensão e/ou composição também é objeto de análise.

Foi seguida uma metodologia que se iniciou com o estudo de algumas arquiteturas de sistemas multiagente até o agrupamento dos padrões em uma coletânea.

4.2 Metodologia

Os padrões arquiteturais propostos foram desenvolvidos da seguinte forma:

- Foram estudados e reescritos alguns padrões abordados no trabalho de Deugo, Wess, Kendall [8], que trata apenas os padrões sob o contexto da coordenação. Na descrição dos padrões propostos foram utilizados a AUML para especificar a estrutura dos padrões e a KQML para representar a interação entre os agentes participantes. A **Tabela 5** traz uma lista de *performatives* utilizadas para essa representação.
- Analisaram-se também, algumas arquiteturas de software multiagente (camada, federativa e quadro-negro) a partir das quais foram extraídos e identificados alguns padrões.
- A técnica "*pattern mining*" foi utilizada como técnica para extrair e identificar novos padrões. Essa técnica foi descrita na seção 2.4.1;
- Depois de extraídos os padrões, estes foram agrupados em coletâneas, para torna-los acessíveis a outros desenvolvedores.

4.3 Trabalhos Relacionados

A construção de componentes reutilizáveis para o desenvolvimento de sistemas multiagente tais como padrões de projeto, vem sendo objeto de várias pesquisas.

Weiss [51] descreve um grupo de padrões arquiteturais para comércio eletrônico baseado em agentes. Esses padrões descrevem atividades de entradas e saídas típicas no comércio eletrônico envolvendo interação com o usuário e delegação de tarefas. Um dos padrões propostos é o *Agent as Delegate* que, apresentamos como exemplo, a seguir:

Descrição

Nome: *Agent as delegate*

Problema:

Como você orienta um assistente de software? Quantas oportunidades você deveria dar ao assistente de software (por exemplo, autoridade para completar um negócio)? Como os assistentes de software interagem com o mundo?

Contexto:

O desenvolvedor está projetando seu sistema como uma sociedade de agentes onde o usuário delega tarefas aos assistentes de software.

Força:

- Sobrecarga de informações no sistema de software;
- Pesquisa com custo (excesso de comunicação).

Solução:

Ë utilizado um agente usuário para agir em favor do usuário do sistema. O agente usuário delega tarefas para o agente comerciante que possui os papéis de comprador e vendedor. O comprador tem a finalidade de identificar um produto que está de acordo com as exigências do usuário, localizar o vendedor para o produto. O agente usuário aprende sobre as necessidades do usuário e constroem um perfil de usuário. Este perfil de usuário permitir que os vendedores gerenciem suas ofertas ao gosto do usuário. A estrutura deste teste padrão é mostrada na **Figura 13**.

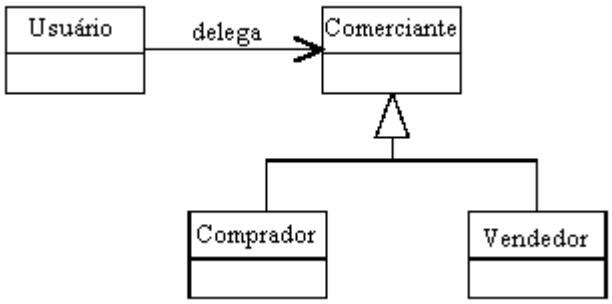

Figura 13-Estrutura do padrão *agent as delegate* [51].

Buschmann [4] descreve vários padrões arquiteturais orientados a objetos, entre eles:

- **Camada:** padrão arquitetural que propõe a decomposição de aplicações em grupos de subtarefas, onde cada grupo se apresenta em um nível particular de abstração. O padrão camada é organizado hierarquicamente, com cada camada servindo a camada acima e utilizando serviços da camada abaixo;
- **Canais e filtros:** este padrão arquitetural suporta a divisão de uma função ou processo em várias etapas de processamento seqüenciais. Cada etapa recebe, processa e disponibiliza uma cadeira de dados e o fluxo de saída de uma etapa correspondente ao fluxo de entrada da etapa seguinte;
- **Quadro-negro:** vários subsistemas organizam o conhecimento para a construção de uma possível solução aproximada ou parcial através do uso de uma memória compartilhada. Os subsistemas trabalham juntos para a construção da solução. Tais subsistemas não interagem diretamente entre si e nem há uma seqüência determinada para as suas ativações. A principal variação deste padrão é o repositório, uma generalização onde não há a especificação de um componente responsável pelo controle interno da aplicação. Os sistemas que utilizam banco de dados tradicionais podem ser considerados como exemplos de repositórios.

Deugo [8], apresenta padrões no nível arquitetural, onde é abordada a coordenação nos padrões quadro negro, reunião, mercado, mestre-escravo e agente negociador, além de destacar a importância do mecanismo de coordenação em sistemas multiagente. Nesse trabalho, também é abordada a necessidade de padronização do desenvolvimento de software agente com a utilização dos padrões propostos.

4.4 Uma Coletânea de Padrões Arquiteturais Baseados em Agentes

A **Tabela 6** apresenta uma coletânea dos padrões arquiteturais propostos descritos segundo os mecanismos de coordenação e cooperação entre os agentes da sociedade. Essa coletânea descreve o nome, o problema e a solução do padrão ao problema.

Problema	Solução	Padrão
Como podem os agentes ter acesso a informações e conhecimento de maneira facilitada sem se comunicarem diretamente?	A solução para o problema está na criação de um ambiente de coordenação passivo chamado de Quadro-negro. Esse ambiente é acessado por dois tipos de agentes: o especialista e o supervisor.	Quadro negro
Como os agentes podem comunicar-se com outros agentes, através de mensagens assíncronas com o objetivo de facilitar e agilizar a comunicação entre os agentes?	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.	Difusor
Quando uma mensagem em *broadcasting* pode inviabilizar todo o processo de comunicação de um sistema, como pode reverter-se esse quadro?	Dividi-se a sociedade em grupos 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.	Federativo
Como 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.	Mestre-escravo
Como se descobrem e resolvem interações que são conflitantes entre os agentes?	A solução envolve um Iniciador que começa um circulo 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.	Agente negociador
Como os agentes aceitam coordenar suas tarefas e mediar suas atividades?	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.	Reunião
Como podem relacionar-se os usuários de um serviço (*compradores*) com os provedores (*vendedores*)?	Define-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.	Mercado
Como estruturar e organizar as dependências entre subsistemas que estão em diferentes níveis de abstração.	A solução envolve a definição de um critério de abstração para a divisão das responsabilidades entre as camadas. Em seguida, determina-se o número de níveis de abstração, dá-se um nome as camadas e associa-se o serviço de cada uma delas.	Camada

Tabela 6 -Coletânea de padrões arquiteturais.

4.4.1 Padrão quadro-negro

Descrição

Nome: Quadro negro

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

Contexto:

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

Força: Os agentes precisam colaborar para executar tarefas complexas que se estendem além das suas capacidades individuais. Esses agentes são projetados para trabalharem de forma autônoma, ou seja, eles independem de outros agentes. Os agentes teriam acesso a um ambiente passivo como um repositório de informações.

O processo de gravação e recuperação das mensagens torna-se demorado à medida que as consultas ao repositório forem ocorrendo. O uso desse padrão não é aconselhado em sistemas de tempo real porque um agente especialista qualquer pode ficar esperando sua vez de acessar o repositório.

Solução: A solução para o problema está na criação de um ambiente de coordenação passivo chamado de quadro-negro [8]. Esse ambiente é acessado por dois tipos de agentes: o especialista e o supervisor. O primeiro pode acrescentar, atualizar ou apagar dados do quadro-negro. Vários agentes no papel de especialista tentam ao mesmo tempo inserir informações no quadro-negro usando a *performative* `insert(X)`. O especialista também monitora continuamente o quadro-negro na procura de mudanças. O supervisor é responsável por decidir qual especialista deve responder a uma mudança, quando múltiplos especialistas quiserem responder ao mesmo tempo. O supervisor é uma espécie de controlador das ações dos especialistas. O quadro-negro usa a *performative* `pipes(X)` para redirecionar as mensagens ao supervisor que responde com a *performative* `reply(X)` e indica quem deve acessar o quadro-negro.

A **Figura 14** ilustra a interação entre pacote participante e o pacote de controle. O pacote participante é composto pelos agentes especialistas e agente supervisor. O pacote de controle pelo quadro-negro e também pelo agente supervisor.

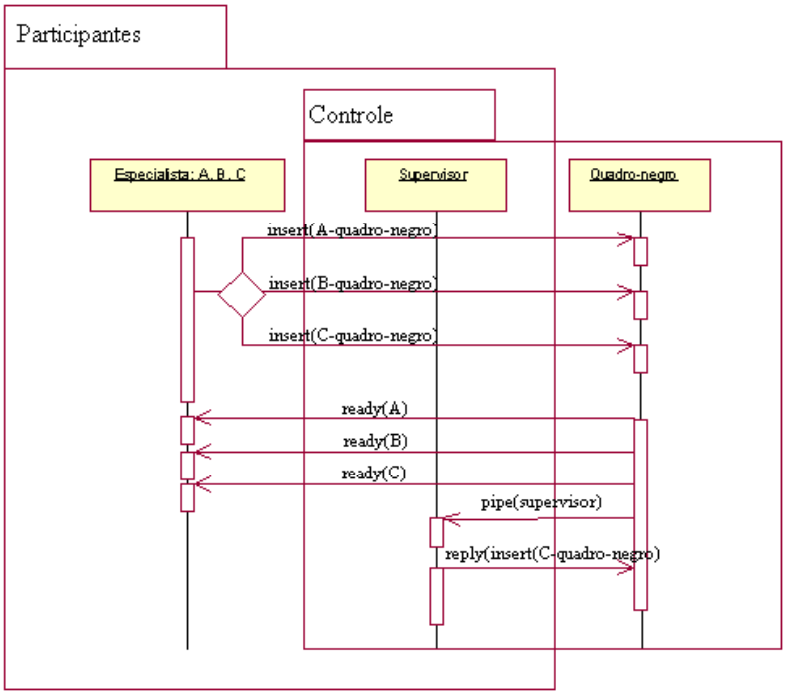

Figura 14 -Diagrama de pacotes com as interações entre os agentes no padrão Quadro-negro.

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

Mecanismo de cooperação

A cooperação neste padrão ocorre através da partilha de resultados. Esses resultados são informações depositadas dentro do quadro-negro e recuperadas por outros agentes quando forem necessárias. A arquitetura proposta neste padrão dá suporte à comunicação indireta entre os agentes especialistas e entre os agentes especialista e o supervisor. Isto ocorre devido à existência do quadro-negro. Portanto, não há troca de mensagens de maneira direta entre os agentes participantes deste padrão. Os agentes especialistas trocam apenas conhecimentos e utilizam *performatives* de discurso [14], isso caracteriza um tipo de interação fraca. O agente supervisor tem poder de decisão e utiliza *performatives* de rede, conseqüentemente realiza interações fortes. A capacidade de comunicação do supervisor é do tipo de um agente interlocutor, devido a sua capacidade de controlar o acesso dos agentes especialista ao quadro-negro A **Tabela 7**, mostra o resumo desse mecanismo no padrão Quadro-negro.

O agente especialista, nesse padrão poderá evoluir e adquirir maior capacidade de comunicação, podendo gerar interações médias ou até interações fortes. Essa implementação poderá ser efetivada através da composição com outros padrões, tais como o agente negociador.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Especialista			X		X		X		
Supervisor	X				X				X

Tabela 7-Mecanismos de cooperação baseados na troca de mensagens do Padrão Quadro-Negro [7],[23],[25].

Mecanismo de coordenação

No padrão Quadro-negro a coordenação é baseada na utilização de um agente chamado de supervisor que controla o acesso ao quadro negro evitando que dois ou mais agentes especialistas acessem o quadro negro ao mesmo tempo. Na **Figura 15** é mostrada a estrutura do padrão quadro-negro, segundo esse mecanismo. O processo de coordenação utilizado é a supervisão direta.

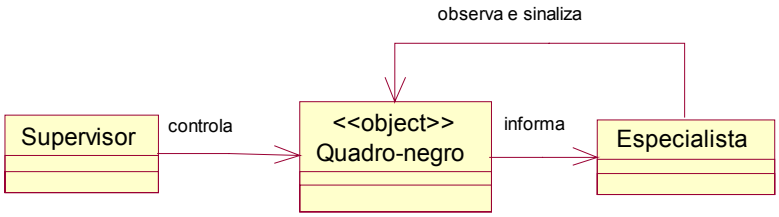

Figura 15 - Estrutura do padrão Quadro-negro [8].

Pode ocorre que vários agentes especialistas sinalizem para o quadro-negro ao mesmo tempo simulando uma concorrência, imediatamente o supervisor toma conhecimento e informa os especialistas. Todos os especialistas agentes estão sobre o controle do agente supervisor que controla o acesso ao quadro-negro (**Figura 14**).

4.4.2 Padrão Difusor

Descrição

Nome: Difusor

Problema: Como os agentes podem comunicar-se com outros agentes, através de mensagens assíncronas com o objetivo de facilitar e agilizar a comunicação entre eles?

Contexto: Um agente necessita comunicar-se diretamente com outros agentes, onde o requisito mínimo para ocorrer essa comunicação é o conhecimento prévio do nome e endereço de todos os agentes da sociedade.

Força: O uso desse padrão é recomendado quando se quer trocar mensagens em um intervalo de tempo pequeno (aplicações em tempo real). Por outro lado, esta solução pode ser bastante cara quando se tem um crescimento exponencial das mensagens trocadas entre os diferentes agentes que formam a sociedade.

Solução: Para que as trocas de mensagens ocorram de maneira adequada entre os agentes é necessário estabelecer um protocolo de comunicação. O protocolo, por exemplo, a KQML é quem dita as regras e impõe o formalismo necessário para que as mensagens sejam encaminhadas e compreendidas pelos agentes. Observando o contexto na **Figura 16**, o agente A envia uma mensagem do tipo `broadcast(ask-if(X))` com uma pergunta inserida, para todos os agentes dos quais conhece o nome e o endereço. Então, o agente capacitado para responder a essa mensagem, no exemplo o agente C, encaminha sua resposta usando `tell(X)` dizendo que o conteúdo da mensagem é verdadeiro. O diagrama de interação e pacote mostra as interações dos diversos agentes no padrão difusor (**Figura 16**).

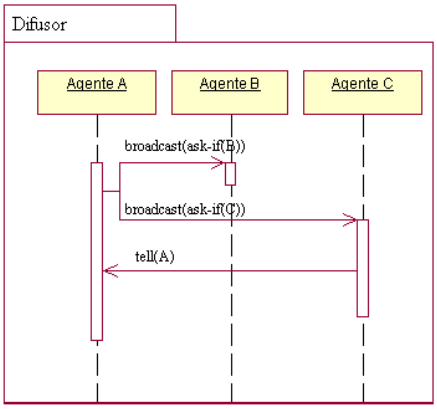

Figura 16- Diagrama de pacotes e interação no padrão Difusor.

Mecanismo de cooperação

A cooperação no padrão difusor ocorre através da partilha de tarefas e resultados. As tarefas podem ser serviços prestados e os resultados são informações trocadas pelos agentes da sociedade. A arquitetura proposta neste padrão dá suporte à comunicação direta entre os agentes participantes (agente A, B e C) da sociedade. Neste padrão, os agentes participantes têm basicamente a capacidade de comunicação de um agente do tipo ativo, ou

seja, eles são capazes de aceitar afirmações, poder fazer perguntas e afirmações. Basicamente as interações entre os agentes ocorrem de maneira fraca, utilizam *performatives* de discurso. Porém existe no mínimo um agente que produz interação forte como no caso do agente A que usa *performative* de rede. A Tabela 8, mostra um resumo desse mecanismo no padrão difusor.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Agente A	X			X				X	
Agente B			X	X				X	
Agente C			X	X				X	

Tabela 8-Mecanismos de cooperação baseados na troca de mensagens Padrão difusor [7],[23],[25].

Mecanismo de coordenação

A coordenação no padrão difusor utiliza o processo de ajuste mútuo para compartilhar seus recursos com outros agentes. Pois, no padrão difusor não existe um agente exclusivo para fazer o papel de coordenador. Neste caso, todos os agentes são responsáveis em obter um ajuste mútuo para alcançar seus objetivos.

4.4.3 Padrão Federativo

Descrição

Nome: Federativo

Problema: Considerando um sistema de troca de mensagens onde o número de agentes é muito grande, a emissão de uma mensagem em *broadcasting* pode inviabilizar todo o processo de comunicação de um sistema.

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

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

Solução: Observando o contexto podemos dividir a sociedade em grupos menores seguindo um critério de agrupamento. Em cada um desses grupos existe um agente facilitador

que é responsável por receber e encaminhar a mensagem para o agente destinatário (**Figura 17**).

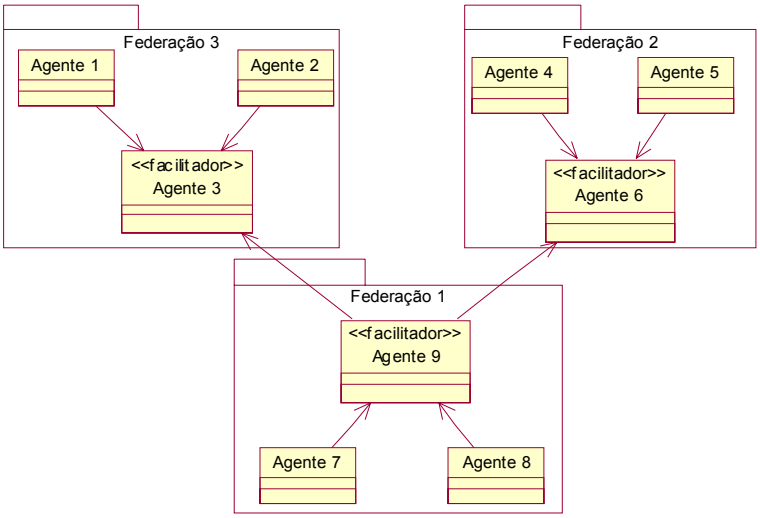

Figura 17-Estrutura do Padrão federativo [42]

A **Figura 18** mostra o diagrama de pacotes onde o agente 8 faz uma pergunta ao agente facilitador 9, usando `ask-if(X)`. Como ele não sabe responder e nem conhece ninguém da sua federação capaz de responder, essa pergunta é encaminhada para os demais agentes facilitadores, através de uma mensagem do tipo `broadcast(ask-if(X))`. Os agentes facilitadores observam se alguém de sua federação pode responder. No exemplo da **Figura 18**, o agente facilitador 6 recebe a pergunta e a encaminha para o agente 5, usando `recommend-one(ask(X))`. O agente 5 responde retornando a solução para o agente facilitador 6, usando `tell(X)` que tem a função de encaminhar a resposta até o solicitante inicial agente 8, usando `forward(tell(X))`.

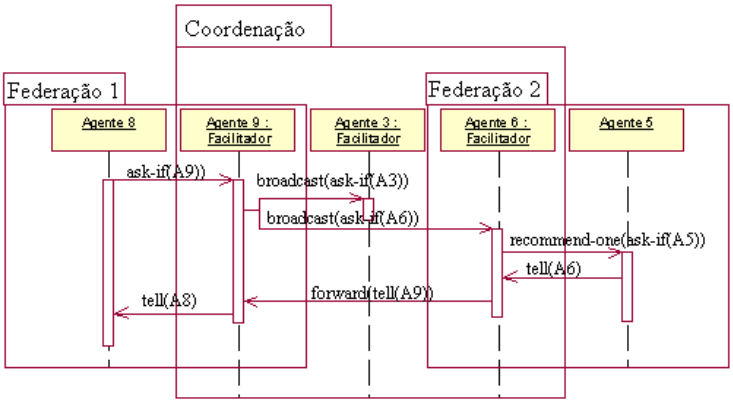

Figura 18-Diagrama de pacotes e a interação dos agentes na federação

Mecanismo de cooperação

A cooperação no padrão federativo ocorre através da partilha de tarefas e resultados. As tarefas são serviços que podem ser prestados por outros agentes na federação. Os resultados são informações enviadas a um ou mais agentes solicitantes dentro da federação. A solução arquitetural proposta neste padrão dá suporte a dois tipos de comunicação: direta e indireta. A comunicação direta ocorre quando um agente facilitador se comunica com outros agentes facilitadores. A comunicação indireta usa a estrutura do agente facilitador para coordenar a comunicação dos agentes participantes de uma mesma federação, por exemplo, o agente 8 com o agente 7. A interação entre os agentes facilitadores que atuam como interlocutores é mais sofisticada e considerada interação forte porque esses agentes são mais complexos e utilizam *performatives* de rede, portanto, com maior poder de interação. Mas, a interação entre os participantes da federação, por exemplo, o agente1 e o agente2, pode ser considerada uma interação fraca porque esses agentes são agentes passivos ou ativos comuns. Essa interação pode evoluir na medida que os agentes participantes, também evoluam. A utilização de agentes facilitadores neste padrão é obtida através da evolução dos agentes participantes do padrão difusor. Essa evolução é descrita na seção 4.5. A Tabela 9, traz um resumo do mecanismo de cooperação no padrão federativo.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Facilitador(3,6,9)	X			X					X
Agente(1,2,4,5,7,8)			X		X		X	X	

Tabela 9-Mecanismos de cooperação baseados na troca de mensagens no Padrão Federativo [7],[23],[25].

Mecanismo de coordenação

O mecanismo de coordenação está baseado na utilização de agentes chamados de facilitadores que controlam os serviços de cada federação enviando a mensagem para o agente apropriado e assim diminuindo o fluxo de mensagens na sociedade (**Figura 18**). O processo de coordenação utilizado é do tipo *supervisão direta*.

4.4.4 Padrão Mestre-Escravo

Descrição

Nome: Mestre-escravo

Problema: Como delegar sub-tarefas e coordenar sua execução?

Contexto: O projetista decide dividir uma tarefa em sub-tarefas para aumentar a confiança, desempenho, ou precisão na sua execução.

Força: Um agente pode dividir uma tarefa e delega sub-tarefas a outros agentes para melhorar a confiança, desempenho, ou precisão da tarefa que é executada. Por exemplo, em quanto um agente mestre tiver outros agentes que trabalham em sub-tarefas por ele delegadas, o agente mestre pode continuar com seu próprio trabalho, enquanto as sub-tarefas são realizadas em paralelo pelos agentes escravos [8].

Solução: A solução envolve um mestre que divide a tarefa em sub-tarefas, em seguida delega essas sub-tarefas a escravos e depois os resultados parciais são enviados dos escravos para o mestre, que tem a responsabilidade de computar o resultado final. O mestre indica um escravo para cada sub-tarefa, neste caso ele utiliza `recruit-all(X)` para delegar tarefas para os agentes escravos. Enquanto o escravo computa o resultado parcial da tarefa que foi nomeada pelo mestre, o mestre pode continuar seu trabalho normalmente [8].

Quando um escravo tiver terminado seu trabalho envia resposta, usando `reply(X)`, para o mestre que compila o resultado final e retorna para o cliente. A interação dos agentes do padrão é mostrada na **Figura 19**.

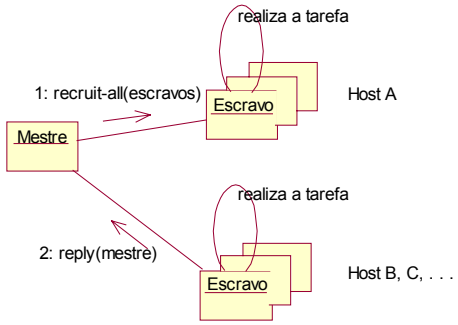

Figura 19-Diagrama de colaboração entre os componentes do padrão Mestre-escravo.

Uma aplicação típica do padrão Mestre-escravo é a computação paralela. Vários escravos podem ser nomeados para um grupo de trabalho que é controlado por um mestre. Cada escravo oferta os seus serviços. Os clientes enviam seus pedidos ao mestre que controla estes pedidos dividindo cada um deles em sub-tarefas que despacha aos escravos ociosos do grupo de trabalho.

Mecanismo de cooperação

O mecanismo de cooperação no padrão mestre-escravo ocorre através da partilha de tarefas. Essas tarefas são delegadas pelo agente mestre aos agentes escravos, Por isso, o agente mestre possui o papel de interlocutor no padrão. A arquitetura proposta neste padrão dá suporte a comunicação direta dos agentes participantes da sociedade (mestre e escravo). Esses agentes não necessitam de intermediários para trocar informações. O agente mestre é um agente mais sofisticado que os agentes escravos que agem de maneira passiva, trabalhando para os agentes mestres. As interações do agente mestre são fortes, pois ele tem poder de decisão e utilizam *performatives* de rede, como `recruit-all(X)`. Porém, os escravos realizam interações fracas devido a não possuir em poder de decisão A **Tabela 10**, apresenta um resumo do mecanismo de cooperação no padrão mestre-escravo.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Mestre	X			X					X
Escravo				X			X		

Tabela 10-Mecanismos de cooperação baseados na troca de mensagens Padrão Mestre-Escravo[7],[23],[25].

Mecanismo de coordenação

O mecanismo de coordenação é implementado pelo agente mestre que tem o objetivo de controlar os pedidos e em seguida enviá-los para os escravos ociosos. O processo de coordenação utilizado é a supervisão direta, pois existe o agente mestre que faz o papel de facilitador para distribuir as tarefas. A KQML também dá suporte a coordenação nesse padrão, quando o agente mestre utiliza a *performative* `recruit-all(X)`, o mestre só deseja receber resposta de escravos que forem capacitados para responder, evitando assim trocas de mensagens desnecessárias.

4.4.5 Padrão Camada

Descrição

Nome: Padrão Camada

Problema: Como estruturar e organizar a dependência entre subsistemas que estão em diferentes níveis de abstração?

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

Força: As camadas são organizadas de maneira hierárquica. Cada camada tem o objetivo de prover serviços bem definidos para a camada seguinte, a qual funciona como um cliente para a camada anterior. Cada camada pode possuir um ou mais agentes. Cada camada possui seu protocolo de comunicação com suas camadas vizinhas. A alteração de uma camada não pode atingir mais de duas outras camadas, visto que uma camada se comunica, no mínimo, com uma camada e, no máximo, com duas. As responsabilidades similares devem ser agrupadas para auxiliar o entendimento e a manutenção.

O fato de se cruzar às fronteiras entre as camadas pode ter um impacto negativo na performance.

Solução: A solução envolve a definição de um critério de abstração para a divisão das responsabilidades entre as camadas. Em seguida, determina-se o número de níveis de abstração, dá-se um nome as camadas e associa-se o serviço de cada uma delas. Observa-se na **Figura 20**, que a camada j, fornece serviços a camada j+1 e delega tarefas à camada j-1.

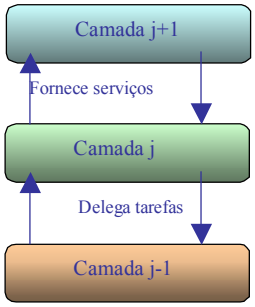

Figura 20-Estrutura do padrão camada

Mecanismo de cooperação

O mecanismo de cooperação no padrão camada é evidenciado através da relação de dependência entre as camadas, ou seja, uma camada pode delegar tarefas usando `recruit-one(X)` e a outra fornece os serviços por ela solicitados utilizando `reply(X)` ou servindo de intermediário quando utiliza `forward(X)` (Figura 21).

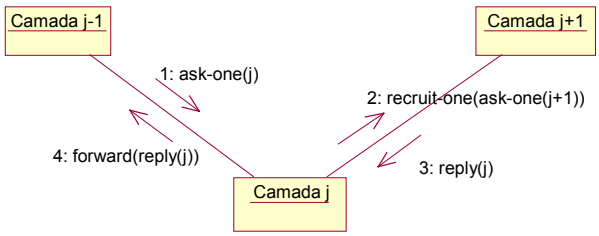

Figura 21- Diagrama de colaboração entre as camadas

A arquitetura desse padrão dá suporte a comunicação direta e indireta entre os agentes pertencentes as diferentes camadas. A comunicação direta ocorre quando um agente se comunica com outros agentes pertencentes a uma camada imediatamente superior ou inferior a sua. A comunicação indireta ocorre quando existe alguma camada que intermédia à comunicação. A interação de tipo forte é a utilizada pelos agentes participantes desse padrão, pois eles necessitam do poder de decisão que caracteriza este tipo de interação. A capacidade de comunicação utilizada é do tipo agente interlocutor, pois uma camada poderá delegar tarefas para outras camadas. A Tabela 11, traz um resumo do mecanismo de cooperação no padrão camada.

Agente/ papel	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Participante	X			X	X				X

Tabela 11-Mecanismos de cooperação baseados na troca de mensagens no Padrão camada.

Mecanismo de coordenação

O mecanismo de coordenação pode ser implementado através da composição com outros padrões que suportam a coordenação, como é o caso do padrão federativo. No caso de se utilizar o padrão federativo para implementar as camadas, o processo de coordenação utilizado seria *supervisão direta*; caso contrario os agentes participantes utilizariam o *ajuste mútuo* para compartilhar recursos a fim de atingirem seus objetivos. A utilização das *performatives* de rede também auxilia na coordenação dos agentes neste padrão.

4.4.6 Padrão Agente Negociador

Descrição

Nome: Agente Negociador

Problema: Como se podem descobrir e resolver interações que são conflitantes entre os agentes?

Contexto: Agentes interagindo com outros agentes participantes da sociedade.

Força: Os agentes organizam suas ações para atingirem suas metas.

Solução: A solução envolve um iniciador que começa um processo de negociação declarando sua intenção a todos os agentes da sociedade. Essa intenção pode vir na forma de uma pergunta `ask-all(X)` e deve ser apresentado antes do iniciador prosseguir com suas ações planejadas. Os agentes fazem uma analise da intenção do iniciador no papel de crítico na negociação. Eles testam se houver um conflito entre a intenção declarada e seu próprio curso da ação planejada. Se não houver nenhum conflito, um crítico aceita a ação proposta com `reply(X)`, se não faz um contra-proposta, usando `advertise(X)` para indicar que só pode responder uma parte ou rejeita a ação completamente usando `sorry(X)`. As contra-propostas contêm as ações alternativas que são aceitáveis ao crítico [8]. As rejeições indicam que há um impasse na estrutura da negociação. A **Figura 22** mostra as interações neste padrão.

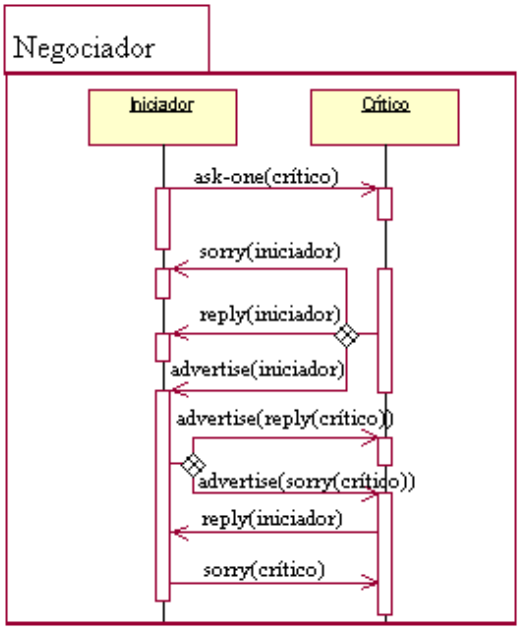

Figura 22- Diagrama de pacotes do padrão negociador e as interações entre o iniciador e crítico

Um agente pode ser um iniciador ou um crítico em negociações diferentes ao mesmo tempo. Conseqüentemente, é criado um outro papel, o participante, que contém os

papéis do iniciador e do crítico. Os participantes agem freqüentemente em nome de outros agentes para quem estão negociando [8]. Por exemplo, um iniciador pode negociar em nome de um agente comprador sobre os termos de uma transação. Uma vez que os termos foram determinados pelo iniciador, o comprador paga à quantia negociada ao vendedor e recebe o bem. As interações principais deste padrão são mostradas na **Figura 22**.

Mecanismo de cooperação

O mecanismo de cooperação no padrão *Agente negociador* ocorre através da partilha de tarefas e resultados. As tarefas podem ser solicitadas pelo iniciador e analisadas pelo crítico que pode aceitar, rejeitar ou fazer uma contra-proposta. Os resultados estão na capacidade do crítico de rejeitar quando a proposta pode gerar conflito. A arquitetura desse padrão dá suporte a comunicação direta entre os agentes que utilizam os papéis de iniciador e crítico. Esses agentes não necessitam de intermediários para a troca de informações. Neste padrão, tanto o iniciador quanto o crítico realizam interações fortes, pois eles possuem poder de decisão e utilizam *performatives* de rede. A capacidade de comunicação do iniciador e do crítico é de um agente do tipo ativo (agente que aceita afirmações, faz perguntas e afirmações). A **Tabela 12** mostra um resumo desse mecanismo no padrão agente negociador.

Agente/ papel	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Iniciador	X			X				X	
Crítico	X			X				X	

Tabela 12-Mecanismos de cooperação baseados na troca de mensagens Padrão Agente Negociador [7],[23],[25].

Mecanismo de coordenação

O mecanismo de coordenação está baseado na utilização de um agente participante que pode assumir dois papéis diferentes, o de iniciador que faz as propostas e o de crítico que tem a função de aceitar, rejeitar ou fazer uma contra-proposta. O processo de coordenação utilizado é chamado de *padronização*, pois são estabelecidos procedimentos a serem seguidos, tais como: a possibilidade de receber uma proposta do agente iniciador. Quando essa proposta não é totalmente aceitável, o agente crítico enviar então uma contra-proposta.

4.4.7 Padrão Reunião

Descrição

Nome: Agente Reunião

Problema: Como os agentes aceitam coordenar tarefas e mediar atividades?

Contexto: Os agentes têm a necessidade de interagir a fim coordenar suas atividades sem a necessidade de explicitamente nomear aqueles envolvidos na tarefa. Estes agentes podem ser estáticos ou moveis. A troca de informações entre agentes e o ambiente é possível e também entre os próprios agentes. Entretanto, a localização dos agentes deve ser conhecida pelo agente que coordena as atividades.

Força: A troca de mensagens entre agentes situados dentro do mesmo ambiente é rápida, segura e simples. Não envolvendo a rede e suas conexões, os agentes podem comunicar-se usando as facilidades internas da infraestrutura do ambiente.

Solução: Cria-se um lugar em um ambiente onde os agentes possam se reunir. Em seguida, deixa-se um agente chamar para reunião os diversos agentes da sociedade usando `ask-one(X)`. Permite-se que as interações possam ocorrer no contexto da reunião, através de agentes que se comunicam de forma síncrona, um com os outros a fim coordenar suas atividades. O gerente da reunião permite que os agentes que tem mobilidade movam-se para o lugar da reunião, mas os agentes estáticos são notificados através de mensagens do tipo `recruit-all(X)` no seu local de origem.

O gerente da reunião reutiliza o padrão *"observer"* [31], pois é ele quem notifica os agentes interessados na reunião quando a mesma é proposta.

O gerente da reunião aceita as mensagens dos agentes remotos ou locais `register(X)` que querem registrar-se para serem notificados de alguma reunião específica. Em suas mensagens, no momento do registro, os agentes devem identificar quem eles são e onde podem ser encontrados. O gerente da reunião também aceita mensagens dos agentes remotos ou locais interessados em chamar uma reunião e informam os agentes registrados de quando a reunião ocorrerá. O gerente da reunião tem a responsabilidade de controlar a reunião, registrar agentes enquanto chegam e apagar o seu registro quando eles partem [8]. A interação do padrão é mostrada na **Figura 23**.

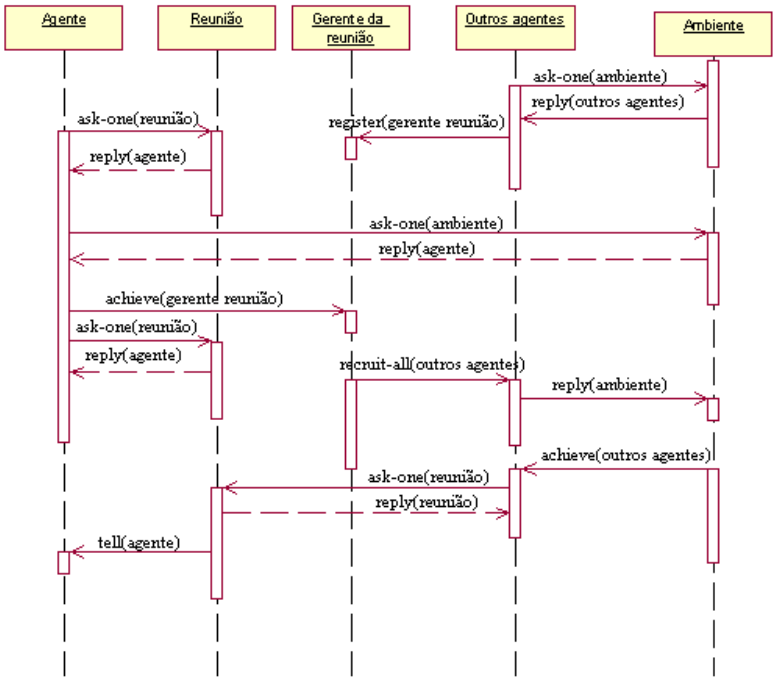

Figura 23-Diagrama de interação do padrão reunião.

Mecanismo de cooperação

O mecanismo de cooperação do padrão reunião ocorre através da partilha de tarefas e resultados. As tarefas e os resultados podem ser trocados pelos agentes no momento da reunião. A arquitetura proposta neste padrão dá suporte à comunicação direta entre os agentes, pois é criado um ambiente onde os agentes podem se reunir e trocar informações. As interações neste padrão são no mínimo fracas, pois trocam apenas conhecimentos, com os agentes participantes da reunião e forte com o agente gerente da reunião porque ele tem poder de decisão. A capacidade de comunicação do agente gerente é de *interlocutor*, pois é ele que controla a reunião. Entretanto, os agentes participantes basicamente têm a característica de agentes passivos. A **Tabela 13** mostra um resumo do mecanismo de cooperação no padrão reunião.

Agente	**Interação**			**Comunicação**		**Capacidade de comunicação**			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Gerente	X			X					X
Agente participante			X	X			X		

Tabela 13-Mecanismos de cooperação baseados na troca de mensagens Padrão Reunião [7],[23],[25].

Mecanismo de coordenação

O mecanismo de coordenação é implementado pelo agente gerente da reunião responsável por fazer as chamadas para reunião, registrar, desregistrar, escrever na reunião e notificar os agentes. O processo de coordenação utilizado é a supervisão direta

4.4.8 Padrão Mercado

Descrição

Nome: Mercado

Problema: Como podem relacionar-se os usuários de serviço (compradores) com os provedores (vendedores)?

Contexto: Necessita-se de um sistema multiagente que possa constantemente evoluir. Em vez de montar relações entre agentes, deixa-se que os agentes localizem os sócios para realizar transações.

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

Solução: A solução envolve um corretor que aceita pedidos de compradores e ofertas de bens (recursos ou serviços) de vendedores. Esses pedidos são implementados usando `ask-one(X)`. O corretor controla a lógica da coordenação, anunciando pedidos usando `forward(X)` a vendedores, colecionando suas ofertas, e apresentando vendedores selecionados ao comprador usando `reply(X)`. Enfatiza-se que os agentes podem fazer os papéis de compradores e vendedores ao mesmo tempo. Introduz-se o papel de comerciante que contém os papéis de comprador e de vendedor. Os pacotes deste padrão são mostrados na **Figura 24**.

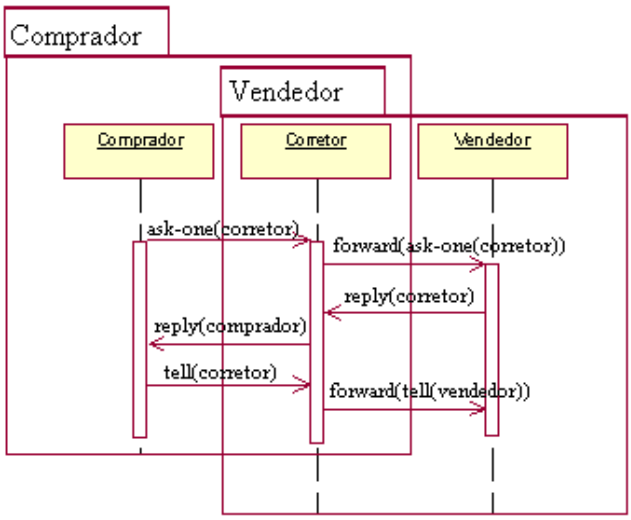

Figura 24-Diagrama de pacotes do padrão mecanismo de mercado e suas interações.

O corretor não tem o poder de escolher um vendedor para um comprador, ele faz somente a intermediação entre os dois, mas a escolha final fica dependendo das múltiplas ofertas vindas do vendedor ao comprador que escolhe a melhor oferta. O corretor, agindo em nome do comprador, emite pedidos de ofertas a todos os vendedores. Os vendedores decidem se querem fornecer um pedido de um bem e submetem as ofertas ao corretor que as envia para o comprador [8].

Mecanismo de cooperação

O mecanismo de cooperação no padrão mecanismo de mercado ocorre através da partilha de tarefas e resultados. A partilha de tarefas é feita através da execução de serviços pelos vendedores. Todavia, a partilha de resultados é evidenciada quando os vendedores disponibilizam informações para os compradores. A arquitetura proposta dá suporte a comunicação indireta, pois o agente corretor é o intermediário no sistema. A capacidade de comunicação do corretor é do tipo de um agente interlocutor. Entretanto, o vendedor e o comprador são do tipo ativo, capaz de aceitar afirmações, fazer perguntas e afirmações. As interações do agente vendedor e comprador são fortes, pois eles têm poder de decisão, o corretor é um intermediário sem poder de decisão (interação fraca). Um resumo do mecanismo de cooperação do padrão mercado estar descrito na **Tabela 14**.

Agente/ papel	Interação			Comunicação		Capacidade de comunicação			
	Forte	Média	Fraca	Direta	Indireta	Básico	Passivo	Ativo	Interlocutor
Vendedor	X				X			X	
Comprador	X				X			X	
Corretor			X	X					X

Tabela 14-Mecanismos de cooperação baseados na troca de mensagens Padrão Mecanismo de mercado [7],[23],[25].

Mecanismo de coordenação

O mecanismo de coordenação é implementado pelo agente corretor. Apesar dele não ter o poder sobre os vendedores e compradores, o agente corretor é responsável pela intermediação do negócio. O processo de coordenação utilizado é a supervisão direta.

4.5 Evolução de Padrões

Um padrão de qualidade dá suporte à geração de novos padrões através da composição e/ou extensão de padrões já existentes. Essas características evidenciam o nível de flexibilidade de um padrão permitindo que ele, uma vez aplicado, gere um contexto resultante que pode coincidir com o contexto inicial de um outro padrão.

A flexibilidades foi um requisito básico considerado na definição dos padrões descritos na seção 4.4 o que permite a evolução de padrões simples, tais como o padrão difusor, agente negociador, mecanismo de mercado para padrões mais complexos, tais como padrão federativo, camada, quadro negro através da composição de padrões e auxiliados pela geração de agentes mais especializados.

Nas próximas seções é abordada a composição como fator indispensável para a evolução dos padrões arquiteturais e sua reutilização.

O método que foi utilizado para compor os padrões consiste em observar o contexto que se resulta da aplicação de um padrão, ou seja, contexto resultante do padrão em seguida aplica-lo a um outro padrão.

4.5.1 Quadro negro especializado

Esse padrão foi descrito na seção 4.4.1 e tem como participantes o agente supervisor e os agentes especialistas. Quando esse padrão é aplicado ele gera um contexto resultante que coincide com um contexto inicial de outro padrão.

Problema resultante: dois especialistas querem acessar o quadro negro ao mesmo tempo.

Solução: o supervisor deve ser capaz de resolver conflitos, então, se aplica o padrão agente negociador no agente supervisor do padrão quadro negro (Figura 25).

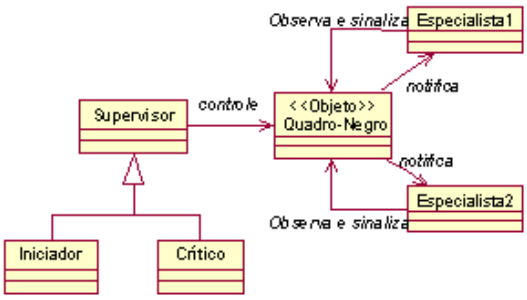

Figura 25- Padrão quadro negro especializado, composto do padrão agente negociador.

4.5.2 Padrão federativo

Esse padrão foi descrito na seção 4.4.3 e tem como participantes os agentes facilitadores e os agentes simples (agente 1, 2, 4, 7, 8), membros das federações. Esse padrão surgiu através da evolução do padrão difusor (descrito na seção 4.4.2), que aconteceu com a especialização de alguns agentes participantes do padrão. Esses agentes participantes que evoluíram passaram a ser chamados de agentes facilitadores e são responsáveis pela coordenação das federações.

Problema resultante: a sociedade é composta por uma quantidade muito grande de agentes, onde a troca de mensagens em *broadcasting* pode inviabilizar o sistema (**Figura 26**).

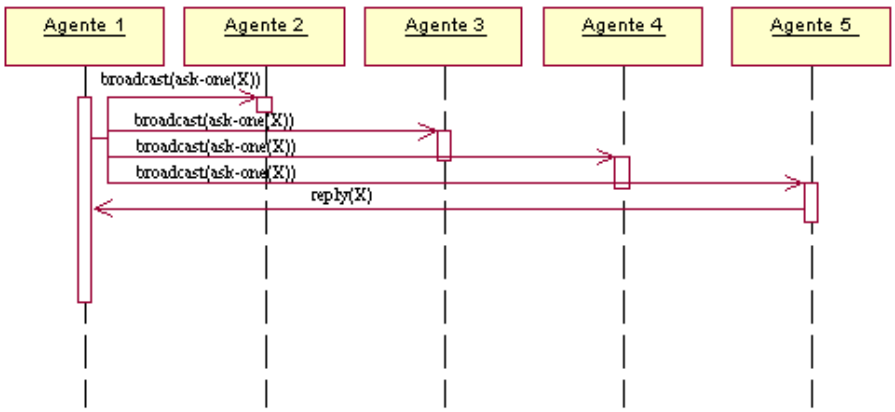

Figura 26- Padrão difusor

Solução: envolve a evolução de alguns agentes participantes do padrão difusor (Figura 27), em seguida dividi-se em federações segundo algum critério, formando assim o padrão federativo.

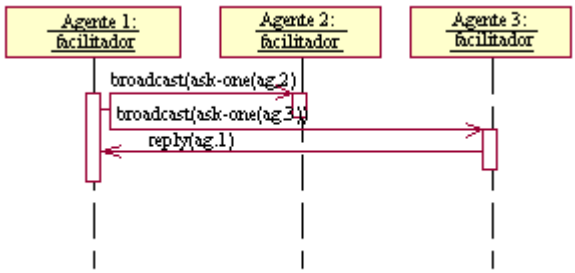

Figura 27- Agentes facilitadores no padrão difusor.

4.5.3 Padrão Camada

Esse padrão foi descrito na seção 4.4.5. Os agentes são dispostos em camadas. Quando esse padrão é aplicado ele gera um contexto resultante que coincide com contexto inicial de outro padrão.

Problema resultante: quem coordena as ações dos diversos agentes pertencentes as diferentes camadas, a fim de existir a harmonia entre os diferentes níveis de interação.

Solução: Cria-se um agente facilitador para ficar responsável por sua respectiva camada na qual foi criado. Esse agente facilitador conhece o nome, a localização e a especialidade de cada agente na respectiva camada. A Figura 28 mostra um agente facilitador responsável por todos agentes de uma determinada camada. A técnica de se usar um agente facilitador para coordenar outros agentes é empregada no padrão federativo. O agente facilitador no padrão federativo pode-se comunicar com um número ilimitado de outros agentes facilitadores, isto vai depender da quantidade de federações. Todavia, no padrão camada, o agente facilitador da camada, pode no máximo se comunicar com dois agentes facilitadores, isto garante um ganho na performance.

Figura 28- Agente facilitador responsável em coordenar uma camada

CAPÍTULO 5

ESTUDOS DE CASO

CAPÍTULO 5 ESTUDOS DE CASO

5.1 Introdução

Os estudos de caso apresentados neste capítulo têm o objetivo de validar os padrões extraídos, utilizando arquiteturas anteriormente projetadas e implementadas com sucesso. As arquiteturas analisadas foram: RETSINA [44], AMALTHAEA [33] e ABARFI [9]. Todas elas abordam o problema da recuperação e filtragem de informação. Escolheu-se esse domínio de aplicação porque este trabalho está inserido no contexto do projeto MaAE [16],[17] que tem a finalidade de desenvolver abstrações de software de alto nível no domínio da recuperação e filtragem de informação.

Descrevem-se as arquiteturas, destacando os tipos de agentes, a coordenação, cooperação e organização dos agentes bem como sua estrutura interna. Em seguida, são apresentados os padrões identificados.

5.2 Metodologia

Cada estudo de caso foi organizado da seguinte forma:

- Escolheu-se uma arquitetura de software do domínio da recuperação e filtragem de informação para validar os padrões extraídos da seção 4.4;
- Analisou-se o mecanismo de cooperação e coordenação utilizados na arquitetura;
- Analisou-se a arquitetura buscando: a identificação de padrões da coletânea proposta ou a detecção de soluções que poderiam ter sido melhor resolvidas através de alguns dos padrões propostos.

A seguir são apresentados as principais características e funcionamento de cada uma das arquiteturas abordadas.

5.3 ESTUDO DE CASO I: Padrões arquiteturais na RETSINA

5.3.1 Descrição da RETSINA

A arquitetura RETSINA (REusable Task Structure-based Intelligent Network Agents) faz parte de pesquisas iniciadas na Carnegie Mellon University, em Pittsburgh Pennsylvania, encabeçada pela Dra. Kátia Sycara [48],[49]. Ela é uma arquitetura multiagente (SMA) que trabalha com comunidades de agentes heterogêneos. A infraestrutura

computacional multiagente RETSINA consiste de um sistema de tipos de agentes reutilizáveis que podem ser adaptados para solucionar uma variedade de problemas específicos de domínio. Ela é composta por coleções distribuídas de agentes de software que cooperam assincronamente entre si, com o objetivo de recuperar e filtrar informação.

Existem basicamente três tipos de agentes em RETSINA: agentes de interface, agentes de tarefa e agentes de informação. Os agentes de interface interagem com o usuário, recebendo especificações e entregando resultados. Eles adquirem, modelam e utilizam as preferências do usuário para guiar a coordenação do sistema em defesa das tarefas do usuário. Os agentes de tarefa ajudam os usuários a executar tarefas, formulando planos para solucionar problemas e cumprindo estes planos através de consultas e troca de informações com outros agentes de software. Os agentes de informação provêem acesso inteligente a uma coleção de fontes de informação heterogêneas [9] .

5.3.2 Tipos de agentes em RETSINA

Em um framework RETSINA, cada usuário é associado a um conjunto de agentes que colaboram para apóia-lo em várias tarefas. Os agentes estão distribuídos e executam em diferentes máquinas; têm acesso aos modelos dos usuários. Baseados neste conhecimento, os agentes decidem como decompor e delegar tarefas, qual informação é necessária para cada ponto de decisão e quando iniciar procuras colaborativas com outros agentes para adquirir, integrar e validar a informação. Deste modo, as atividades de reunir informações dos agentes são automaticamente ativadas pelos modelos de tarefas e as necessidades de processamento dos agentes, em vez de serem completamente iniciadas pelo usuário. O usuário pode deixar algumas das decisões referentes à reunião de informações a critério dos agentes. Isto economiza tempo ao usuário, diminui sua carga cognitiva e aumenta sua produtividade. O grau de autonomia do agente é controlado pelo usuário. Á medida que o usuário ganha mais confiança nas capacidades dos agentes, maior poder de decisão é permitido a estes. Durante a procura, os agentes comunicam-se uns com os outros para requisitar ou prover informações, procurar fontes de informação, filtrar e integrar informações e negociar para resolver conflitos em informações e modelos de tarefas. A informação adquirida é retornada aos agentes para que estes a exiba de forma apropriada ao usuário [9],[48],[49].

A arquitetura distribuída do RETSINA pode ser vista na **Figura 29**, que mostra os três tipos de agente que a compõem e seus respectivos relacionamentos.

Os a**gentes de interface** interagem com o usuário, recebem requisições de informação e entregam resultados. Eles adquirem, modelam e utilizam preferências do usuário para guiar a coordenação do sistema em suporte às tarefas dos usuários [9],[48],[49].

As principais funções de um agente de interface incluem:

- Colecionar informações relevantes de seus usuários, para que uma tarefa possa ser iniciada;
- Apresentar informações relevantes, incluindo resultados e explicações;
- Perguntar ao usuário por informações adicionais durante a solução de um problema;
- Solicitar confirmação ao usuário, quando necessário.

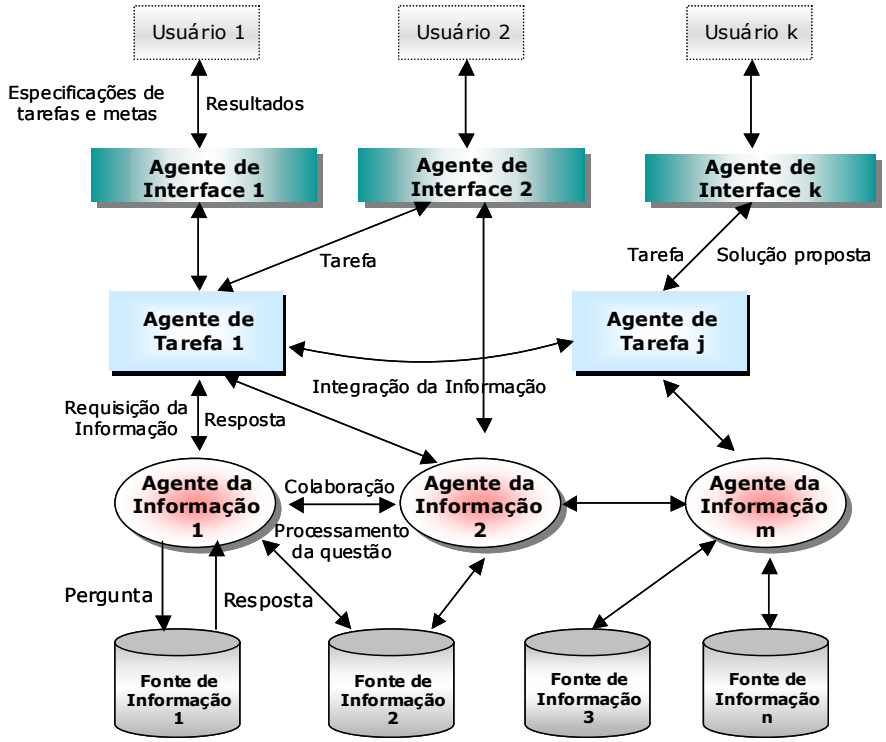

Figura 29-A organização distribuída dos agentes na arquitetura RETSINA [48],[49].

Do ponto de vista do usuário, o uso de um agente de interface que interaja diretamente com ele tem duas vantagens principais: primeira, toda a complexidade ligada ao processo de recuperar informação distribuída é transparente ao usuário e, segunda, ele não precisa interagir com uma grande quantidade de agentes (de tarefa e de informação).

Os **agentes de tarefa** suportam a tomada de decisões, formulando planos para a solução de problemas e fazendo com que estes planos sejam executados, através de consultas

e trocas de informação com outros agentes de software. Eles têm conhecimento do domínio da tarefa e de que outros agentes de tarefas ou agentes de informação que são relevantes para a execução das várias partes da tarefa. Em adição, possuem estratégias para resolver conflitos e integrar as informações recuperadas pelos agentes de informação. Um agente de tarefa executa a maioria das soluções de problemas de forma autônoma, exibindo um alto nível de sofisticação e complexidade em comparação aos demais agentes que compõem a arquitetura. Um agente de tarefa [9],[48],[49]:

- recebe de um agente de interface as requisições de informação solicitadas pelo usuário;
- interpreta as requisições e extrai metas para solução dos problemas;
- forma planos para satisfazer as metas;
- identifica as informações buscando submetas que estão presentes em seu plano;
- decompõe os planos e coordena-os, delegando tarefas aos agentes de tarefa ou agentes de informação apropriados, para composição da execução, monitoramento e resultado da execução dos planos.

Os a**gentes de informação** provêem acesso inteligente a uma coleção de fontes de informação heterogêneas, conforme pode ser visto na parte inferior da **Figura 29**. Eles possuem modelos de fontes de informação associadas, além de estratégias para seleção de fontes, acesso à informação, resolução de conflitos e integração da informação. As atividades de um agente de informação podem ser iniciadas de duas formas: de cima para baixo (*top down*), através das requisições feitas pelo usuário ou agente de tarefa, ou de baixo para cima (*bottom up*), através do monitoramento de fontes de informação para detectar a ocorrência de padrões de informações particulares. Uma vez que a condição para monitoramento tenha sido observada, o agente de informação envia mensagens de notificação para agentes que tenham mostrado interesse na ocorrência destas informações. Assim, os agentes de informação são autônomos, no sentido em que eles ativamente monitoram fontes de informação e pró-ativamente entregam a informação, em lugar de apenas esperar por ela [9],[48],[49].

Um agente de informação após o recebimento de mensagens de outros agentes pode agir de três diferentes formas:

- responder uma consulta sobre fontes de informações associadas,

- responder consultas periódicas que serão executadas repetidamente, e os resultados enviados para o solicitador a cada tempo;
- monitorar uma fonte de informação para uma mudança em uma parte da informação.

5.3.3 Coordenação e organização dos agentes

Na arquitetura RETSINA, os agentes são distribuídos através de diferentes máquinas e são diretamente ativados seguindo um fluxo *top-down* de acordo com o estado atual do sistema. Esta organização baseada em tarefas pode mudar com o passar do tempo, mas também pode permanecer relativamente estática por períodos extensos. Desta forma, a organização dos agentes não mudará como resultado do aparecimento ou desaparecimento das fontes de informação, mas as interações dos agentes poderão ser afetadas pelo aparecimento (ou desaparecimento) de agentes que são capazes de cumprir submetas de tarefas de novas maneiras. As informações que são importantes para a tomada de decisões (podendo causar uma eventual mudança na estrutura organizacional) são monitoradas pelos níveis mais baixos da organização e passadas para cima quando necessário. Neste tipo de organização, os agentes específicos de tarefas continuamente intercalam planejamento, escalonamento, coordenação e execução de ações para solução de problemas no nível do domínio [9],[48],[49].

A organização do sistema tem as seguintes características:

- Existe um número finito de agentes de tarefa com os quais cada agente se comunica;
- Os agentes de tarefa são eventualmente responsáveis por resolver conflitos e integrar informações obtidas de fontes de informação heterogêneas;
- Os agentes de tarefa são responsáveis pela ativação dos agentes de informação e pela coordenação das atividades de encontrar e filtrar informação.

Estes processos de coordenação distribuída podem ser vistos da seguinte forma. Quando um agente específico de tarefa recebe uma tarefa de um agente de interface ou de um outro agente específico de tarefa, ele decompõe a tarefa baseado no conhecimento do domínio que ele tem e, então, delega as sub-tarefas para outros agentes específicos de tarefa ou diretamente para agentes específicos de informação. Ao agente específico de tarefa será atribuída a responsabilidade de colecionar dados, resolver conflitos, coordenar entre os agentes relacionados e informar a quem iniciou a tarefa. Os agentes que são responsáveis por

sub-tarefas nomeadas ou decomporão estas sub-tarefas mais adiante, ou executarão a recuperação de dados (ou possivelmente outras atividades para solução de problemas locais de domínio específico) [9],[48],[49].

O *framework* RETSINA usa a linguagem KQML [14] para comunicação entre agentes. Com o objetivo de incorporar e utilizar agentes de software pré-existentes ou serviços de informação que já tenham sido desenvolvidos adota-se a seguinte estratégia: se um agente está sob controle do RETSINA, ele será construído usando KQML como linguagem de comunicação. Se não, é construído um agente *"gateway"* que conecta o sistema à organização de agentes e manuseia diferentes canais de comunicação, diferentes dados e formatos de requisições.

5.3.4 Estrutura interna do agente em RETSINA

Com o objetivo de operar em ambientes multiagentes ricos e dinâmicos, os agentes de software podem ser capazes de efetivamente utilizar e coordenar seus limitados recursos computacionais. Em RETSINA, cada agente de software é construído baseado em uma arquitetura de raciocínio sofisticada (Arquitetura de Controle de Tarefa [45] e TEAMS [6]), que consiste de diferentes módulos reutilizáveis, que são responsáveis pela comunicação, planejamento, escalonamento e monitoramento da execução de tarefas (Figura 30):

- O Módulo de Comunicação e Coordenação permite que um agente interaja com usuários, fontes de dados e outros agentes em redes distribuídas. Ele aceita e interpreta mensagens e requisições de outros agentes;
- O Módulo de Planejamento tem como entrada um conjunto de metas e produz um plano que deve ser seguido pelo agente para satisfazer estas metas;
- O Módulo de Escalonamento usa a estrutura de tarefa criada pelo módulo de planejamento para ordenar as tarefas, estabelecendo a ordem com que estas serão executadas;
- O Módulo de Execução monitora o processo e assegura que as ações serão cumpridas de acordo com os recursos disponíveis. Ele usa o plano escalonado para executar, manter-se informado e monitorar os processos que um agente está executando.

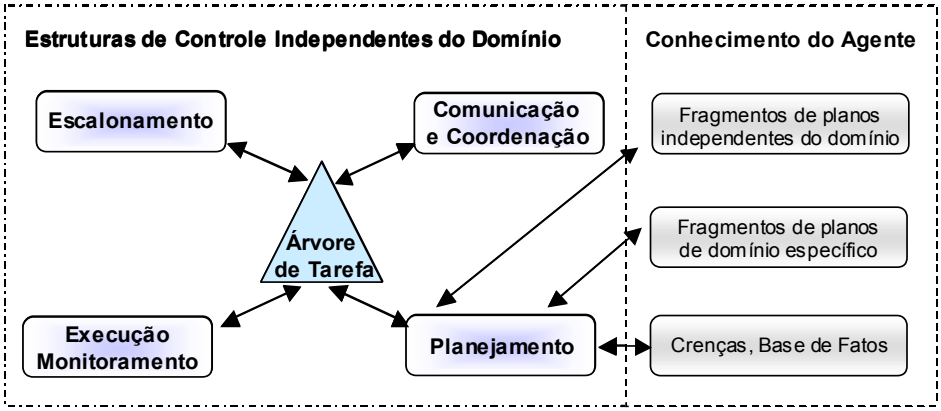

Figura 30-A arquitetura do agente em RETSINA: uma visão funcional [49],[48].

O **módulo de planejamento** recebe como entrada um conjunto de metas e produz um plano para satisfazer essas metas. O processo de planejamento do agente é baseado no formalismo de planejamento de uma rede hierárquica de tarefa (HTN – *Hierarchical Task Network*) [52]. Ele recebe como entrada o conjunto de metas atuais do agente, o conjunto atual das estruturas de tarefa, e uma biblioteca de esquemas de redução de tarefas. Um esquema de redução de tarefa apresenta uma maneira de executar (cumprir) uma tarefa especificando um conjunto de sub-tarefas/ações e descrevendo as relações de fluxo de informação entre eles. Isso é, a redução pode especificar que o resultado de uma sub-tarefa (por exemplo, decidir o nome de um agente) seja provido como uma entrada de outra sub-tarefa (por exemplo, enviar uma mensagem). As ações podem requerer que certas informações estejam disponíveis antes que possam ser executadas e podem também produzir informações sobre a execução. Por exemplo, o ato de enviar mensagens KQML requer o nome do destinatário e o conteúdo da mensagem, enquanto que o ato de decidir a quem enviar alguma mensagem produziria o nome de um agente. Uma ação é habilitada quando todas as entradas requeridas estão disponíveis [9],[49],[48].

O **Módulo de Comunicação e Coordenação** aceita e interpreta mensagens de outros agentes em KQML. Os agentes de interface também aceitam e interpretam mensagens de e-mail, o que é considerado um meio de comunicação conveniente com o usuário e/ou outros agentes de interface (por exemplo, agentes que provêem serviços de notificação de eventos). As mensagens podem conter requisições para serviços fazendo com que estas se tornem metas do agente destinatário [9], [48],[49].

O **Módulo de Escalonamento** programa cada um dos passos do plano. O processo de escalonamento do agente, em geral, recebe como entrada instâncias do conjunto de planos atuais do agente, em particular, o conjunto de todas as ações executáveis e decide

qual ação é a próxima a ser executada. Esta ação é então identificada como uma intenção fixa até que ela de fato seja cumprida pelo componente de execução [9],[48],[49].

Desta forma, este *framework* tem sido implementado através do desenvolvimento de agentes que colaboram para executar diversas tarefas complexas do mundo real, tais como tomadas de decisões organizacionais, e gerenciamento de pasta financeira [9],[48],[49].

5.3.5 Identificação e análise de padrões na RETSINA

Como mostrado na **Figura 29**, a arquitetura RETSINA organiza seus agentes em camadas segundo a sua funcionalidade (agente de interface, agente de tarefa e agente de informação). Cada camada é responsável por desenvolver um determinado serviço. A Figura 31, mostra as camadas identificadas em RETSINA e os fluxos de cooperação entre as camadas.

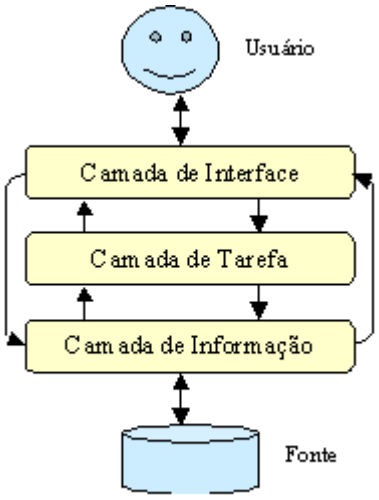

Figura 31-Camadas da arquitetura RETSINA.

A **Tabela 15**, mostra os padrões que foram identificados na arquitetura RETSINA, bem como aqueles que são sugeridos para melhorar o potencial de reutilização dessas arquiteturas. Esses padrões serão analisados em seguida segundo os mecanismo de coordenação e cooperação.

Arquitetura	Padrão
RETSINA	Camada
	Federativo
	Agente Negociador
	Mestre-Escravo

Tabela 15-Padrões identificados na arquitetura RETSINA.

5.3.5.1 Padrão Camada

A arquitetura RETSINA utiliza uma forma não comum de padrão camada que permite a quebra do encapsulamento das camadas (**Figura 31**). A quebra do encapsulamento das camadas favorece a dependência entre as camadas e dificulta a alteração ou evolução das camadas. Isto compromete a capacidade de reutilização da RETSINA.

Para solucionar esses problemas é necessária a utilização do padrão Camada proposto na seção 4.5.3. A utilização deste padrão permitiu minimizar a dependência entre as camadas e reduzir o impacto de mudanças nas camadas, favorecendo assim a reutilização.

Cooperação

O mecanismo de cooperação está implícito na arquitetura devido à relação de dependência entre as camadas: as camadas superiores necessitam dos serviços das inferiores e as inferiores prestam serviço para as superiores.

A comunicação entre os agentes participantes de cada camada ocorre de forma direta (comunicação que não necessita de intermediários) ou indireta com auxilio de interlocutores (agentes participantes de camadas intermediárias) que auxiliam na passagem das mensagens de uma camada para outra que não esteja diretamente ligada.

Coordenação

O mecanismo de coordenação nesta arquitetura é responsabilidade da camada de tarefa que é composta pelos agentes de tarefa que faz um papel de interlocutores na arquitetura.

Os agentes participantes da arquitetura e do padrão camada são:

- Agente de interface – permite interações do tipo médio devido à capacidade do agente em colecionar informações de seus usuários, de perguntar ao usuário por informações adicionais e solicitar confirmações. Portanto, esse agente não possui poder de decisão;
- Agente de tarefa – é o agente mais sofisticado da sociedade, capaz de realizar interações fortes, pois esse agente dá suporte à tomada de decisões:
  - Formulando planos;
  - Fazendo consultas e trocando de informações;

- Agente de informação - monitora as fontes de informação na procura de mudanças. As interações desses agentes com as fontes de informação são fracas.
- A camada de tarefa realiza uma espécie de supervisão direta das demais camadas.

5.3.5.2 Padrão federativo

Um outro padrão potencialmente reutilizável na arquitetura RETSINA é o federativo. Nessa arquitetura o agente de tarefa é uma espécie de agente facilitador que media o processo de comunicação entre os outros agentes de tarefa distribuídos. Portanto, na arquitetura RETSINA cada federação é estruturada na forma do padrão camada. Entretanto, existe também nesse padrão a quebra da estrutura organizacional do padrão que dificulta a reutilização, evolução e manutenção do sistema. Isto é mostrado na **Figura 32** que mostra a comunicação direta entre os agentes de informação.

Para solucionar esses problemas é necessária a utilização do padrão federativo proposto na seção 4.4.3. A utilização desse padrão minimiza o impacto do excesso da troca de mensagens, favorecendo assim a reutilização. A **Figura 33** reorganiza a RETSINA segundo o padrão federativo proposto neste trabalho.

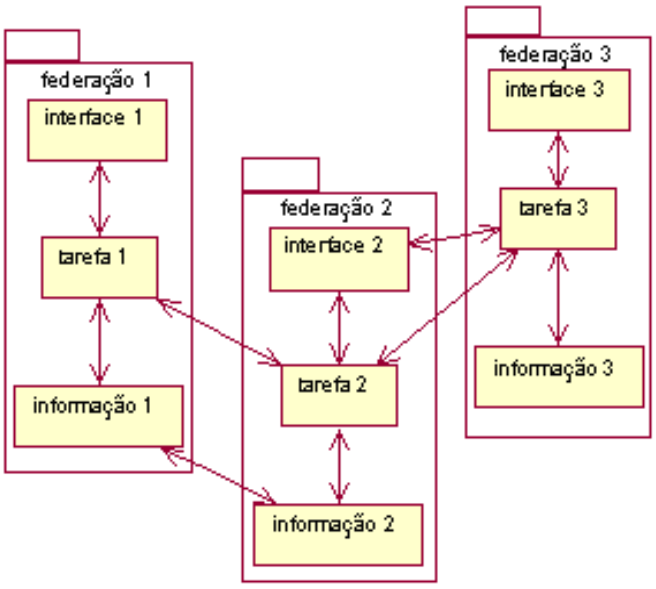

Figura 32-Federações na RETSINA.

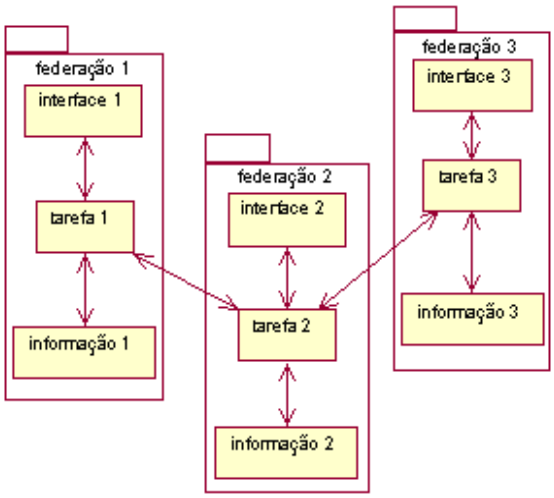

Figura 33-Reorganização da arquitetura RETSINA segundo o padrão Federativo

Cooperação

A cooperação está baseada na utilização da KQML, que facilita a comunicação entre as diferentes federações. Essa estrutura dá suporte a trocar informações ou serviços.

Coordenação

A coordenação da sociedade fica a cargo do agente de tarefa que tem o papel de interlocutor. Porém, não só existe a supervisão direta do agente de tarefa, existe também um ajuste mútuo que faz com que os diversos agentes da federação procurem alcançar seus objetivos independentemente do interlocutor.

5.3.5.3 Padrão agente negociador

A arquitetura RETSINA dá suporte ao tratamento de conflitos em todas as suas camadas. Por isso, em cada agente participante das camadas pode identificar-se o padrão agente negociador que é capaz de descobrir e resolver interações que são conflitantes entre os agentes da camada.

Cooperação

A cooperação está evidenciada na capacidade dos agentes em trocarem informações ou serviços, e na capacidade de comunicação que pode ser da forma direta ou indireta:

- Comunicação direta – esse tipo de comunicação ocorre quando os agentes de uma determinada camada interagem com agentes de camadas subseqüentes (acima ou abaixo) da sua.

- Comunicação indireta – quando a comunicação passa a ser intermediada por agentes participantes de alguma camada.

Coordenação

A coordenação dos papéis iniciador e critico está diretamente ligada com o processo de negociação entre os agentes. Cada agente pode alternar o seu papel dependendo do momento que ele está passando (**Figura 34**). O processo de coordenação utilizado neste padrão é a *padronização* porque os agentes seguem um padrão de negociação.

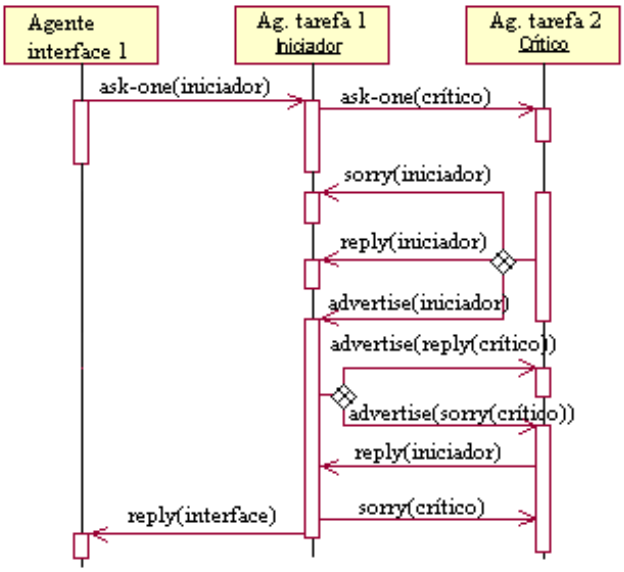

Figura 34- Interação do Agente de tarefa no papel de iniciador e crítico

5.3.5.4 Padrão mestre-escravo

O padrão mestre–escravo é utilizado na camada de tarefa. Os agentes de tarefa, participantes dessa camada, podem fazer o papel de mestre na arquitetura, decompondo planos e coordenando para delegar tarefas a outros agentes de tarefa ou agentes de informação. O agente de tarefa em um determinado momento pode fazer o papel de agente ativo (mestre) ou passivo (escravo) (**Figura 35**).

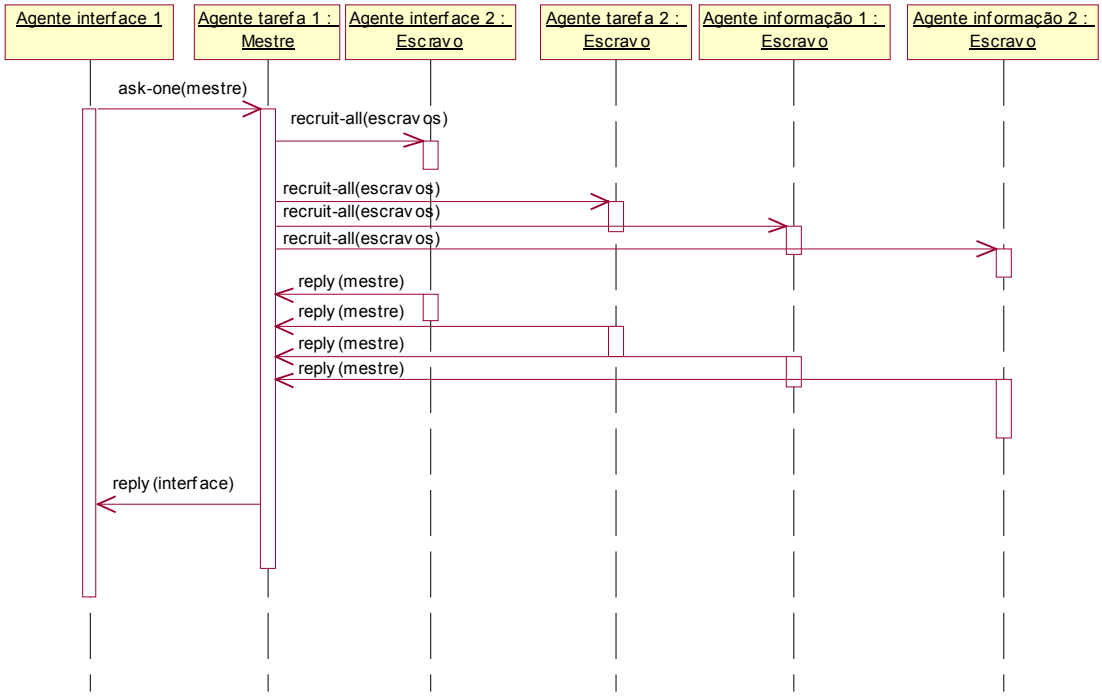

Figura 35-Interação do agente de tarefa 1 no papel de mestre interagindo com seus escravos.

Os demais agentes que compõem a arquitetura, também podem organizar-se segundo o padrão mestre-escravo.

Cooperação

O mecanismo de cooperação é evidenciado no momento em que os agentes de tarefa e de informação trabalham em função do agente de tarefa que delegou as tarefas.

Coordenação

A coordenação fica a cargo do agente de tarefa (mestre) que tem a capacidade de coordenar as ações e delegar às tarefas para os diversos agentes distribuídos pela sociedade. A supervisão direta é o processo de coordenação utilizado.

5.3.5.5 Padrão quadro-negro

O padrão quadro negro é identificado na ultima camada da arquitetura (camada de informação). Nessa camada os agentes de informação ficam monitorando constantemente as fontes de informação em busca de mudanças. A **Figura 36** mostra o processo de interação dos diversos agentes de informação com uma fonte de informação.

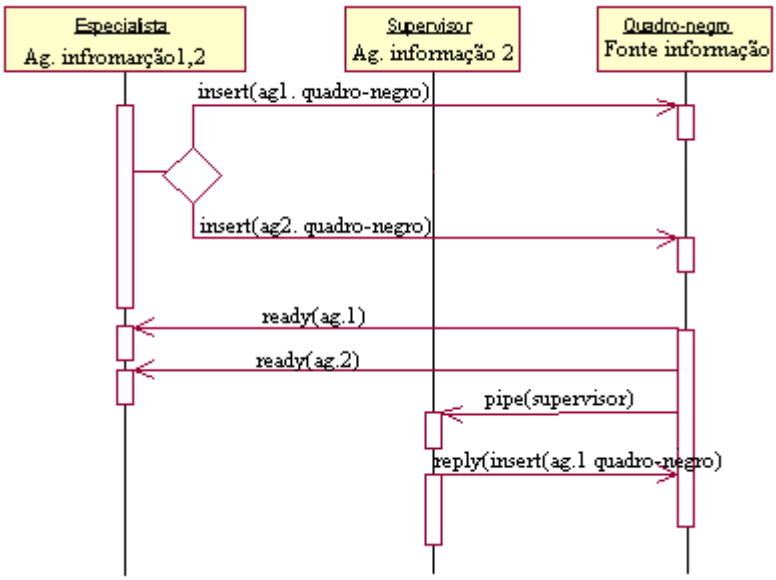

Figura 36- Interação do padrão quadro negro na arquitetura RETSINA

Cooperação

O mecanismo de cooperação é evidenciado no momento em que os agentes de informação ficam monitorando as fontes de informação atrás de mudanças para em seguida comunicar aos agentes de tarefas.

Coordenação

A coordenação fica a cargo do agente de informação (supervisor) que tem a capacidade de coordenar as ações e os acessos às fontes de informação.

5.4 ESTUDO DE CASO II: Padrões arquiteturais na AMALTHAEA

5.4.1 Descrição da AMALTHAEA

O Amalthaea é um projeto de autoria de Dr. Alexandros G. Moukas, apresentado à *Media Arts and Sciences School of Architecture and Planning* como requisito para obtenção do seu grau de *Master of Science in Media Technology* pelo *Massachusetts Institute of Technology* [33].

O Amalthaea consiste de uma sociedade de pequenos e simples agentes especializados que tentam resolver coletivamente o problema de recuperar informações relevantes ao usuário. Para isto foi idealizado um ecossistema artificial para desenvolver agentes de descobrimento e filtragem de informação que cooperam e competem em um ambiente de comercialização. O sistema adapta-se aos interesses do usuário, que podem

mudar com o passar do tempo, enquanto pró-ativamente exploram novos domínios (na *WWW*) que podem ser de interesse para o usuário [33].

O domínio focalizado é o descobrimento e filtragem de informação personalizada. Desta maneira, o Amalthaea pode ser visto como um sistema personalizado que pró-ativamente tenta descobrir informações das várias fontes distribuídas que interessam a seu usuário, apresentado-as em forma de um sumário. Ele aprende sobre os interesses do usuário examinando seus vínculos favoritos e o histórico das páginas visitadas e recebendo *feedback* do usuário após avaliação das páginas sugeridas. Então o sistema autonomamente coleciona documentos relacionados e URLs. Ele obtém dados em três domínios em paralelo [9],[33]:

- Documentos WWW e *descobrimento de dados*: o Almathaea não procura na WWW ele próprio, mas, ao invés disso, lança múltiplos agentes que utilizam as máquinas de indexação existentes e executam um *"meta-serch"* com o objetivo de descobrir informações que são de interesse do usuário. Então, o sistema analisa os documentos recuperados usando as técnicas do modelo do espaço vetorial com o objetivo de selecionar aquelas mais próximas às preferências do usuário;
- Fluxo contínuo de informação (no caso do Amalthaea, o *MIT Media Laboratory's Associated Press Newsfeed*). Neste caso, múltiplos agentes analisam os artigos e selecionam somente os convenientes;
- Monitorando freqüentemente mudanças nos recursos de informação. Algumas vezes o usuário quer monitorar certas URLs que são atualizadas de tempo em tempo. Por exemplo, ele quer saber sobre o novo resultado da corrida de automóveis da Fórmula 1 que é atualizado a cada dois domingos, ou um novo artigo em um jornal *on line*, ou, ainda, sobre um novo CD de um artista. Tais URLs serão monitoradas em intervalos regulares para a detecção de mudanças. Se tais mudanças existem e são significantes, então o usuário será notificado.

A operação do Amalthaea não requer a presença ou a atenção do usuário. A informação é apresentada ao usuário na forma de um sumário: novas URLs que podem ser interessantes, notícias personalizadas, notificações sobre um novo material em certos *sites*. O usuário folheia o sumário, pode seguir os *links*, prover uma avaliação sobre a qualidade de um

item, avaliar a relevância de um artigo ou palavra chave. A idéia é que o sistema se adapte ao usuário e siga seus interesses, evoluindo com o passar do tempo [9],[33].

Existem duas espécies gerais de agentes no Amalthaea: os agentes de filtragem de informação (FI) e os agentes de descobrimento de informação (DI). Os agentes de filtragem de informação são responsáveis pela personalização do sistema e por não perder de vista os interesses dos usuários. Os agentes de descobrimento de informação são responsáveis pelos recursos de informação: manusear, adaptar-se às fontes de informação, achar e trazer as informações atuais em que o usuário está interessado.

O Almalthaea é inspirado na abordagem de vida artificial *bottom-up* que resulta em mecanismos de representação e aprendizagem completamente distribuídos. Com o objetivo de entender o comportamento global do sistema de filtragem, tem-se que pensar em cada agente de filtragem de informação como um filtro muito especializado que é aplicado somente a um estreito setor do domínio. Quando um usuário muda de interesse, os filtros nomeados para os interesses antigos são destruídos e os novos agentes que são criados são diretamente direcionados para os novos interesses por evolução e seleção natural.

A arquitetura do Amalthaea é composto de cinco partes básicas (**Figura 37**):

- A interface do usuário, onde se apresenta ao usuário as informações recuperadas e o usuário dá *feedback* sobre sua relevância;
- Dois distintos tipos de agentes compõem o ecossistema: agentes de filtragem, que tentam selecionar documentos que preencham as preferências do usuário e agentes de descobrimento da informação, que procuram encontrar na *Web* os *sites* com informações relevantes ao usuário;
- A máquina de interface WWW para recuperar documentos;
- O *stemming*, máquina de geração de palavras chaves com peso indicando o grau de relevância e extração de palavras chave, que realiza o processamento do texto contidos nas páginas e respectivos mecanismos de vetorização de documentos, extração de palavras irrelevantes e remoção de prefixos e sufixos;
- Um banco de dados dos documentos recuperados, onde constam as URLs de todos os documentos considerados relevantes para o utilizador.

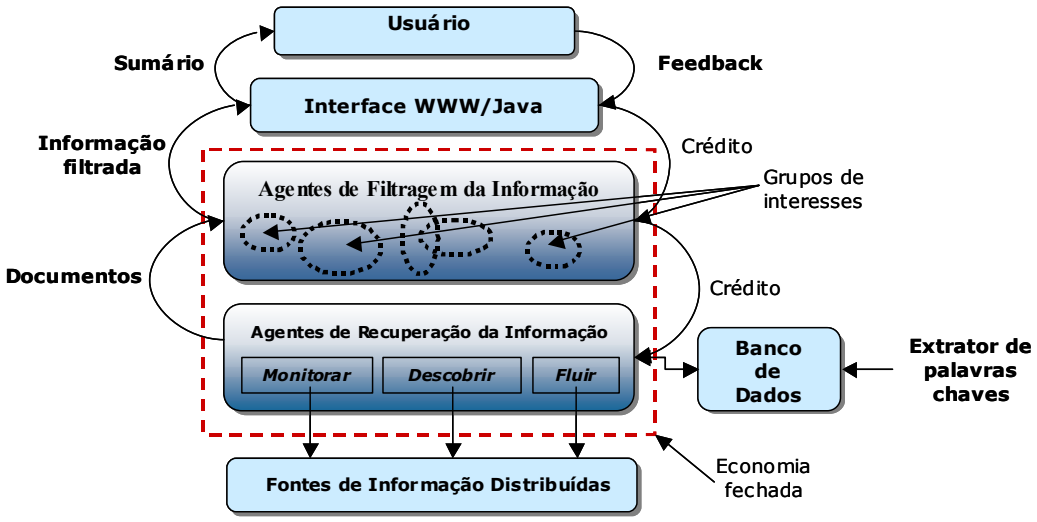

Figura 37-Uma visão geral da arquitetura do AMALTHAEA [33].

5.4.1.1 Representação do documento: vetor de palavras chave

A representação interna de documentos do Almathaea é baseada nas técnicas do modelo do espaço vetorial. Depois de sua recuperação, os documentos são processados pelo Gerador de Vetor de Palavras Chaves ponderadas [41] (*Weighted Keyword Vector Generator* – WKVG) que, como seu nome insinua, transforma os documentos em um vetor de termos com o peso correspondente a sua relevância. Inicialmente, o documento HTML é analisado gramaticalmente e convertido em texto pelo filtro *html2txt*. Os *links* que o documento HTML contém são recuperados usando-se um outro filtro, o *html2url* e são tratados como um tipo especial de palavra chave. Somente algumas características HTML importantes são preservadas, como as marcas de cabeçalho que serão usadas depois na criação dos vetores de palavras chaves ponderadas. O arquivo texto criado é então passado através de um *stemmer*, que remove o sufixo das palavras e mantém somente suas raízes (por exemplo, as palavras "*propor"* e *"proposta"* ambas tornam-se *"propo"*) [9],[33].

Finalmente, o grau de importância de cada palavra chave é estabelecido em função da medida *tfidf*. A sigla *"tfidf"* é a relação entre a freqüência de ocorrência da palavra chave no documento e a freqüência inversa de ocorrência da palavra chave. Neste caso, a coleção de documentos é o conjunto de todos os vetores de palavras chaves ponderados, que são a representação interna dos documentos recuperados [9],[33].

5.4.1.2 Feedback de relevância e interface usuário

O primeiro papel da interface usuário é apresentar a informação recuperada pelo Amalthaea ao usuário. A informação é visualizada em um *browser Java* que se encontra

estruturada, em seções, sendo que cada seção contém os documentos que foram encontrados por um certo grupo de agentes de filtragem da informação. O usuário então pode percorrer as seções e seguir os vínculos hipertexto. À medida que acessa as informações apresentadas, o usuário fornece ao Amalthaea um *feedback* de quão relevantes são os documentos. O *feedback* é usado pelo sistema para suportar o mecanismo de alocação de créditos pelos agentes de filtragem da informação que foram responsáveis pela seleção daquele documento. Um mecanismo de *feedback* de cinco níveis está disponível ao usuário. O usuário pode também escolher um número de palavras chaves do vetor de documentos que melhor descreve determinado documento e os pesos destas palavras chaves serão reforçados [9],[33].

O *feedback* de relevância não é a única maneira que o usuário tem de alterar a dinâmica do sistema. Depois que o sistema é auto-carregado o usuário tem a capacidade de introduzir seus próprios agentes. Isto é possível, provendo uma passagem do texto em que o usuário está interessado ou provendo a URL do documento que ele gostaria. Este documento é analisado gramaticalmente e um novo agente de filtragem da informação é criado baseado nisto. O usuário pode especificar novas URLs, enquanto, em uma outra janela do Amalthaea, analisa as opções sugeridas.

Os agentes que são criados pelo usuário diretamente são considerados agentes de "longo termo" no sentido que eles não são facilmente punidos, mesmo se eles não se desempenham tão bem como os outros agentes.

O usuário pode "ajustar" o Amalthaea a sua necessidade, modificando parâmetros como a quantidade de documentos que o sistema recuperará, a quantidade de documentos que apresentará, o número de agentes descobridores de informação e de agentes de filtragem da informação do sistema. Um outro parâmetro que pode ser ajustado é a evolução e a taxa de mutação do sistema que estão diretamente relacionadas com sua capacidade de adaptação. Se o usuário sentir que seus interesses têm mudado muito e que o sistema não está correspondendo a eles (por exemplo, uma mudança de trabalho), o usuário pode acelerar a evolução do sistema diretamente. Sob circunstâncias normais o sistema é capaz de adaptar-se às mudanças de interesses dos usuários automaticamente.

5.4.1.3 Evolução

A evolução do sistema é baseada em algoritmos genéticos e controlada por duas medidas: seu *fitness* individual e o *fitness* global do sistema. Somente é permitido produzir descendentes a um número variável de agentes da população inteira selecionado daqueles com

as melhores performances. A classificação de um agente é baseada somente no seu *fitness*. O número de agentes que poderão produzir descendentes está linearmente relacionado ao número de agentes que serão punidos por causa do baixo desempenho. Esses números são inconstantes e estão relacionados como o *fitness* global do sistema. Se o *fitness* global está diminuindo então a evolução é aumentada na procura de adaptar-se mais rapidamente aos novos interesses do usuário. Se o *fitness* global está aumentando a evolução é uma taxa constante configurada para permitir que o sistema explore lentamente o espaço de procura por melhores soluções [9],[33].

Os novos agentes são criados por *crossover* e/ou *mutação*. Ambos os operadores são aplicados à parte evolutiva dos agentes, o genótipo. A outra parte dos agentes, o fenótipo contém informações não evolutivas, usualmente instruções de como controlar a parte evolutiva.

5.4.2 Tipos de agentes no AMALTHAEA

Os **agentes de filtragem** de informação estão baseados em um vetor de palavras chaves, (que é a maior parte de seu genótipo) derivadas do processo WKVG descrito anteriormente. Os vetores WKVG são usados para avaliar a similaridade de dois documentos, assim como o casamento entre um agente de filtragem da informação e um documento particular. Esses vetores são acrescidos de outras informações tais como: o autor do documento, se ele foi criado pelo usuário explicitamente ou não, etc. Caso o agente tenha sido criado pelo usuário, os documentos propostos por este agente são tratados mais favoravelmente. De certo modo, cada agente de filtragem da informação especializa-se em uma coleção específica de documentos.

Os agentes descobridores apresentam documentos aos agentes de filtragem. Cada agente de filtragem seleciona um documento que está mais próximo ao seu vetor e calcula o quanto aquele documento específico interessará ao usuário. Nesta abordagem descentralizada cada agente acredita que é um modelo perfeito do usuário. Assim, se um documento combinar com seu vetor completamente, a confiança do agente é 1.0 [9],[33].

Nem todos os documentos introduzidos pelos agentes de filtragem farão parte do sumário. O sistema decide se o agente apresentará alguma coisa para o usuário, classificando os documentos propostos.

Cada agente de filtragem de informação distribui "contratos" aos agentes descobridores da informação sobre o tipo de documentos que eles estão interessados em encontrar. Os agentes descobridores, então, selecionam que contratos eles querem pegar.

Um **agente descobridor** de informação é baseado em um genótipo que contém informações sobre um número de palavras chaves que eles poderiam utilizar quando questionassem as máquinas de indexação *WWW*. O objetivo é criar um corpo diverso de agentes que procurarão por documentos através de diferentes máquinas de busca. O fenótipo dos agentes descobridores inclui um pequeno histórico das transações com diferentes agentes de filtragem e o quanto os agentes de filtragem executaram aquelas transações de forma satisfatória. Dependendo deste histórico, eles criam uma lista enfileirada de créditos ordenados de todos os agentes de filtragem para os quais eles trabalharam e tentam pegar "contratos" dos mais ordenados para maximizar suas próprias aptidões. Desta maneira, os agentes descobridores compreendem que agentes de filtragem eles melhor servem, baseados na máquina de indexação *WWW* que usam e na escolha das palavras chaves que eles requisitam [9],[33].

Os agentes descobridores da informação que estão monitorando *sites* operam um pouco diferentemente dos agentes descobridores especializados *na WWW* e no fluxo de informação estável. Eles não usam nenhuma máquina de busca como mediador, mas, em vez disso, visitam diretamente o *site* de interesse e analisam o documento para descobrir se ele tem mudado e o quanto, usando o banco de dados onde as URLs são armazenadas.

Todas as URLs recuperadas são mantidas em um banco de dados na forma de vetores de palavras chaves com peso. Elas são acompanhadas de um *checksum* que habilita os agentes descobridores para saber se uma URL especifica mudou desde a última visita, neste caso ela é um *site* monitorado. Uma lista separada dos *links* contidos em cada URL é gerada para auxiliar o monitoramento de certos *sites* que atuam como "pontos de partida" na exploração da Internet.

Finalmente um banco de dados é usado para manter um rastro (pista) das URLs já incluídas em sumários anteriores para prevenir a apresentação de informações duplicadas ao usuário.

5.4.3 Coordenação e organização dos agentes

Os agentes que compõem o ecossistema operam sob a estratégia de penalidade/recompensa, suportada pela noção de "crédito" que é designado indiretamente pelo usuário, dependendo da performance do sistema. O usuário dá *feedback* de acordo com o seu nível de interesse por um documento do sumário. O sistema relaciona este *feedback* para o agente de filtragem que propôs o documento e para o agente descobridor que o recuperou e, então, atribui o crédito. O crédito serve como função de aptidão em ambas as populações que evoluem separadamente. Quanto maior os créditos que um agente possui, mais chances ele tem de produzir descendentes.

Se o *feedback* do usuário é positivo, então o agente de filtragem de informação que propôs o documento é premiado com uma certa quantidade de créditos diretamente relacionado com seu nível de confiança na relevância do documento proposto. Se um agente está confiante de que o usuário gostará do documento que ele propôs, então ele recebe crédito positivo. Entretanto, se eles forem muito confiantes de que um documento será de interesse do usuário e o *feedback* é negativo, então eles recebem muitos créditos negativos, o que é ruim para sua aptidão. Agentes de filtragem da informação "pagam" uma porcentagem da quantidade de créditos que eles recebem aos agentes descobridores da informação cujas saídas eles usaram.

É evidente que nem todos os agentes são capazes de recuperar informação para cada sumário. Os documentos propostos pelos agentes são enfileirados pelo nível de confiança e somente os mais relevantes são selecionados (os que se encontram no topo) e apresentados ao usuário. Com o objetivo de acelerar a destruição de agentes não competentes e a evolução de novos agentes, uma função de crédito de decadência linear que pode ser vista como um tipo de "aluguel" é implementado pelo Amalthaea. Para que os agentes habitem o ecossistema, eles têm que pagar uma espécie de aluguel. Se os créditos que eles ganham excedem este "aluguel", então eles continuam pertencendo ao sistema. Caso contrário, eles são removidos e novos agentes são criados. Além disso, se dois agentes propõem o mesmo documento então eles recebem penalidade, através da *função de penalidade,* com o objetivo que isto seja evitado, de forma a aumentar a diversidade da população. Os agentes de filtragem de informação recebem avaliações diretamente do usuário, dependendo de suas performances. Eles, em troca, fornecem alguns créditos para os agentes descobridores da informação que os ajudaram a localizar e recuperar a informação avaliada pelo usuário [9].

O número de agentes que são destruídos em cada ciclo do sistema não é fixo, ao invés disso, ele é baseado em uma quantidade de *feedbacks* positivos do usuário que é dada pelo sistema. Se o *feedback* negativo global do sumário corrente é maior que a média dos últimos dez sumários (um parâmetro definido pelo usuário) então a porcentagem de agentes destruídos aumenta. Se, por outro lado, existem mais *feedbacks* positivos, este número diminui. Até mesmo no cenário do melhor caso de muitos *feedbacks* positivos, existe também uma porcentagem da população que é destruída, com o objetivo de que novos agentes explorem o espaço de busca por novos possível interesses do usuário.

O fluxo de requisições e informações no Amalthaea é a seguinte [9],[33]:

- Os agentes de filtragem de informação enviam requisições ou anunciam "contratos" de documentos aos agentes descobridores na forma de um vetor de palavras chave, fazendo assim o papel de coordenador do ecossistema;
- Os agentes descobridores selecionam as requisições dos agentes de filtragem e, baseados em seus fenótipos, tentam recuperar documentos relevantes utilizando a máquina de indexação *WWW* para os quais eles são nomeados;
- Cada agente descobridor apresenta para seu agente de filtragem um conjunto de documentos recuperados e o agente de filtragem seleciona o que melhor combina com seu vetor de palavras chaves;
- Alguns agentes de filtragem incluem os documentos selecionados no sumário do usuário baseados na sua confiança na relevância dos documentos.
- O usuário avalia os documentos que lhe são apresentados, assim dando o *feedback* de relevância para o sistema.

Finalmente, vale notar que o fluxo de informação padrão, às vezes, não é o convencional: em vez dos agentes descobridores da informação apresentarem dados para os agentes de filtragem da informação, os agentes descobridores da informação são instruídos pelos agentes de filtragem da informação para o tipo de informação a procurar e recuperar no caso do domínio WWW [9],[33].

5.4.4 Estrutura interna dos agentes no AMALTHAEA

A estrutura interna dos agentes no AMALTHAEA e composta por duas partes, o genótipo e fenótipo.

O genótipo é parte do agente que pode evoluir. Entretanto, o fenótipo dos agentes contém a parte não evolutiva do agente como sua aptidão, o campo de interesse em longo prazo e os comandos que habilitam os agentes para se comunicarem uns com os outros e com o sistema. Essencialmente o fenótipo assemelha-se a um molde fixo que é "preenchido" com a informação do genótipo e, então, "executado".

O genótipo dos agentes de filtragem de informação é essencialmente um vetor de palavras chaves ponderadas.

5.4.5 Identificação e análise de padrões no AMALTHAEA

Como mostra a **Figura 37**, a arquitetura AMALTHAEA é organizada em camada para organizar sua cadeia hierárquica entre os agentes participantes da arquitetura(agente de interface, agente de filtragem e agente de recuperação).

A **Figura 38** mostra as camadas identificadas na AMALTHAEA e os fluxos de cooperação entre as camadas.

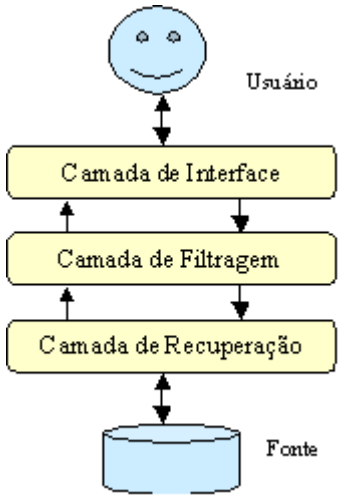

Figura 38-Arquitetura AMALTHAEA na visão padrão camada

A **Tabela 16**, mostra os padrões que foram identificados na arquitetura AMALTHAEA. Esses padrões serão analisados segundo os mecanismos de coordenação e cooperação.

Arquitetura	Padrão
AMALTHAEA	Camada
	Mecanismo de Mercado
	Reunião

Tabela 16-Padrões identificados na arquitetura AMALTHAEA.

5.4.5.1 Padrão camada

A arquitetura AMALTHAEA é dividida em três camadas. Porém, somente as camadas de filtragem e de recuperação são compostas por agentes.

A camada de filtragem é composta por agentes de filtragem que tem o objetivo de filtrar as informações enviadas pelos agentes descobridores segundo o perfil do usuário do sistema, previamente estabelecido.

A camada de recuperação é composta por agentes descobridores de informação. Esses agentes podem assumir diferentes papéis na sociedade: monitor, descobridor e de fluxo contínuo.

A comunicação entre os agentes nesta arquitetura ocorre de forma direta entre as camadas de interface e filtragem e entre as camadas de filtragem e recuperação. As camadas de interface e recuperação se comunicam de forma indireta através da camada de filtragem, proporcionando desta maneira maior capacidade de reuso e fácil manutenção.

Cooperação

O usuário informa suas necessidades através da camada de interface, neste momento são criados agentes de filtragem que colecionam informações do usuário, depois enviam requisições para os agentes descobridores que tentam recuperar documentos relevantes usando a máquina de indexação.

Coordenação

A coordenação nesse padrão fica a cargo do agente de filtragem que é responsável em receber dos agentes descobridores documentos recuperados e envia-os para o sumário do usuário baseado na sua confiança na relevância dos documentos. A camada de filtragem realiza a supervisão direta das demais camadas.

5.4.5.2 Padrão Mercado

Como a arquitetura AMALTHAEA utiliza agentes que simulam um ecossistema que trabalha sob a estratégia penalidade/recompensa, suportando a noção de crédito e realizando compra e venda de serviços, pode-se observar que, esta arquitetura utiliza o padrão mercado onde alguns agentes assumem o papel de vendedores e outros de compradores.

Os agentes de filtragem compram serviços dos agentes descobridores. O pagamento depende da satisfação dos usuários com as informações recuperadas e filtradas. A

satisfação vem na forma de créditos positivos ou negativos. Esses créditos demonstram o grau de evolução de cada agente e também são utilizados para pagar uma espécie de aluguel para continuar habitando o ecossistema.

Os agentes de filtragem possuem os papéis de vendedor e comprador (**Figura 39**). Já os agentes descobridores possuem o papel de vendedor.

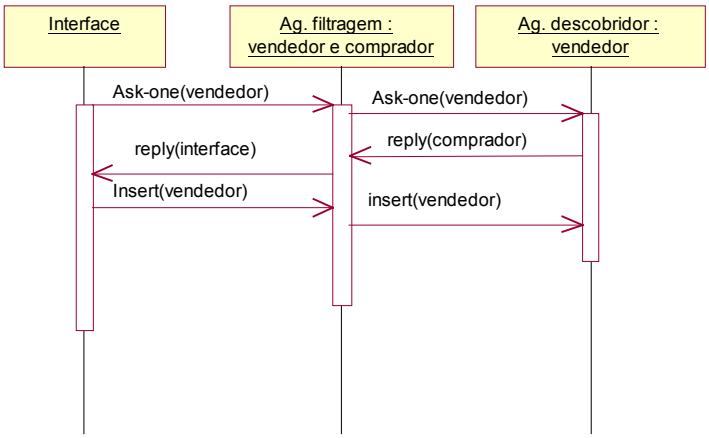

Figura 39-Agente de filtragem com os papéis do padrão mercado.

Cooperação

A cooperação nessa arquitetura é indispensável, pois há um preço tanto a pagar quanto a receber. Em virtude disso, o ecossistema trabalha com a noção de crédito, o sucesso da cooperação é recompensado com créditos positivos e o fracasso com créditos negativos.

O uso do padrão mecanismo de mercado no agente de filtragem e descobridor, promove o aumento da capacidade de comunicação transformando agentes ativos ou passivos, em agentes interlocutores que têm maior poder de cooperação.

Coordenação

A coordenação é de responsabilidade do agente de filtragem que seleciona o papel devido (vendedor ou comprador) e os documentos relevantes para entrega posterior ao usuário final.

5.4.5.3 Padrão Reunião

Na arquitetura AMALTHAEA observa-se que os agentes participantes trabalham em um ecossistema (o lugar de reunião) onde existe a troca de informações e a livre negociação. O ecossistema assemelha-se muito ao padrão reunião. Na **Figura 40** podemos observar o processo de interação dos agentes com o ecossistema. No ecossistema existem

vários papéis incluídos no padrão reunião tais como: o gerente da reunião, a reunião e o próprio ambiente.

A interface interage com o ecossistema para a criação dos agentes de filtragem e recuperação. A vida desses agentes está condicionada com o pagamento de um "aluguel" para continuar habitando o ecossistema.

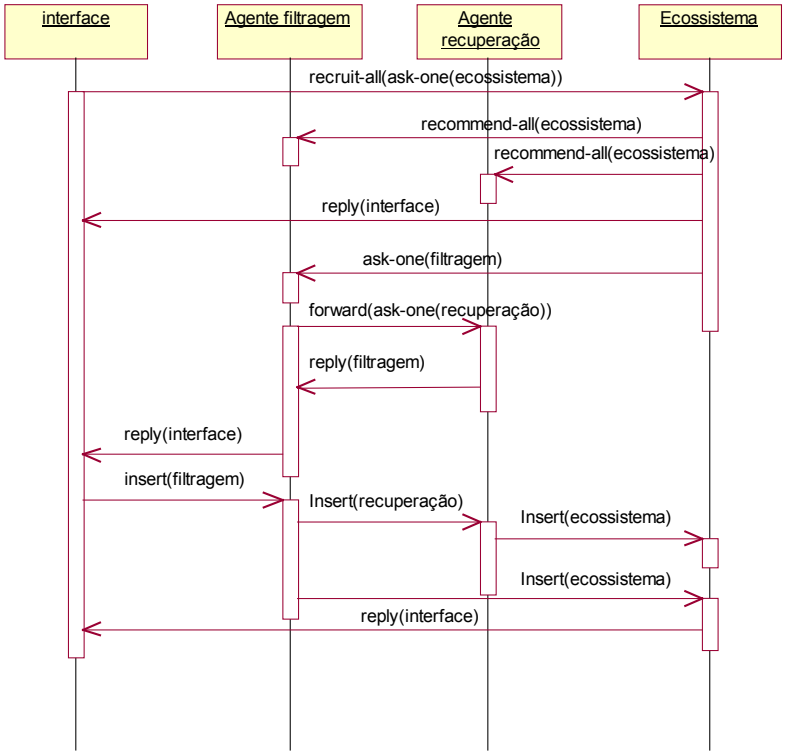

Figura 40-Interação dos agentes com o ecossistema que simula o padrão reunião.

Cooperação

A cooperação no padrão está baseada na negociação dos agentes de filtragem e recuperação com o ecossistema. Essa negociação é condicionada ao pagamento de um aluguel para participar do ecossistema.

Coordenação

A coordenação é de responsabilidade do ecossistema que cria os agentes e permite que eles cooperem entre si.

5.5 ESTUDO DE CASO III: Padrões arquiteturais na ABARFI

5.5.1 Descrição da ABARFI

A arquitetura ABARFI (**A**rquitetura **B**aseada em **A**gentes para a **R**ecuperação e **F**iltragem da **I**nformação) [9] é formada por agentes que trabalham de forma distribuída, cooperando assincronamente entre si para executar a tarefa de recuperar e filtrar informações para o usuário do sistema. O seu desenvolvimento baseou-se nas arquiteturas RETSINA e ALMATHAEA, buscando melhoras e aprimoramentos necessários.

Na construção da arquitetura foi utilizada a Metodologia Baseada em Agentes para o Desenvolvimento de Software (MADS) [46].

5.5.1.1 Objetivos da arquitetura

O principal objetivo de um sistema de filtragem e recuperação de informação é atender às necessidades de informação do usuário, ajudando-o na tarefa de encontrar e filtrar as informações de que necessita. Dependendo de seu tipo, esta necessidade pode ser expressa pelo usuário de duas formas: através de uma requisição feita de forma direta ao sistema, no caso de uma necessidade de informação pontual, o que caracterizaria um sistema de recuperação de informação, ou através da especificação de um perfil ou de um pedido de monitoramento de alguma informação, no caso de uma necessidade de informação à longo prazo, o que caracterizaria um sistema de filtragem de informação. Devido a grande complexidade envolvendo o objetivo geral do sistema, na ABARFI este objetivo foi dividido nos sub-objetivos descritos a seguir:

- Interagir com o usuário – o sistema deve oferecer um meio para o recebimento das requisições ou interesses do usuário e entrega do resultado ao mesmo;
- Modelar estas necessidades de acordo com o interesse do usuário (perfil do usuário) – em se tratando de filtragem de informação, o sistema deve ter algum mecanismo de modelagem do perfil do usuário;
- Executar a requisição – o sistema deve possuir algum mecanismo que garanta e controle a execução das requisições;
- Filtrar informações – para atender as necessidades de informação em longo prazo, o sistema deve possuir mecanismos de filtragem das informações;

- Recuperar informações – para atender as necessidades de informação pontuais, o sistema deve possuir mecanismos para recuperação da informação;
- Estabelecer uma ordem de relevância dos documentos recuperados;
- Monitorar informações – o sistema deve prover um mecanismo para que o usuário possa ter a possibilidade de obter informações atualizadas provenientes de fontes que sejam de sua preferência;
- Indexar fontes de informações – para facilitar o posterior processo de recuperação, o sistema deverá ser equipado com um mecanismo de indexação de fontes de informação;
- Percorrer as fontes de informação – o sistema deverá apresentar algum mecanismos para o acesso às diversas fontes de informação.

Baseando-se nos objetivos que foram propostos em alcançar, existem os papéis que trabalham para realização destes objetivos e sub-objetivos anteriormente citados; os papéis são:

- Interface – serve como um meio de comunicação entre o usuário e o sistema. Tem como funções principais atender as necessidades de informação do usuário, através de uma linguagem de consulta e entregar resultados;
- Modelador – tem como principal objetivo à modelagem dos interesses do usuário para criação do perfil do usuário;
- Coordenador – tem como função principal criar metas e elaborar planos para alcançar estas metas, além de coordenar o sistema e delegar atividades;
- Filtrador – responsável pela filtragem das informações de acordo com os interesses dos usuários;
- Recuperador – responsável por recuperar um conjunto de documentos relevantes a uma consulta do usuário;
- Indexador – cria a representação interna de cada documento obtido de uma fonte de informação selecionada;
- Monitor – monitora informações de interesse do usuário para o caso de ocorrência de mudanças;

- Descobridor – percorre as diversas fontes de informação e seleciona aqueles documentos que serão indexados.

5.5.1.2 Arquitetura da ABARFI

Através da metodologia MADS [46], foram identificados os papéis, entidades externas e os agentes inter-relacionados e baseado nestes pontos foi desenvolvida a arquitetura ABARFI. A **Figura 41** mostra a arquitetura global da ABARFI, que pode ser dividida em cinco camadas, de acordo com os níveis de funcionalidade de cada agente que compõe a arquitetura.

A primeira camada, Camada de Interação com o Usuário, é composta pelos agentes de interface e modelagem. Ela permite que o usuário interaja com o sistema, através de uma consulta, e adquira os resultados desta consulta, além de sugerir informações relevantes ao usuário. Isto tudo é possível, graças aos serviços oferecidos pela segunda camada.

A segunda camada, Camada de Coordenação e Planejamento, é formada pelo agente de coordenação e é responsável pelo planejamento e coordenação das ações a serem tomadas pelo sistema, através da delegação de tarefas que deverão ser executadas pela terceira camada.

A terceira camada, Camada de Execução, é formada pelos agentes de recuperação e filtragem que são responsáveis pela realização das tarefas de recuperação e filtragem dos documentos relevantes à consulta do usuário. Para alcançar suas metas, esta camada utiliza os serviços de indexação da quarta camada.

A quarta camada, Camada de Indexação, é formada pelo agente de indexação e é responsável pela criação do banco de dados contendo a representação dos documentos recuperados pelos agentes que formam a quinta camada.

A quinta e última camada, Camada de Interação com as Fontes de Informação, é formada pelos agentes de monitoramento e descobrimento, que trabalham diretamente com as fontes de informação à procura de documentos que possam ser de interesse dos usuários do sistema.

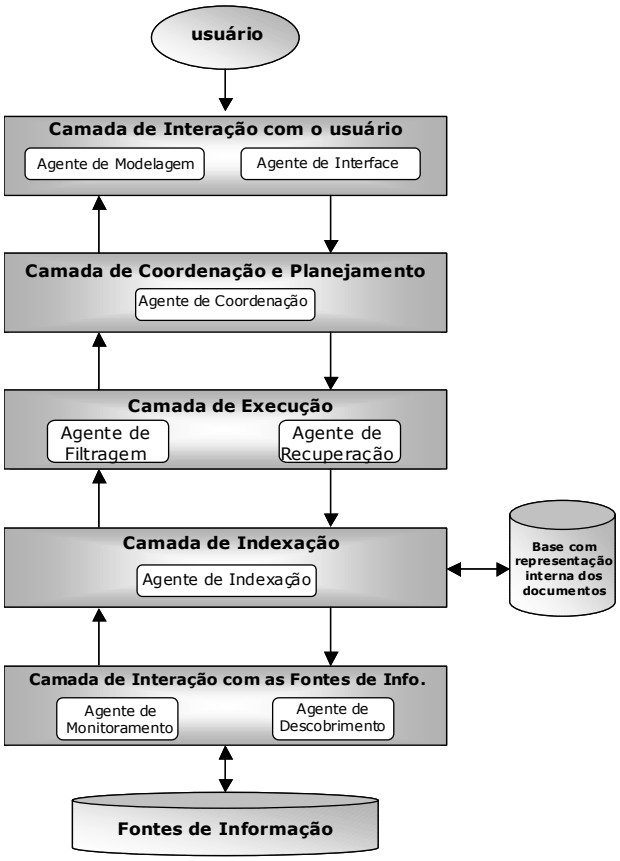

Figura 41-A arquitetura ABARFI [9]

A arquitetura ABARFI é baseada nos conceitos de Recuperação e Filtragem de Informação e no estudo de arquiteturas para o acesso à informação baseadas em agentes já desenvolvidas, como é o caso da RETSINA e AMALTHAEA.

A ABARFI é uma arquitetura global baseada em agentes distribuídos, que colaboram para ajudar o usuário na tarefa de obter informações que sejam relevantes ao seu interesse.

A arquitetura pode ser vista como um sistema multiagente, onde existem dois lados bem distintos dentro do sistema: um que trata do usuário e outro que trata das fontes de informação. Entretanto, estes dois lados têm um mesmo objetivo: satisfazer a necessidade de informação do usuário.

Uma característica importante verificada na arquitetura ABARFI diz respeito à sua reusabilidade. A arquitetura é bem geral e pode ser reutilizada no desenvolvimento de uma grande variedade de aplicações específicas.

5.5.2 Tipos de agentes na ABARFI

Através metodologia MADS, os agentes que irão trabalhar para executar os papéis. Existem 8 (oito) tipos de agentes que integram o sistema, eles são os seguintes:

- Agente de Interface – desempenha o papel da interface – é um agente do tipo reativo – se relaciona com a entidade externa usuário e, há a necessidade de uma única instância deste agente;
- Agente de Modelagem – desempenha o papel do modelador – é um agente do tipo deliberativo – há a necessidade de uma única instância deste agente;
- Agente de Coordenação – desempenha o papel do coordenador – é um agente do tipo deliberativo – há a necessidade de uma única instância deste agente;
- Agente de Filtragem – desempenha o papel do filtrador – é um agente do tipo reativo – há a necessidade de uma ou mais instâncias deste agente, levando-se em consideração que cada agente de filtragem está associado a um usuário e filtra informações relevantes a este;
- Agente de Recuperação – desempenha o papel do recuperador – é um agente do tipo reativo – se relaciona com a entidade externa base de representações comuns dos documentos – há a necessidade de uma única instância deste agente;
- Agente de Monitoramento – desempenha o papel do monitor – é um agente do tipo reativo – se relaciona com a entidade externa fontes de informação – há a necessidade de uma ou mais instâncias deste agente, uma vez que um novo agente monitor deve ser instanciado a cada pedido de monitoramento de uma determinada informação pelo usuário;
- Agente de Indexação – desempenha o papel do indexador – é um agente do tipo deliberativo – se relaciona com a entidade externa base de representações comuns dos documentos – há a necessidade de uma única instância deste agente;
- Agente de Descobrimento – desempenha o papel do descobridor – é um agente do tipo reativo – se relaciona com a entidade externa fontes de informação –

há a necessidade de uma ou mais instâncias deste agente, uma vez que estes percorrem uma grande quantidade de fontes de informação.

Os agentes de Interface, Filtragem, Recuperação, Monitoramento e Descobrimento são classificados como agentes reativos baseando-se no fato de que estes agentes reagem a mudanças de seu ambiente ou a mensagens provenientes de outros agentes. Eles não são capazes de raciocinar sobre as suas intenções. As suas ações se realizam de acordo à seleção de uma regra do tipo <condição, ação> que estabelece a ação a ser executada dada a condição provocada pelo estímulo recebido. Ou seja, estes agentes trabalham com a noção de estímulo/resposta.

Para recuperar uma necessidade específica de informação, o agente de recuperação utilizaria um conjunto de regras de ação, segundo a definição do padrão Agente Reativo. Da mesma forma, o agente de filtragem utilizaria também um conjunto de regras de ação para fazer a filtragem das informações recuperadas.

Os agentes de Modelagem, Coordenação e Indexação são classificados como agentes deliberativos, pois estes agentes são capazes de raciocinar sobre suas intenções e conhecimentos, por isso, podem ser considerados como sistemas de planejamento, podendo selecionar suas metas, criar planos e executá-los, além de poder identificar conflitos.

O agente de modelagem precisaria de um modelo simbólico de raciocínio para modelar, segundo as informações fornecidas pelo agente de interface, os interesses de usuários para a definição do perfil destes usuários.

O agente de coordenação deve ter um comportamento inteligente para que possa interpretar as especificações de usuários e analisar seus perfis e a partir daí identificar metas e formular planos para satisfazer as especificações destes usuários.

Da mesma forma, o agente indexador deve possuir um modelo simbólico de raciocínio para criar uma base de representações comuns das informações recuperadas.

5.5.3 Coordenação e organização dos agentes

A arquitetura ABARFI teve como base a RETSINA e a ALMATHAEA, e buscou suprir algumas das falhas encontradas e utilizou-se de alguns dos pontos de destaque, como por exemplo, o agente de tarefa do RETSINA que serviu como base para a definição do agente de coordenação da ABARFI, e as questões de avaliação e monitoramento existentes no AMALTHAEA serviram como base para a construção do agente de monitoramento.

No caso da ABARFI o agente de coordenação é responsável pela coordenação do sistema, delegando tarefas aos agentes de filtragem e recuperação.

5.5.4 Estrutura interna dos agentes

Em comparação às outras arquiteturas, a ABARFI possui um tipo de agente para cada papel do sistema, o que de certa maneira facilita a reutilização da arquitetura por projetistas de sistemas. Nas outras arquiteturas, o que se pode observar é que alguns agentes acumulam mais de um papel, como é o caso dos agentes de interface e de informação no RETSINA e dos agentes de interface e de descobrimento no AMALTHAEA.

5.5.5 Identificação e analise de padrões

A ABARFI utiliza o padrão camada na disposição de seus agentes. Os agentes são agrupados segundo a finalidade para a qual foram criados (interação usuário, coordenação, indexação e interação forte) (**Figura 42**).

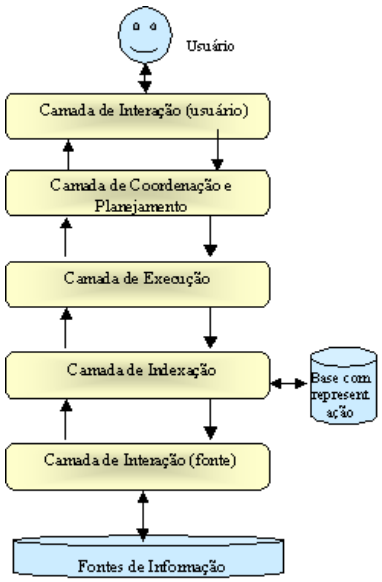

Figura 42-Arquitetura ABARFI disposta em 5 camadas.

A **Tabela 17**, mostra os padrões que foram identificados na arquitetura AMALTHAEA. Esses padrões serão analisados segundo os mecanismos de coordenação e cooperação.

Arquitetura	Padrão
ABARFI	Camada
	Agente negociador
	Mestre-escravo

Tabela 17-Padrões identificados na arquitetura ABARFI.

5.5.5.1 Padrão camada

A utilização do padrão camada nessa arquitetura, também foi utilizada devido a necessidade de dividir todo o sistema em partes menores com o objetivo de facilitar a manuteção e a evolução das camadas.

A arquitetura é dividida em 5 (cinco) subsistema, cada subsistema tem sua importância em desenvolver um papel no sistema. Os subsistemas são composto por diversos agentes:

- Interação (usuário): agentes de modelagem e agentes de interface;
- Coordenação e planejamento: agentes de coordenação;
- Execução: agentes de filtragem e agentes de recuperação;
- Interação (forte): agente de monitoramento e agente descobridor.

Cooperação

A cooperação entre as camadas pode se iniciar de cima para baixo, como de baixo para cima. O fluxo da comunicação depende do momento em que o sistema se encontra.

A comunicação é direta entre as camadas vizinhas e indireta quando utiliza alguma camada para fazer a intermediação.

Segundo a capacidade de interação dos agentes, o padrão utilizado dá suporte as seguintes interações:

- Interação fraca: agente de interface, filtragem, recuperação e monitoramento;
- Interação forte: agente de modelagem, coordenação e indexação.

Coordenação

A coordenação no padrão é feita por uma camada especificamente criada para esse fim. O agente coordenador dessa camada é encarregado de fazer o planejamento de todo o sistema.

5.5.5.2 Padrão Mestre-escravo

O padrão mestre-escravo é utilizado na camada de coordenação e planejamento para delegar tarefas aos agentes participantes da camada de execução. O agente coordenador desempenha o papel de mestre e escravo (Figura 43).

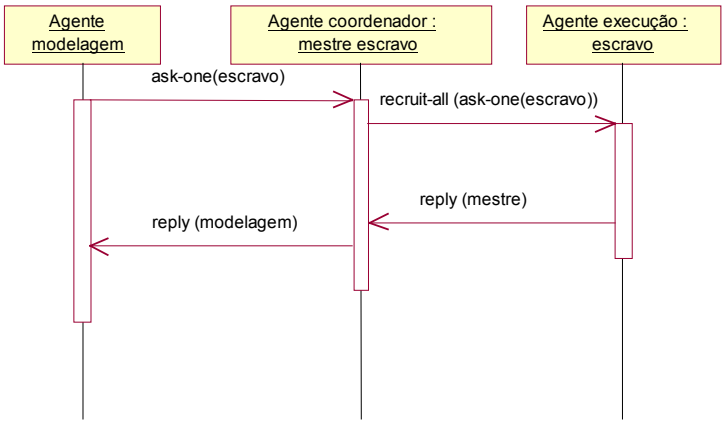

Figura 43- Interação do agente coordenador utilizando os papéis de mestre e escravo.

Cooperação

O mecanismo de cooperação é evidenciado na capacidade do agente coordenador em receber solicitação de serviços, desempenhando o papel de escravo e de delegar esses serviços a outros agentes, desempenhando o papel de mestre. Essa troca de papéis acontece no decorrer das interações do agente coordenador com os outros agentes.

Coordenação

A coordenação dos papéis é feita pelo agente coordenador. Dependendo do momento em que se encontra o sistema, o agente coordenador utilizará o papel apropriado.

5.5.5.3 Padrão agente negociador

O padrão agente negociador é utilizado em agentes mais especializados quando necessitam de tomada de decisão, visando a identificação e solução de conflitos.

A arquitetura ABARFI utiliza esse padrão em alguns agentes para dar maior segurança e coerência nos relacionamento no sistema. Os agentes que usam esse padrão são descritos a seguir:

- Agente de modelagem;
- Agente de coordenação;
- Agente de indexação.

Na **Figura 44** o agente de modelagem faz uma pergunta para o agente de coordenação que inicia uma negociação no papel de iniciador com o agente de indexação no papel de crítico.

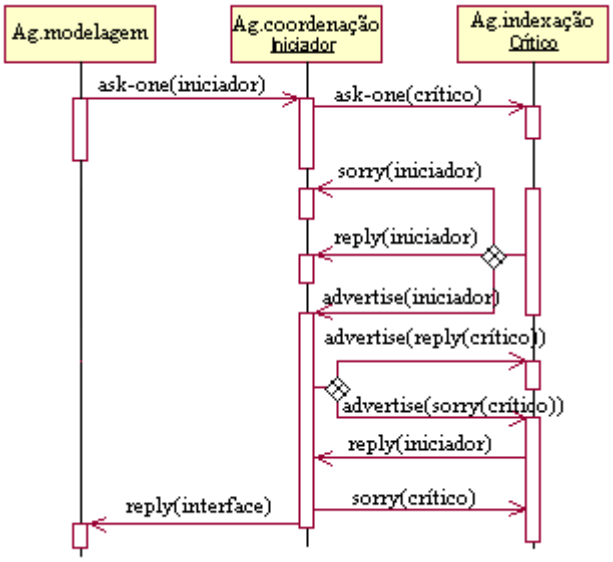

Figura 44- Agente de modelagem utilizando o padrão agente negociador

Cooperação

A cooperação desse padrão é representada na partilha de informações entre os agentes com o objetivo de evitar conflitos.

Coordenação

A coordenação dos papéis de iniciador e crítico é de responsabilidade do agente ao qual os papéis pertencem.

5.6 Conclusões extraídas a partir dos estudos de caso desenvolvidos

Nas arquiteturas analisadas identificaram-se vários dos padrões propostos, dentre eles: o padrão quadro-negro, difusor, federativo, mestre –escravo, camada, agente negociador, reunião e o padrão mercado.

O padrão quadro-negro foi bem empregado na arquitetura RETSINA para auxiliar na organização do acesso as fontes de informação, ou seja, para a coordenar o acesso dos agentes aos repositórios de informação.

A reutilização dos padrões federativo e difusor teriam permitido na RETSINA uma melhor estruturação dos subsistemas que compõem a arquitetura; diminuindo o custo da comunicação entre os agentes provendo assim melhor desempenho do sistema.

O Padrão mestre-escravo foi utilizado na arquitetura RETSINA e na ABARFI para realizar a coordenação das tarefas no padrão camada, suportando o processo de troca de informações entre as camadas.

O padrão camada foi utilizado em todas as arquiteturas analisadas para a hierarquização do processo de recuperação e filtragem de informação, facilitando o reuso e a manutenção das camadas.

O padrão agente negociador foi empregado nas arquiteturas RETSINA e ABARFI integrando o padrão mestre-escravo para suportar o processo de negociação de tarefas entre as camadas das arquiteturas. Esse padrão fornece uma solução para a estruturação do processo de negociação, evitando conflitos entre os diversos agentes na sociedade.

O padrão Mercado quando empregado substitui a utilização do padrão mestre-escravo e do padrão agente negociador porque promove uma estrutura de mercado onde os agentes suportam o trabalho sob a condição penalidade/recompensa. Portanto, na AMALTHAEA que reutiliza este padrão, o usuário tem a possibilidade de recompensar ou punir o agente dependendo do seu desempenho. Este padrão é bem empregado nos sistemas de filtragem e recuperação de informação para construção dos perfis dos usuários.

O padrão reunião foi empregado na AMALTHAEA e utilizado em conjunto com o padrão mercado para criar um ambiente de negociação (reunião) que facilita o processo de interação entre os agentes. No AMALTHAEA o ambiente de negociação é chamado de ecossistema.

Portanto, a análise realizada nos estudos de caso das arquiteturas para recuperação e filtragem de informação permitiram identificar aqueles padrões de particular relevância no desenvolvimento de tais arquiteturas:

- Padrão camada – permite uma adequada estruturação hierárquica dos típicos subsistemas existentes numa arquitetura para filtragem e recuperação de informação (camada de interface com usuário, camada de tarefa, camada de acesso às fontes de informação), facilitando a criação de framework e reutilização de cada camada;
- Padrão federativo – fornece uma solução para a construção de agrupamentos entre os agentes que compõem este tipo de arquitetura (federação de agente de interface, federação agente de tarefa, federação de agentes para o acesso as

fontes de informação). A integração desse padrão com o padrão camada fornece uma solução apropriada para assegurar um bom desempenho da sociedade através dos mecanismos de coordenação e cooperação fornecidos.

- Padrão quadro-negro - fornece uma solução apropriada para o acesso compartilhado às fontes de informação distribuídas pelos agentes de informação neste tipo de arquitetura;
- Padrão mercado – através do mecanismo recompensa/penalidade, fornece uma solução apropriada para atender requisitos típicos deste tipo de arquitetura, como controle de evolução dos modelos de usuário e melhora progressiva da efetividade da recuperação e filtragem;
- Padrão mestre-escravo – integrado ao padrão camada permite estabelecer uma hierarquia de controle entre as camadas que compõem este tipo de arquitetura.
- Padrão difusor – fornece uma solução para a comunicação dos agentes facilitadores (agentes de tarefas) com os agentes os agentes da sua federação e outros agentes facilitadores;
- Padrão reunião – fornece uma solução para a organização do ambiente onde os agentes da sociedade realizaram suas interações e negociações;
- Padrão agente negociador – integrado aos padrões camada e mestre-escravo fornece uma solução para cooperação entre camadas que assumem o papel de mestre e aquelas que assumem o papel de escravo.

CAPÍTULO 6

CONCLUSÃO

CAPÍTULO 6 CONCLUSÃO

Os padrões ajudam a reduzir o custo e melhorar a qualidade das aplicações de software. Eles repassam conhecimento e experiência adquirida em projetos de software anteriores que obtiveram sucesso e documentam partes recorrentes do projeto de software facilitando seu entendimento e aplicação em um contexto particular. Também, eles fornecem um vocabulário comum aos desenvolvedores, facilitando a comunicação entre os projetistas.

A construção de padrões arquiteturais baseados em agentes se beneficia de todas as vantagens e desvantagens dos padrões arquiteturais já existentes baseados na orientação a objetos. Porém, a tecnologia de agentes aborda alguns conceitos que a orientação a objetos não trata como, por exemplo, a utilização de protocolos de comunicação mais sofisticados.

Este trabalho contribui com uma coletânea de padrões para o desenvolvimento de sistemas multiagente descritos utilizando a notação AUML e a KQML como linguagem que dá suporte a cooperação.

6.1 Resultados e contribuição da pesquisa

6.1.1 Descrição de padrões segundo a notação AUML

A notação AUML foi utilizada para a especificação da estrutura dos padrões propostos no nível arquitetural. O diagrama de pacotes e o de interação mostraram sua importância na representação das soluções fornecidas pelos padrões propostos. O diagrama de interação da AUML dá suporte a concorrência e auxilia na especificação dos agentes segundo a noção de papéis, como demonstrado na estrutura do padrão quadro-negro na **Figura 14**.

6.1.2 Demonstração da importância da KQML como linguagem que dá suporte a cooperação e coordenação

Nos padrões propostos, a comunicação entre os agentes é representada utilizando mensagens na linguagem KQML. Essas mensagens codificam informações em três níveis: conteúdo, mensagem e comunicação. Cada nível contém informações que são importantes para o processo de cooperação e coordenação.

A KQML, sendo um protocolo de estar mensagens pré-formatadas com significados e funções bem específicos, não sugere como deve estruturada a arquitetura dos agentes, ela é caracterizada como uma linguagem de propósito geral, independente do ambiente ou estrutura dos agentes. O conjunto de mensagens KQML pode ser ampliado desde

que novas *performatives* sejam criadas obedecendo à especificação original da linguagem. Portanto, como os padrões abordados neste trabalho são padrões arquiteturais nada mais coerente do que representar a cooperação utilizando esta linguagem.

6.1.3 Evolução de padrões

Os padrões arquiteturais abordados neste trabalho demonstraram ter a capacidade de poder integrar-se com outros padrões para resolver problemas maiores, apresentando assim alto nível de flexibilidade e reuso.

A evolução de padrões também pode ajudar na construção de estruturas mais complexas como é o caso dos *frameworks*. Com *frameworks* bem projetados e documentados com o auxilio de padrões fica muito mais fácil fazer extensões, melhorar a manutenção e confiabilidade do software.

Podemos citar como exemplos da evolução de padrões, os frameworks que foram analisados nos estudos de caso apresentados no CAPÍTULO 5 , onde foram identificados diversos padrões arquiteturais, ou seja, boas soluções de projeto.

6.1.4 A utilização dos mecanismos de cooperação e coordenação adequados facilita a construção de aplicações mais eficientes

O mecanismo de coordenação tem a função de organizar os agentes, prevenir o caos e distribuir tarefas e recursos. A cooperação promove a colaboração entre os agentes, objetivando a diminuição do tempo de execução das tarefas. Portanto, os atributos dos mecanismos de coordenação e cooperação são indispensáveis para a construção de aplicações de software eficientes.

6.1.5 Coletânea de Padrões

Depois de identificados e descritos os padrões foram agrupados em uma coletânea. Essa coletânea pode ser considerada um manual de consultas de problemas/soluções que auxilia os desenvolvedores na construção de aplicações baseadas em agentes.

6.1.6 Validação dos padrões propostos

Os estudos de caso desenvolvidos contribuíram com uma primeira avaliação da coletânea de padrões propostos. Neles foram demonstradas as vantagens de construir as arquiteturas dos sistemas multiagente segundo os lineamentos fornecidos pelos padrões

arquiteturais camada, federativo, mestre-escravo, agente negociador, mercado, reunião, difusor e quadro-negro. Por outro lado, a análise realizada nos estudos de caso das arquiteturas para recuperação e filtragem de informação mostraram a particular relevância dos padrões propostos no desenvolvimento de tais arquiteturas.

6.2 Sugestões para trabalhos futuros

6.2.1 Utilização dos padrões propostos

Através dos estudos de caso apresentados no CAPÍTULO 5 , foi avaliada a coleção de padrões proposta. Porém, é necessária uma maior experimentação na aplicação desses padrões no desenvolvimento de sistemas multiagente.

6.2.2 Construção de padrões de projeto detalhado

Os padrões arquiteturais abordados neste trabalho apresentam soluções para o nível de projeto global de um sistema multiagente. Entretanto, o projeto detalhado necessita de um estudo aprofundado para a identificação dos problemas e posterior descrição das soluções referentes ao processamento interno dos agentes. Essas soluções auxiliarão na construção de agentes de qualidade que serão parte do sistema multiagente [38].

6.2.3 Desenvolvimento de uma técnica para o projeto global e detalhado de sistemas multiagente

Para posterior integração dos padrões no nível global e detalhado é necessário à elaboração de uma técnica para o projeto de software de sistemas multiagente que aborde esses dois níveis de abstração. A técnica deveria enfatizar a reutilização de padrões de projeto e permitir a construção de frameworks multiagente reutilizáveis [13].

Sugerimos também a utilização da linguagem KQML para especificação das interações entre os agentes, como foi realizado neste trabalho.

6.2.4 Construção de frameworks reutilizáveis.

A técnica para o projeto de software auxiliará na construção de frameworks utilizando os padrões arquiteturais propostos. Os frameworks reutilizáveis poderão ser agrupados em uma biblioteca de componentes para posterior reuso por desenvolvedores [13].

REFERÊNCIAS

REFERÊNCIAS

[1] Appleton, Brad. *Patterns and Software: Essential Concepts and Terminology*, encontrado no site de URL: http://www.enteract.com/~bradappdocpatterns-intro.html, acessado em 10/10/2002.

[2] Bass,L., Clements, D., Kazman, R., *Software Achitecture in Practice*, 1ed., Addison-Wesley, 1999.

[3] Booch, Grady, *Object-Oriented Analysis and Design with Applications,* (2nd Edition), Addison-Wesley,1999.

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

[5] Coad, Peter. *Object-Oriented Patterns*. Communications of the ACM, V. 35, nº9, p. 152-159, 1992.

[6] Decker, Keith. *Environment Centeres Analysis and Design of Coordination Mechanisms*. PhD thesis, University of Massachusetts, 1995.

[7] Demazeu, Yves; Muller, Jean-Pierre. *Descentralized Artificial Intelligence*. North-Holland: Elselvier Science Publishers, p.3-131, 1990.

[8] Deugo, Dwight; Weiss, Michael; Kendall, Elizabeth. *Reusable Patterns for Agent Coordination*. Computer Systems Engineering, Royal Melbourne Institute of Technology. Australia, 1998.

[9] Diniz, Alessandra. *Uma Arquitetura baseada em Agentes para a Recuperação e Filtragem de Informação*, Dissertação de Mestrado, CPGEE – UFMA, 2001.

[10] Faraco, Rafael Ávila. *Uma Arquitetura de Agentes para Negociação dentro do Domínio do Comércio Eletrônico*. Programa de Pós-Graduação em Engenharia de Produção – Universidade Federal de Santa Catarina, Dissertação de Mestrado, 1998.

[11] Ferber, Jacques. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, ed.Addison-wesley, 1999.

[12] Ferreira, Steferson Lima Costa. *Arquiteturas Baseadas em Agentes em Sistemas de Acesso à Informação*. Monografia do Curso de Graduação em Ciência da Computação. UFMA, 2001.

[13] Ferreira, Steferson Lima Costa. *Uma técnica para o projeto global e detalhado de sistemas multiagente*. proposta de dissertação, CPGEE, UFMA, 2003.

[14] Finin, Tim; Fritzson, Rich; and McKay, Don. *A Language and Protocol to Support Intelligent Agent Iinteroperability*. In Proceedings of the CE and CALS Washington 92 Conference, June 1992.

[15] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns -Elements of Reusable Object-Oriented Software*. Reading-MA, Addison - Wesley, 1995.

[16] Girardi, Maria del Rosario, *Reuse in Agent-based Application Development*,1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS´2002), May 2002 .

[17] Girardi, Maria del Rosário. *Agent-Based Application Engineering*. Proceedings of The III International Conference on Enterprise Information Systems (ICEIS 2001), Vol. I, 123-129, Setúbal, Portugal, 2001.

[18] Girardi, Maria del Rosário. *An Analysis of the Contributions of the Agent Paradigm for the Development of Complex System*, In: Joint Meeting of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001) and the 7th Internatinal Conference on Information Systems Analysis and Systenthesis (ISAS 2001), Orlando, Florida, 2001.

[19] Girardi, María del Rosário. *Software Abstractions in Agent-Based Aplication Engineering*, in: The 5th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, 2001.

[20] 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, acessado em 21/10 2002.

[21] Gurp, J. Van, Bosch, J. Design, *Implementation and Evolution of Object Oriented Frameworks: Concepts and Guidelines*, Revista Software-practice and experiênce, pg. 277-300, 2001.

[22] Hermans, B. *Intelligent Software Agents on the Internet*; http://www.firstmonday.dk/issues/, acessado em 12/10/2002.

[23] Huhns, M. N. e . Stephens, L. M, *Multiagent Systems and Societies of Agents*, em Multiagent Systems – A Modern Approach to Distrbuted Artificial Intelligence, Gerhard Weiss, The MIT Press, London, England, pp. 79-114,1999.

[24] Jacobson, J., Booch, G. And Rumbaugh. *The Unified Software Development Process*, Reading: Addison Wesley, 1999.

[25] Jaques, Patrícia Augustin. *Agente de Software na Monitoração da Colaboração em Ambientes Telemáticos de Ensino*, Dissertação de Mestrado em Informática-PUCRS, Porto Alegre, 1999.

[26] Jennings, Nicholas R. *Coordination Techniques for Distributed Artificial Intelligence*, in O'Hare G M P and Jennings N R (Eds): Foundations of Distributed Artificial Intelligence, London, Wiley, pp 187-210,1990.

[27] Johnson, Ralph. E, Foote B. Designing Reusable Classes. Jornal of Object Orientend Programming-JOOP, pp: 22-25,1998.

[28] Kaelbling, L. P. *A Situated Automata Approach to the Design of Embedded Agents*. SIGART Bulletin, pp:85-88, 1991.

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

[30] Koskimies, Kai. *Pattern Mining and Pattern Life-Cycle*, encontrado no site de URL: http://www.cs.tut.fi/~ohar/Slides2000/Luento5/sld032.htm, acessado em 12/09/2002.

[31] 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, Agosto-Setembro 1988.

[32] Maldonado, José Carlos, Braga, Teresinha Vaccare, Germano, Fernão Stella Rodrigues, Masiero, Paulo César. *Patterns and Frameworks of Software*. Notas didáticas, USP, São Paulo, 2000.

[33] Moukas, Alexandros and Maes, Pattie. *Amalthaea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem*. Proceedings of the Conference on Practical Applications of Agents and Multiagent Technology,1996.

[34] Moulin, Bernard; Chaib-Draa, Brahim. *An Overview of Distributed Artificial Intelligence*. In: O'HARE, Greg; Jennings, Nicholas R. (Eds.). Foundations of Distributed Artificial Intelligence. [S.l.]: John Wiley and Sons. cap.1, 1996.

[35] Odell, James; Parunak,H.Van Dyke e Bauer,Bernhard. *Extending UML for Agentes*, paper, 1999.

[36] Oliveira, F. *Inteligência Artificial Distribuída*. In: IV Escola Regional de Informática, SBC, SC, 1996.

[37] Oliveira, Ismênia Ribeiro e Girardi. *Uma Análise de Padrões de Projeto para o Desenvolvimento de Software Baseado em Agentes*. Monografia de Graduação, UFMA, DEINF, 2001.

[38] Oliveira, Ismênia Ribeiro. *Padrões de Projeto baseados em agentes para a Modelagem de usuários*. proposta de dissertação, CPGEE, UFMA, 2003.

[39] Paraiso, Emerson C. *Concepção e Implementação de um Sistema Multiagente para Monitoração e Controle de Processos Industriais*. Curitiba: Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial - CEFET - PR, Dissertação de Mestrado,1997..

[40] Russell, S. and Norvig, P. *Artificial Intelligence:a Modern Approach*. Prentice-Hall, 1995.

[41] Salton, G. and McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[42] Shaw, M., Garlan, D., *An Introduction to Software Architecture*. In: V.Ambriola and G. Tortora (eds), Advances in Software Engineering and Knowledge, Series on Software Engineering and Knowledge Engineering, Vol 2 World Scientific Publishing Company, pp 1-39, 1993.

[43] Shaw, Mary e Garlan, David. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, upper Saddle River, New Jersey, 1996.

[44] Shehory, Onn and SYCARA, Katia. *The Retsina Communicator*. In Proceedings of Autonomous Agents, Poster Session, 2000.

[45] Simmons, Reid. *Structured Control for Autonomous Robots*. IEEE Journal of Robotics and Automation, 1994.

[46] Sodré, Alidia. *Metodologia Baseada em Agentes para o Desenvolvimento de Softwares – MADS*. Dissertação de Mestrado. Universidade Federal do Maranhão. São Luís, 2001.

[47] Steels, L. *Cooperation between Distributed Agents through sell Organization*. In Y. Demazeau and J. P. Müller, editors, Descentralized AI – Proceeding of the First European Wrokshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89). Pages 175-196. Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, 1990.

[48] Sycara, Katia e Zeng, Dajun. *Coordination of Multiple Intelligent Software Agents*. International Journal of Cooperative Information Systems, 1997.

[49] Sycara, Katia., Decker, K., Pannu, A., Williamson, M. and Zeng, D. *Distributed Intelligent Agents*. IEEE Expert, Dec., 1996.

[50] Vlissides, J., Coplien, J., Kerth, N . *Pattern Languages of Program Design 2*. Reading-MA; Addison-Wesley, 1996.

[51] Weiss, Michael. Patterns for e-Commerce Agent Architectures: Using Agents as Delegates, PLoP 2001.

[52] Williamson, M., Decker, K. And Sycara, K. *Unified Information and Control Flow*. In Proceedings of the AAAI-96 Workshop on Theories of Action, Planning and Control: Bridging the Gap, Portland, Oregon, AAAI, Agosto, 1996.

[53] Witt, Bernard, Baker, F. Terry, Merritt, Everett W. *Software Architecture and Design: Principles, Models and Methods*. VNR Computer Library, 1994.

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

[55] Xavier, José Ricardo. *Criação e Instanciação de Arquiteturas de Software Específicos de Domínio no Contexto de uma Infra –estrutura de Reutilização* dissertação de mestrado, COPPE/UFRJ, 2001.